Computer Algebra 2

—

Fast polynomial arithmetic and factorization

Thibaut Verron,
based on previous lecture notes by Manuel Kauers

February 28, 2019

1 Notations and conventions

Unless otherwise mentioned, we use the following notations:

- $k, K, \mathbb{K}$ are (commutative) fields
- R is a (commutative, with 1) ring

Given a ring R, R^* is the group of its invertible elements.

We assume that algebraic computations (sum, inverse, test of 0, test of 1, inverse where applicable) can be performed.

For a vector v in a vector space V of dimension n, we denote its coordinates by $(v_0, \ldots, v_{n-1})$. If f is a polynomial of degree $\deg(f) = d$, its coefficients are denoted $f_0, \ldots, f_d$, such that

$$f(X) = f_0 + f_1 X + \cdots + f_d X^d = \sum_{i=0}^{d} f_i X^i.$$

In order to simplify notations, we may at times use the convention that $f_i = 0$ if $i < 0$ or $i > \deg(f)$, so that

$$f = \sum_{i \in \mathbb{Z}} f_i X^i.$$

By convention, the degree of the 0 polynomial is $-\infty$.

The logarithm log, without a base, is in base 2.

Definition 1.1. Given two functions $f, g : \mathbb{N} \to \mathbb{R}_{>0}$

$$\begin{aligned} f = O(g) &\iff \frac{f(n)}{g(n)} \text{ is bounded when } n \to \infty \\ &\iff \exists c \in \mathbb{R}_{>0}, n_0 \in \mathbb{N}, \forall n \geq n_0, f(n) \leq c g(n); \end{aligned}$$

$$f = \tilde{O}(g) \iff \exists l \in \mathbb{N}, f = O(g \log(g)^l).$$

1.1 Exercises

Exercise 1.1. Show that the "when $n \to \infty$" clause in the definition of O can be left out. In

other words, given $f, g : \mathbb{N} \to \mathbb{R}_{>0}$, show that

$$\begin{aligned} f = O(g) &\iff \frac{f(n)}{g(n)} \text{ is bounded} \\ &\iff \exists c \in \mathbb{R}_{>0}, \forall n \in \mathbb{N}, f(n) \leq cg(n) \end{aligned}$$

2 Semi-fast multiplication

In this chapter, let R be any ring.

Given $f, g \in R[X]$ with degree less than n, we want to compute the coefficients of $h = f \cdot g$.

The complexity of the algorithm will be evaluated in number of multiplications and additions in R. Typically, multiplications are more expensive!

2.1 Naive algorithm

Each coefficient h_k $(0 \leq k < 2n)$ can be computed with

$$h_k = \sum_{i=0}^{k} f_i g_{k-i},$$

each costing $O(n)$ multiplications and additions.

The total complexity of the naive algorithm is $O(n^2)$ multiplications and $O(n^2)$ additions.

2.2 Karatsuba's algorithm

Remark 2.1. Linear polynomials can be multiplied using 3 multiplications instead of 4 :

$$(a + bX)(c + dX) = ac + (ad + bc)X + bdX^2$$

with

$$ad + bc = ad + bc + ac + bd - ac - bd = (a + b)(c + d) - ac - bd.$$

This can be used recursively to compute polynomial multiplication faster.

Algorithm 1 Karatsuba

Input: $f = f_0 + \cdots + f_{n-1}X^{n-1}$, $g = g_0 + \cdots + g_{n-1}X^{n-1}$

Output: $h = h_0 + \cdots + h_{2n-1}X^{2n-1}$ such that $h = fg$

1. If $n = 1$, then return $f_0 g_0$
2. Write $f = A + BX^{\lceil n/2 \rceil}$, $g = C + DX^{\lceil n/2 \rceil}$ where all of A, B, C, D have degree $< \lceil \frac{n}{2} \rceil$.
3. Compute recursively:
   - $P = AC$
   - $Q = BD$
   - $R = (A + B)(C + D)$
4. Return $P + (R - P - Q)X^{\lceil n/2 \rceil} + RX^{2\lceil n/2 \rceil}$

Theorem 2.2. *Karatsuba's algorithm multiplies polynomials with $O(n^{\log_2(3)}) = O(n^{1.585})$ multiplications and additions.*

Proof. Let $M(n)$ (resp. $A(n)$) be the number of multiplications (resp. additions) in a run of Algo. 1 on an input with size n. Then:

$$M(n) = 3M(n/2)$$

and

$$A(n) = 3A(n/2) + O(n)$$

so $M(n) = O(n^{\log_2(3)})$ and $A(n) = O(n^{\log_2(3)})$. □

Remark 2.3. Karatsuba's algorithm hides an evaluation/interpolation mechanism:

$$\begin{aligned} a &= (a + bX)_{X=0} \\ a + b &= (a + bX)_{X=1} \\ b &= \left(\frac{a + bX}{X}\right)_{X=\infty} \end{aligned}$$

and for two linear polynomials f, g, if $fg = h = h_0 + h_1X + h_2X^2$, we have

$$\begin{aligned} &f(0)g(0) = h(0) = h_0 \\ &f(1)g(1) = h(X = 1) = h_0 + h_1 + h_2 \\ &\left(\frac{f}{X}\right)_{X=\infty}\left(\frac{g}{X}\right)_{X=\infty} = \left(\frac{h}{X^2}\right)_{X=\infty} = h_2 \end{aligned}$$

2.3 Toom-k algorithm

For the remainder of this section, assume that the ring R is an infinite field.

In general the coefficients of h can be obtained as a linear combination of $f(i)g(i)$ for $i \in \{0, \ldots, 2n-1\}$ via

$$\begin{pmatrix} h_0 \\ h_1 \\ h_2 \\ \vdots \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & \ldots \\ 1 & 1 & 1 & \ldots \\ 1 & 2 & 4 & \ldots \\ \vdots & \vdots & \vdots & \ddots \end{pmatrix}^{-1} \left[\begin{pmatrix} 1 & 0 & 0 & \ldots \\ 1 & 1 & 1 & \ldots \\ 1 & 2 & 4 & \ldots \\ \vdots & \vdots & \vdots & \ddots \end{pmatrix} \begin{pmatrix} f_0 \\ f_1 \\ f_2 \\ \vdots \end{pmatrix} \odot \begin{pmatrix} 1 & 0 & 0 & \ldots \\ 1 & 1 & 1 & \ldots \\ 1 & 2 & 4 & \ldots \\ \vdots & \vdots & \vdots & \ddots \end{pmatrix} \begin{pmatrix} g_0 \\ g_1 \\ g_2 \\ \vdots \end{pmatrix} \right]$$

where $\odot$ is the component-wise multiplication of two vectors.

This suggests the following generalization of Algo. 1 for any fixed $k \geq 2$. First, let $V = \left(i^j\right)_{i,j=0}^{2k-1}$ (Vandermonde matrix), and precompute V^{-1}.

Algorithm 2 Toom-k

Input: $f = f_0 + \cdots + f_{n-1}X^{n-1}$, $g = g_0 + \cdots + g_{n-1}X^{n-1}$

Output: $h = h_0 + \cdots + h_{2n-1}X^{2n-1}$ such that $h = fg$

1. If $n < \max(k, 16)$, compute h naively and stop # Forget the "16" until Sec. 2.4
2. Write $f = F_0 + F_1X^{\lceil n/k \rceil} + \cdots + F_{k-1}X^{(k-1)\lceil n/k \rceil}$ and $g = G_0 + G_1X^{\lceil n/k \rceil} + \cdots + G_{k-1}X^{(k-1)\lceil n/k \rceil}$ where $\deg(F_i)$ and $\deg(G_i) < \frac{n}{k}$
3. Define $F_i = G_i = 0$ for $\frac{n}{k} < i \leq 2k-1$
4. Compute $\bar{f} = V\begin{pmatrix} F_0 \\ F_1 \\ \vdots \\ F_{2k-1} \end{pmatrix}$ and $\bar{g} = V\begin{pmatrix} G_0 \\ G_1 \\ \vdots \\ F_{2k-1} \end{pmatrix}$
5. Compute $\bar{h} = \bar{f} \odot \bar{g}$ recursively
6. Return $V^{-1}\bar{h}$

Remark 2.4. If we write $F_i = f_0^{(i)} + \cdots + f_d^{(i)}X^d$ for $i \in \{0, \ldots, k-1\}$, one can compute the product $V \cdot (F_i)$ as

$$V \cdot \begin{pmatrix} F_0 \\ F_1 \\ \vdots \\ F_{k-1} \end{pmatrix} = V \cdot \left[\begin{pmatrix} f_0^0 \\ f_0^{(1)} \\ \vdots \\ f_0^{(k-1)} \end{pmatrix} + \begin{pmatrix} f_1^0 \\ f_1^{(1)} \\ \vdots \\ f_1^{(k-1)} \end{pmatrix} X + \cdots + \begin{pmatrix} f_d^0 \\ f_d^{(1)} \\ \vdots \\ f_d^{(k-1)} \end{pmatrix} X^d \right]$$

$$= V \cdot \begin{pmatrix} f_0^0 \\ f_0^{(1)} \\ \vdots \\ f_0^{(k-1)} \end{pmatrix} + V \cdot \begin{pmatrix} f_1^0 \\ f_1^{(1)} \\ \vdots \\ f_1^{(k-1)} \end{pmatrix} X + \cdots + V \cdot \begin{pmatrix} f_d^0 \\ f_d^{(1)} \\ \vdots \\ f_d^{(k-1)} \end{pmatrix} X^d$$

so the cost of computing that product is $O(dk^2)$.

Theorem 2.5. *A run of Algorithm 2 requires $O(n^{\log_k(2k-1)})$ operations. In particular, for any fixed $\varepsilon > 0$, there exists a multiplication algorithm for $R[X]$ which requires $O(n^{1+\varepsilon})$ operations in R.*

Proof. See Exercise 2.2. □

Remark 2.6. For fixed k, the cost of precomputing V and V^{-1} can be neglected, since it is a fixed cost of $O(k^2)$ and $O(k^3)$ respectively.

2.4 Toom-Cook algorithm

Theorem 2.7 (Toom-Cook). *There exists a multiplication algorithm for $R[X]$ that requires $O(n^{1+2/\sqrt{\log(n)}})$ operations in R. This algorithm is obtained by adapting Algo. 2 to choose at each recursion level $k = \left\lfloor 2^{2\sqrt{\log(n)}} \right\rfloor$.*

Proof. See Exercise 2.3. □

Remark 2.8. This complexity is better than that of Toom-k, since it is better than $O(2^{1+\varepsilon})$ for *all* $\varepsilon > 0$.

Remark 2.9. Strassen's algorithm for matrix multiplication is based on the same idea as Karatsuba's algorithm, and runs in time $O(n^{\log_2(7)}) \leq O(n^{2.82})$. Is there a Toom-Cook style algorithm for matrix multiplication, with complexity better than $O(2^{2+\varepsilon})$ for all $\varepsilon > 0$?

For even k, we can multiply $k \times k$ matrices with $\frac{1}{3}k^3 + 6k^2 - \frac{4}{3}k$ operations, so there are matrix multiplication algorithms with complexity $O(n^{\log_k(\frac{1}{3}k^3+6k^2-\frac{4}{3}k)})$. But $\log_k(\frac{1}{3}k^3 + 6k^2 - \frac{4}{3}k)$ tends to 3 when k tends to ∞. Its minimum (over $2\mathbb{N}$) is reached at $k = 70$, leading to a complexity $O(n^{2.796})$ (Pan's algorithm).

The current record is $O(n^{2.372\,863\,9})$ (Le Gall 2014), and yes, that many decimal points are necessary! It is conjectured that a complexity of $O(2^{1+\varepsilon})$ for all ε is realizable.

Remark 2.10. It is conjectured that polynomial multiplication in $O(n)$ operations is not possible.

2.5 Exercises

Exercise 2.1. Is it possible to use the ideas of the Algorithm of Toom-k with evaluation at $\{0, 1, \ldots, k-2, \infty\}$? Describe the matrices V and V^{-1}.

Exercise 2.2. Prove Theorem 2.5.

Exercise 2.3. Prove Theorem 2.7.

Exercise 2.4. Show that there is no algorithm which can multiply two linear polynomials (over any ring) in 2 multiplications.

3 Fast multiplication in $\bar{k}[X]$

In this chapter, let k be an *algebraically closed* field. The problem to solve is the same as previously, but this time, we assume that $\deg(f) + \deg(g) < n$.

We will be considering evaluation/interpolation methods.

Algorithm 3 Evaluation/interpolation

Input: $f = f_0 + \cdots + f_{k-1}X^k$, $g = g_0 + \cdots + g_{l-1}X^l$ with $k + l < n$

Output: $h = h_0 + \cdots + h_{n-1}X^{n-1}$ such that $h = fg$

1. Fix $(x_0, \ldots, x_{n-1}) \in k^n$
2. Compute $f(x_i)$, $g(x_i)$ for $i = 0, \ldots, n-1$
3. Compute $h(x_i) = f(x_i)g(x_i)$ for $i = 0, \ldots, n-1$
4. Compute h by interpolating $h(x_i)$ for $i = 0, \ldots, n-1$

Remark 3.1. In general, Algo. 3 requires $O(n^2) + O(n) + O(n^2) = O(n^2)$ operations in k, like the classical algorithm. The idea is to choose specific values of $x_0, \ldots, x_{n-1}$ so that steps 2 and 4 can be done faster.

3.1 Roots of unity and discrete Fourier transform

Definition 3.2. An element $\omega \in k$ is called a *n'th root of unity* if $\omega^n = 1$. It is a *primitive* n'th root of unity if additionally $\omega^i \neq 1$ for $0 < i < n$.

Example 3.3. In $\mathbb{C}$, -1 is a primitive second root of unity. i is a primitive 4th root of unity.

In $\mathbb{F}_{17}$, 2 is a primitive 8th root of unity.

Definition 3.4. The matrix

$$\mathrm{DFT}_n := \mathrm{DFT}_n^{(\omega)} := \left(\omega^{ij}\right)_{i,j=0}^{n-1} = \begin{pmatrix} 1 & 1 & 1 & \ldots & 1 \\ 1 & \omega & \omega^2 & \ldots & \omega^{n-1} \\ 1 & \omega^2 & \omega^4 & \ldots & \omega^{2(n-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \omega^{n-1} & \omega^{2(n-1)} & \ldots & \omega^{(n-1)^2} \end{pmatrix} \in k^{n \times n}$$

is called the *discrete Fourier transform* (wrt ω).

Example 3.5. In $\mathbb{C}$, the discrete Fourier transform wrt i is

$$\mathrm{DFT}_4^{(i)} = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & i & -1 & -i \\ 1 & -1 & 1 & -1 \\ 1 & -i & -1 & i \end{pmatrix}.$$

Remark 3.6. The DFT is a Vandermonde matrix. In particular, if $f = f_0 + f_1 X + \cdots + f_{n-1}X^{n-1}$,

$$\mathrm{DFT}_n^{(\omega)} \cdot \begin{pmatrix} f_0 \\ f_1 \\ \vdots \\ f_{n-1} \end{pmatrix} = \begin{pmatrix} f(\omega^0) \\ f(\omega^1) \\ \vdots \\ f(\omega^{n-1}) \end{pmatrix}.$$

Definition 3.7. Let $f, g \in k^n$. The *product* $f \odot g$ is the vector whose i'th coordinate is given by $f_i g_i$. The *convolution* $f * g$ is the vector whose i'th coordinate is given by

$$\sum_{k=0}^{n-1} f_k g_{(i-k) \bmod n}.$$

Lemma 3.8. *Let ω be a primitive n'th root of unity. Then*

1. *there is a factorization*

$$X^n - 1 = (X - \omega)(X - \omega^2) \cdots (X - \omega^n);$$

2. *for any $j \in \{1, \ldots, n-1\}$,*

$$\sum_{i=0}^{n-1} \omega^{ij} = 0.$$

3. *there is a group isomorphism*

$$\left(\{\omega^i : i \in \mathbb{Z}\}, \cdot\right) \simeq (\mathbb{Z}/n\mathbb{Z}, +)$$

4. *the DFT matrix is easy to invert:*

$$\left(\mathrm{DFT}_n^{(\omega)}\right)^{-1} = \frac{1}{n}\mathrm{DFT}_n^{(1/\omega)}$$

5. *if $m \mid n$, then ω^m is a primitive (n/m)'th root of unity*
6. *the DFT is compatible with convolution*

$$\mathrm{DFT}_n(f * g) = \mathrm{DFT}_n(f) \odot \mathrm{DFT}_n(g)$$

Proof. 1. All ω^i are distinct: if $\omega^i = \omega^j$ with $1 \le i < j \le n$, then $\omega^{j-i} = 1$ with $0 < j - i < n$, which is a contradiction because ω is a primitive root of unity. All ω^i are roots of $X^n - 1$, since $(\omega^i)^n = (\omega^n)^i = 1$, so the $X - \omega^i$ are n distinct factors of $X^n - 1$. By comparing the degree and leading coefficient, we get the wanted factorization.

2. Use the formula

$$\left(\sum_{i=0}^{n-1} X^i\right)(X-1) = X^n - 1$$

Evaluated at $X = \omega^j$ for $0 < j < n$, the right hand side is 0, the factor $(\omega^j - 1)$ is non-zero, so the sum has to be zero.

3. Clear.

4. Evaluate the product:

$$\begin{aligned}
\mathrm{DFT}_n^{(\omega)}\mathrm{DFT}_n^{(1/\omega)} &= \left(\omega^{ij}\right)_{i,j=0}^{n-1} \cdot \left(\omega^{-ij}\right)_{i,j=0}^{n-1} \\
&= \left(\sum_{k=0}^{n-1} \omega^{ik}\omega^{-kj}\right)_{i,j=0}^{n-1} \\
&= \left(\sum_{k=0}^{n-1} \omega^{k(i-j)}\right)_{i,j=0}^{n-1} \\
&= \left(n\delta_{ij}\right)_{i,j=0}^{n-1}.
\end{aligned}$$

5. Clear.

6. If we associate the vector $f = (f_0, \ldots, f_{n-1})$ with the polynomial $f(X) = f_0 + \cdots + f_{n-1}X^{n-1}$, convolution is equivalent to multiplication in $k[X]/\langle X^n - 1\rangle$, that is

$$(f * g)(X) = f(X)g(X) + q(X) \cdot (X^n - 1)$$

for some $q \in k[X]$. Indeed, write

$$\begin{aligned}
f(X)g(X) &= \sum_{i,j=0}^{n-1} f_i g_j X^{i+j} \\
&= \sum_{i+j<n} f_i g_j X^{i+j} + \sum_{n\le i+j<2n} f_i g_j X^{i+j} \\
&= \underbrace{\sum_{i+j<n} f_i g_j X^{i+j} + \sum_{n\le i+j<2n} f_i g_j X^{i+j-n}}_{(f*g)(X)} - \underbrace{\sum_{n\le i+j<2n} f_i g_j X^{i+j-n} + \sum_{n\le i+j<2n} f_i g_j X^{i+j}}_{(\sum_{n\le i+j<2n} f_i g_j X^{i+j-n})(X^n-1)}
\end{aligned}$$

The claim follows by evaluation at ω^i. □

The remark, together with property 4, makes powers of ω a good choice for evaluation and interpolation: if we can just find a fast way to evaluate $\mathrm{DFT}_n \cdot f$, we can perform both steps in a fast way.

3.2 Fast Fourier transform

Given $f = \begin{pmatrix} f_0 \\ \vdots \\ f_{2n-1} \end{pmatrix}$, we want to compute $\bar{f} = \mathrm{DFT}_{2n} \cdot f$.

Let's expand the j'th coefficient:

$$\begin{aligned}
\left(\mathrm{DFT}_{2n}^{\omega} f\right)_j &= \sum_{i=0}^{2n-1} \omega^{ij} f_i \\
&= \sum_{i=0}^{n-1} \omega^{2ij} f_{2i} + \sum_{i=0}^{n-1} \omega^{(2i+1)j} f_{2i+1} \\
&= \sum_{i=0}^{n-1} (\omega^2)^{ij} f_{2i} + \omega^j \sum_{i=0}^{n-1} (\omega^2)^{ij} f_{2i+1} \\
&= \begin{cases} \left(\mathrm{DFT}_n^{(\omega^2)} f_{\mathrm{even}}\right)_j + \omega^j \left(\mathrm{DFT}_n^{(\omega^2)} f_{\mathrm{odd}}\right)_j & \text{for } 0 \le j < n \\ \left(\mathrm{DFT}_n^{(\omega^2)} f_{\mathrm{even}}\right)_{j-n} + \omega^j \left(\mathrm{DFT}_n^{(\omega^2)} f_{\mathrm{odd}}\right)_{j-n} & \text{for } n \le j < 2n \end{cases} \\
&= \begin{cases} \left(\mathrm{DFT}_n^{(\omega^2)} f_{\mathrm{even}}\right)_j + \omega^j \left(\mathrm{DFT}_n^{(\omega^2)} f_{\mathrm{odd}}\right)_j & \text{for } 0 \le j < n \\ \left(\mathrm{DFT}_n^{(\omega^2)} f_{\mathrm{even}}\right)_{j-n} - \omega^{j-n} \left(\mathrm{DFT}_n^{(\omega^2)} f_{\mathrm{odd}}\right)_{j-n} & \text{for } n \le j < 2n \end{cases}
\end{aligned}$$

We can use this property to perform the evaluation and interpolation steps.

Algorithm 4 Fast Fourier Transform

Input: $f \in k^n$, ω a primitive n'th root of unity, $n = 2^k$

Output: $\bar{f} = \mathrm{DFT}_n^{(\omega)} f$

1. If $n = 1$ then return (f_0)
2. $u \leftarrow \mathsf{FFT}([f_0, f_2, \ldots,], \omega^2, n/2)$, $v \leftarrow \mathsf{FFT}([f_1, f_3, \ldots,], \omega^2, n/2)$
3. Return $[u_0 + v_0, u_1 + \omega v_1, u_2 + \omega^2 v_2, \ldots, u_{n/2-1} + \omega^{n/2-1} v_{n/2-1},$
   $u_0 - v_0, u_1 - \omega v_1, u_2 - \omega^2 v_2, \ldots, u_{n/2-1} - \omega^{n/2-1} v_{n/2-1}]$

Theorem 3.9. *Algo. 4 requires $O(n \log(n))$ operations in k.*

Proof. Similar to before, with the recurrence

$$T(n) = 2T\left(\frac{n}{2}\right) + O(n).$$

□

This allows us to rewrite Algo. 3 with the FFT.

Algorithm 5 Evaluation/interpolation multiplication using FFT

Input: $f = f_0 + \cdots + f_{k-1}X^k$, $g = g_0 + \cdots + g_{l-1}X^l$ with $k + l < n$

Output: $h = h_0 + \cdots + h_{n-1}X^{n-1}$ such that $h = fg$

1. $\omega \leftarrow$ primitive n'th root of unity
2. $\bar{f} \leftarrow \mathsf{FFT}(f, \omega)$, $\bar{g} \leftarrow \mathsf{FFT}(g, \omega)$
3. $\bar{h} \leftarrow \bar{f} \odot \bar{g}$
4. Return $\frac{1}{n}\mathsf{FFT}(\bar{h}, \omega^{-1})$

Theorem 3.10. *Multiplication in $k[X]$ can be done with $O(n \log n)$ operations in k if k is algebraically closed.*

Remark 3.11. This complexity is currently the best known complexity for polynomial multiplication.

Remark 3.12. Let P be the permutation matrix such that

$$P \cdot f = \begin{pmatrix} f_{\text{even}} \\ f_{\text{odd}} \end{pmatrix}$$

and Δ be the diagonal matrix

$$\Delta = \begin{pmatrix} 1 & & & \\ & \omega & & \\ & & \omega^2 & \\ & & & \ddots \end{pmatrix}.$$

Then the computations above yield that

$$\begin{aligned} \mathrm{DFT}_{2n} &= \begin{pmatrix} \mathrm{DFT}_n & \Delta\mathrm{DFT}_n \\ \mathrm{DFT}_n & -\Delta\mathrm{DFT}_n \end{pmatrix} \cdot P \\ &= \begin{pmatrix} I & \Delta \\ I & -\Delta \end{pmatrix} \cdot \begin{pmatrix} \mathrm{DFT}_n & \\ & \mathrm{DFT}_n \end{pmatrix} \cdot P \\ &= \begin{pmatrix} I & I \\ I & -I \end{pmatrix} \cdot \begin{pmatrix} I & \\ & \Delta \end{pmatrix} \cdot \begin{pmatrix} \mathrm{DFT}_n & \\ & \mathrm{DFT}_n \end{pmatrix} \cdot P \end{aligned}$$

This can be generalized to divisions by m instead of 2. Skipping over the details, this gives

$$\mathrm{DFT}_{mn} = \begin{pmatrix} I & I & I & \dots \\ I & \omega^n I & \omega^{2n} I & \dots \\ I & \omega^{2n} I & \omega^{4n} I & \dots \\ \vdots & \vdots & \vdots & \ddots \end{pmatrix} \cdot \begin{pmatrix} I & & & \\ & \Delta & & \\ & & \Delta^2 & \\ & & & \ddots \end{pmatrix} \cdot \begin{pmatrix} \mathrm{DFT}_n & & & \\ & \mathrm{DFT}_n & & \\ & & \mathrm{DFT}_n & \\ & & & \ddots \end{pmatrix} \cdot P.$$

This is a result due to Cooley and Tuckey, which can be used to refine Algo. 4 so that it reduces an FFT of *any* size quickly to FFT's of prime size.

4 Fast multiplication in $R[X]$

4.1 FFT outside of a field

FFT multiplication (Algos. 4 and 5) does not require that the base ring R be an algebraically closed field, but that:

1. R contains a primitive n'th root of unity ω;
2. $n = 1 + 1 + \cdots + 1$ is invertible.

In this chapter, we will see how to perform FFT without those hypotheses.

This will be done by extending R with roots of unity, *i.e.* by working in rings of the form $R[Y]/\langle Y^k - 1\rangle$. Such rings are not in general fields, so in order to take advantage of the techniques used for the FFT in fields, we need to extend the definition of a primitive root of unity to rings.

Definition 4.1 (Primitive root of unity)**.** Let R be a ring, $\omega \in R$ and $n \in \mathbb{N}$. We say that ω is a *primitive n'th root of 1* if:

- $\omega^n = 1$
- for all $k \leq n$, $\omega^k - 1$ is not a zero-divisor in R (*i.e.* if $x \in R$ is such that $x(\omega^k - 1) = 0$, then $x = 0$).

The second hypothesis is stronger than the corresponding requirement for fields, which was that $\omega^k - 1 \neq 0$. Note that this definition is equivalent to the previous one in the case of fields (or integral domains), because they do not have non-zero zero-divisors.

Proposition 4.2. *Let R be a ring, $\omega \in R$ and $n = 2^k \in \mathbb{N}$ such that ω is a primitive n'th root of 1. Let $f, g \in R[X]$. Then*

- *Fast Fourier Transform (Algo. 4) computes the evaluations of f at $1, \omega, \ldots, \omega^{n-1}$ modulo $\omega^{n/2} + 1$;*
- *FFT multiplication (Algo. 5) computes fg modulo $\omega^{n/2} + 1$.*

4.2 Schönhage-Strassen's algorithm if 2 is invertible

In this section, assume that R has no n'th root of unity, but that $2 \in R^*$.

Let $f, g \in R[X]$, with $\deg f + \deg g < n = 2^k$, as before we want to compute $h = fg$. Write $n = pq$ where $p = 2^{\lceil k/2 \rceil}$ and $q = 2^{\lfloor k/2 \rfloor}$, so $p \simeq q \simeq \sqrt{n}$.

Write

$$f = F_0 + F_1 X^q + F_2 X^{2q} + \dots$$
$$g = G_0 + G_1 X^q + G_2 X^{2q} + \dots$$

with $\deg F_i < q$, $\deg G_i < q$, and define two polynomials in $R[X, Y]$

$$\bar{f} = F_0 + F_1 Y + F_2 Y^2 + \dots$$
$$\bar{g} = G_0 + G_1 Y + G_2 Y^2 + \dots.$$

Then $\deg_X \bar{f}, \deg_X \bar{g} < q$, $\deg_Y \bar{f}, \deg_Y \bar{g} < p$, and $f = \bar{f}(X, X^q)$, $g = \bar{g}(X, X^q)$. Let $\bar{h} = \bar{f}\bar{g}$, then $\deg_X \bar{h} < 2q$ and $\deg_Y \bar{h} < 2p$.

Note 4.3. It suffices to compute $\bar{h} \bmod Y^p + 1$ because

$$\deg h = \deg \bar{h}(X, X^q) < pq = n.$$

Note 4.4. Since $\deg_X \bar{h} < 2q$,

$$\bar{h}(X, Y) = \bar{h}(X, Y) \bmod X^{2q} + 1.$$

Hence, together with the previous note, we can compute in

$$\left(R[X]/\left\langle X^{2q} + 1\right\rangle\right)[Y]/\langle Y^p + 1\rangle.$$

We denote by D the ring

$$D := R[X]/\left\langle X^{2q} + 1\right\rangle.$$

Proposition 4.5. *In the ring D, X is a $4q$'th primitive root of unity. Furthermore, let*

$$\omega = \begin{cases} X^2 & \text{if } p = q \\ X & \text{if } p = 2q. \end{cases}$$

Then ω is a $2p$'th primitive root of unity in D.

With this setting, if

$$\bar{f}(Y) \cdot \bar{g}(Y) = \bar{h}(Y) \bmod Y^p + 1$$

then

$$\bar{f}(\omega Y) \cdot \bar{g}(\omega Y) = \bar{h}(\omega Y) \bmod (\omega Y)^p + 1 = 1 - Y^p$$

Algorithm 6 Schönhage-Strassen

Input: $f, g \in R[X]$ with $\deg f, \deg g < n = 2^k$

Output: $h = fg \bmod X^n + 1$

1. If $k \leq 2$ then compute h directly
2. Define $p, q \in \mathbb{N}$, $\bar{f}, \bar{g} \in D[Y]$ and $\omega \in D$ as above
3. Use Algo. 5 to compute $\bar{h} \in D[y]$ with

$$\bar{h}(\omega Y) = \bar{f}(\omega Y)\bar{g}(\omega Y) \bmod Y^p - 1$$

   using ω^2 as a p'th root of unity in D and Algo. 6 recursively for multiplications in D
4. Return $h = \bar{h}(X, X^q) \bmod X^n + 1$

Remark 4.6. The algorithm requires that 2 be invertible for the FFT step: each call to the FFT multiplication algorithm is with a power of 2 as n.

Theorem 4.7. *Algo. 6 requires $O(n \log(n) \log(\log(n)))$ operations in R.*

Remark 4.8. For all practical purposes, $\log \log n \leq 6$.

Proof. Let $n \gg 1$ and suppose that

$$T(m) \leq c_1 m \log m \log \log m$$

for all $m < n$ and some constant c_1. Recall that $n = 2^k$, $p = 2^{\lceil k/2 \rceil} \leq 2\sqrt{n}$, $q = 2^{\lfloor k/2 \rfloor} \leq \sqrt{n}$.

The runtime function satisfies the recurrence

$$T(n) \leq pT(2q) + O(n \log n)$$

where $pT(2q)$ is the cost of p component-wise multiplication of polynomials of degree at most $2q$, and the trailing $O(n \log n)$ is the cost of the FFT.

Let $T_l(k) = T(2^k)$, and expand in terms of k :

$$\begin{aligned}
T_l(k) &\leq 2^{\lceil k/2 \rceil} T_l\left(\left\lfloor \frac{k}{2} \right\rfloor + 1\right) + c_2 2^k k \\
&\leq c_1 2^{\lceil k/2 \rceil} 2^{\lfloor k/2 \rfloor + 1} \left(\left\lfloor \frac{k}{2} \right\rfloor + 1\right) \log\left(\left\lfloor \frac{k}{2} \right\rfloor + 1\right) + c_2 2^k k \\
&\leq c_1 \underbrace{2^{\lceil k/2 \rceil + \lfloor k/2 \rfloor}}_{=2^k} \cdot \underbrace{2\left(\left\lfloor \frac{k}{2} \right\rfloor + 1\right)}_{\leq k+2} \underbrace{\log\Big(\underbrace{\left\lfloor \frac{k}{2} \right\rfloor + 1}_{\leq \frac{3}{4}k}\Big)}_{\leq \log k - \log(4/3)} + c_2 2^k k \\
&\qquad \underbrace{\qquad\qquad}_{\leq k \log k - k \log(4/3) + 2 \log k - 2 \log(4/3)} \\
&\leq c_1 2^k k \log(k) + \underbrace{c_1 2^k \underbrace{\left(2 \log k - 2 \log(\frac{4}{3})\right)}_{\leq \frac{1/2}{k} \log(4/3)} + (c_2 - c_1 \log(\frac{4}{3})) 2^k k}_{\leq (c_2 - \frac{1}{2}\log(4/3)) 2^k k}
\end{aligned}$$

Without loss of generality we can assume that $c_1 \geq 2c_2/\log 4/3$, so

$$T_l(k) \leq c_1 2^k k \log(k)$$

and indeed

$$T(n) = O(n \log n \log\log n).$$

□

4.3 Schönhage-Strassen's algorithm in the general case

The previous algorithm requires 2 to be invertible in order to divide the reverse DFT by 2^k. Without this assumption, we can skip that division, and Algo. 5 returns $2^k fg$. Analogously, we can compute $3^l fg$ using a 3-adic FFT. Then, Euclid's extended algorithm yields $u, v \in \mathbb{Z}$ such that

$$u \cdot 2^k + v \cdot 3^l = 1,$$

so

$$u \cdot 2^k fg + v \cdot 3^l fg = fg.$$

Theorem 4.9. *Polynomials in $R[X]$ of degree less than n can be multiplied using $O(n \log n \log\log n)$ operations in R, for any commutative ring R with a unity.*

Remark 4.10. This is the current world record.

4.4 Multiplication time function

Definition 4.11. Let R be a ring. A function $M : R \to \mathbb{N}$ is called *multiplication time* for $R[X]$ if there exists an algorithm that multiplies $f, g \in R[X]$ with $\deg f$, $\deg g < n$ using no more than $M(n)$ operations in R.

Finding the best possible M for various rings in an active field of research.

Proposition 4.12. *We can assume that:*

1. *if R is infinite, M is worse than linear:*

$$\frac{M(n)}{n} > \frac{M(m)}{m} \text{ if } n > m;$$

2. *in particular,*

$$M(mn) \geq mM(n)$$

and

$$M(m+n) \geq M(m) + M(n);$$

3. *M is at most quadratic:*

$$M(nm) \leq m^2 M(n)$$

4. *M is at most the complexity of the general algorithm by Schönhage and Strassen:*

$$M(n) = O(n \log n \log\log n).$$

5 Fast multiplication in $\mathbb{Z}$

Here, we are given two *integers* $f, g \in \mathbb{Z}$ with at most n digits (in base 2), and we want to compute $h = fg$.

5.1 Integer multiplication in theory

Remark that if

$$f = f_0 + 2f_1 + \cdots + 2^{n-1}f_{n-1},$$

f is the evaluation of the polynomial

$$\bar{f} = f_0 + f_1 X + \cdots + f_{n-1}X^{n-1}$$

at $X = 2$.

This reduces integer multiplication to polynomial multiplication, with similar complexity results.

Theorem 5.1 (Schönhage-Strassen)**.** *Integers of length n can be multiplied in time* $O(n \log n \log\log n)$.

Remark 5.2. It is conjectured that the lower bound for the complexity of integer multiplication is $cn \log n$.

The current best results are the following.

Definition 5.3. For $x \in \mathbb{R}_{>1}$, the *iterated logarithm* of x is

$$\log^*(x) = \max\{k \in \mathbb{N} : \log^k(x) \leq 1\}.$$

Remark 5.4. For all practical purposes, $\log^*(n) \leq 4$.

Theorem 5.5 (Fürer, 2007)**.** *Integers of length n can be multiplied in time*

$$n \log n 2^{O(\log^*(n))}.$$

Remark 5.6. Beware of constants! In general,

$$2^{O(f(n))} \neq O(2^{f(n)})$$

Indeed

$$2^{cf(n)} = (2^{f(n)})^c$$

which will in general grow faster than $2^{f(n)}$.

In the recent years, researchers have focused on improving that constant, the current best result is the following:

Theorem 5.7 (Harvey, van der Hoeven, 2018)**.** *Integers of length n can be multiplied in time*

$$O(n \log n 2^{2 \log^*(n)}).$$

Remark 5.8. Forgetting the constants, we have

$$\log \log n \geq 2^{2 \log^* n} \iff n \geq 2^{2^{2^{12}}}.$$

Remember that n is the *number of digits* of the integers we want to multiply!

5.2 Integer multiplication in practice

Those algorithms are only of theoretical interest. The following algorithm follows a more pragmatic approach, which is usually superior.

Write $F = (f_{n-1} \dots f_1 f_0)_w$ and $G = (g_{n-1} \dots g_1 g_0)_w$ in base w with w as large as possible. In practice, one can for example choose w to be the largest possible processor word.

Define

$$\bar{f} = f_0 + f_1 X + \cdots + f_{n-1} X^{n-1}$$
$$\bar{g} = g_0 + g_1 X + \cdots + g_{n-1} X^{n-1}$$

so that $\bar{f}(w) = f$ and $\bar{g}(w) = g$. Let

$$\bar{h} = \bar{f}\bar{g} = \bar{h}_0 + \bar{h}_1 X + \dots$$

Note that $0 \leq \bar{h}_i \leq nw^2$ for all i.

Assume that $n < w/8$ and fix three primes p_1, p_2, p_3 between $w/2$ and w, for which the field $\mathbb{F}_{p_i}$ contains a 2^t'th root of unity for some large t. Then compute $\bar{f}\bar{g}$ in $\mathbb{F}_{p_i}[X]$ for $i = 1, 2, 3$, and reconstruct the coefficients of $\bar{h}$ with the Chinese remainder theorem. Finally compute $h = \bar{h}(w)$.

Example 5.9. On a 64-bits processor, let's choose $w = 2^{64}$. Then

$$p_1 = 95 \cdot 2^{57} - 1$$
$$p_2 = 108 \cdot 2^{57} - 1$$
$$p_3 = 123 \cdot 2^{57} - 1$$

are suitable primes, with $t = 57$ and $55, 65$ and 493 the respective 57'th roots of unity.

This is the method of choice for multiplying integers up to $\simeq 500$ millions of bits on a 64-bits architecture.

6 Fast multiplication in $R[X, Y]$

Given $f, g \in R[X, Y]$, we want to compute $h = fg$.

6.1 Isolating a variable

We can use Algo. 6 in $R[X][Y]$. But the complexity is bounded in number of operations in $R[X]$, not in R. In order to get a complete bound, we need an estimate for the degree growth in X.

If we define

$$d_X := \deg_X(h) = \deg_X(f) + \deg_X(g)$$
$$d_Y := \deg_Y(h) = \deg_Y(f) + \deg_Y(g)$$

it suffices to compute the product in

$$R[X]/\langle X^{d_X+1} - 1\rangle[Y]/\langle Y^{d_Y+1} - 1\rangle.$$

Let

$$D := R[X]/\langle X^{d_X+1} - 1\rangle.$$

If we use for example Algo. 6 to compute the multiplication in $D[Y]/\langle Y^{d_Y+1} - 1\rangle$, it requires $M(d_Y)$ operations in D, each of them requires at most $M(d_X)$ operations in R.

Theorem 6.1. *Polynomials $f, g \in R[X, Y]$, with $\deg_X(f), \deg_X(g) \leq n$ and $\deg_Y(f), \deg_Y(g) \leq m$, can be multiplied with $M(n)M(m)$ operations in R.*

6.2 Kronecker substitution

Algorithm 7 Multiplication using Kronecker substitution

Input: $f, g \in R[X]$ with $\deg_X(fg) < n$, $\deg_Y(fg) < m$

Output: $h = fg$

1. $\bar{f} \leftarrow f(X, X^n)$, $\bar{g} \leftarrow g(X, X^N) \in R[X]$
2. Compute $\bar{h} = \bar{f} \cdot \bar{g} \in R[X]$ with a fast algorithm
3. Write $\bar{h} = h^{(0)} + h^{(1)}X^n + h^{(2)}X^{2n} + \cdots + h^{(m-1)}X^{(m-1)n}$ with $\deg(h^{(i)}) < n$
4. Return $h = h^{(0)} + h^{(1)}Y + h^{(2)}Y^2 + \cdots + h^{(m-1)}Y^{m-1}$

Theorem 6.2. *Algo. 7 requires $M(mn)$ operations in R.*

Proof. The only multiplication computed involves polynomials in $R[X]$ with degree at most nm. □

Remark 6.3. $M(mn)$ may not be strictly less than $M(m)M(n)$.

7 Fast division

Let K be a field. The task is, given $f, g \in K[X]$, to find $q, r \in K[X]$ such that $f = qg + r$ and $\deg(r) < \deg(g)$.

7.1 Horner's rule

Horner's rule is a technique for evaluating a polynomial f with degree m at some value v with $O(m)$ multiplications, instead of the naive m^2. It avoids computing successive powers of v, and instead relies on the following rewriting of f:

$$\begin{aligned} f &= a_0 + a_1 X + \cdots + a_m X^m \\ &= a_0 + X\Big(a_1 + X\big(\cdots + X(a_m)\ldots\big)\Big). \end{aligned}$$

The resulting algorithm is actually the naive Euclidean algorithm used to compute f divided by $g = X - v$. The remainder of that division is $f(v)$.

The same algorithm can be used for a polynomial g with degree n, and it then uses $O(nm)$ operations in K.

7.2 A Karatsuba-style algorithm: Jebelean's algorithm

There is also a Karatsuba-style division algorithm. Assume that $\deg f < 2 \deg g$ and $\deg g$ is a power of 2.

Algorithm 8 Jebelean's algorithm (1993)

Input: $f, g \in K[X]$, $k \in \mathbb{N}$, with $\deg g = n = 2^i$, $\deg f < 2n + k$.

Output: q, r such that $f = gX^k q + r$ and $\deg(r) < n + k$

1. If $\deg f < \deg g + k$, then return $q = 0$, $r = f$
2. If $\deg g = 1$, then use Horner's algorithm
3. Write $g = g^{(0)} + g^{(1)}X^{n/2}$ with $\deg g^{(0)} < \frac{n}{2}$ # $\deg g^{(1)} = \frac{n}{2}$

Compute $q^{(1)}, r^{(1)}$ such that $f = q^{(1)}X^{n+k}g^{(1)} + r^{(1)}$ with $\deg r^{(1)} < \frac{3n}{2} + k$

4. Find $q^{(1)}, r^{(1)}$ by calling Algo. 8 with $f, g = g^{(1)}$ and $k = n + k$

Compute the true remainder $u = f - q^{(1)}X^{n+k}g$

5. Compute $u = r^{(1)} - X^{n/2+k}g^{(0)}q^{(1)}$ using Algo. 1

Compute $q^{(0)}, r^{(0)}$ such that $u = q^{(0)}X^{n/2+k}g^{(1)} + r^{(0)}$ with $\deg r^{(0)} < \frac{n}{2} + k$

6. Find $q^{(0)}, r^{(0)}$ by calling Algo. 8 with $f = u, g = g^{(1)}$ and $k = \frac{n}{2} + k$

Compute the true remainder $r = u - q^{(0)}X^k g$

7. Compute $r = r^{(0)} - g^{(0)}q^{(0)}X^k$ using Algo. 1
8. Return $q = q^{(0)} + q^{(1)}X^{n/2}$ and r

Theorem 7.1. *Algo. 8 is correct.*

Proof. We prove it by induction on n, then on k. The case $n = 1$ is clear, as is the case $\deg f < n + k$. Now assume that the algorithm is correct for all input of size $< n$ or third argument $> k$. Consider $f, g \in R[X]$, $k \in \mathbb{N}$ with $\deg g = n$ and $\deg f < 2n + k$. In particular, $\deg g^{(1)} = \frac{n}{2}$.

So the call to Algo. 8 with $f = f$, $g = g^{(1)}$ and $k = n + k$ is correct, and the results are $q^{(1)}, r^{(1)}$ such that $f = g^{(1)}X^{n+k}q^{(1)} + r^{(1)}$, $\deg(r^{(1)}) < n + k$, and

$$\deg(q^{(1)}) = \deg(f) - \deg(X^{n+k}g^{(1)}) < \frac{n}{2}.$$

The polynomial u satisfies

$$\begin{aligned} u &= r^{(1)} - X^{n/2+k}g^{(0)}q^{(1)} \\ &= f - X^{n+k}g^{(1)}q^{(1)} - X^{n/2+k}g^{(0)}q^{(1)} \\ &= f - X^{n/2+k}q^{(1)}g, \end{aligned} \tag{7.1}$$

and it has degree

$$\deg(u) < \max\left(n + k, \frac{n}{2} + k + \frac{n}{2} + \frac{n}{2}\right) < \frac{3n}{2} + k.$$

The call to Algo. 8 with $f = u$, $g = g^{(1)}$ and $k = \frac{n}{2} + k$ is correct, and $\deg r^{(0)} < n + k$ and

$\deg(q^{(0)}) = \frac{n}{2} + k$. So we get

$$\begin{aligned} u &= X^{n/2+k} g^{(1)} q^{(0)} + r^{(0)} \\ &= X^{n/2+k} g^{(1)} q^{(0)} + X^k g^{(0)} q^{(0)} + r \quad \text{(by definition of } r\text{)} \\ &= g X^k q^{(0)} + r. \end{aligned}$$

The polynomial r has degree

$$\deg(r) < \max\left(n + k, \frac{n}{2} + \frac{n}{2} + k\right) < n + k,$$

and putting it all together using Eq. (7.1), we find

$$f = X^{n/2+k} q^{(1)} g + X^k q^{(0)} g + r = X^k \left(X^{n/2} q^{(1)} + q^{(0)}\right) g + r.$$

□

Theorem 7.2 (Jebelean, 1993)**.** *Algo. 8 requires at most $2M_K(n)$ multiplications in K where $M_K(n)$ is the number of multiplications performed by Algo. 1 (Karatsuba).*

Remark 7.3. There is no O in that result.

Proof. Recall the recurrence relation

$$M_K(2n) = 3M_K(n).$$

If we proceed by induction, the number of multiplications $T(n)$ performed by Algo. 8 satisfies the recurrence relation

$$\begin{aligned} T(n) &= 2T\left(\frac{n}{2}\right) + 2M_K\left(\frac{n}{2}\right) \\ &= 2 \cdot 2M_K\left(\frac{n}{2}\right) + 2M_K\left(\frac{n}{2}\right) \\ &= 6M_K\left(\frac{n}{2}\right) \\ &= 2M_K(n). \end{aligned}$$

□

Remark 7.4. The integer version of Algo. 8 is the best-performing division algorithm for integers of a certain size.

Remark 7.5. The total number of operations (including additions) is $O(M_K(n)\log(n))$.

7.3 Division with the cost of multiplication

We now want to perform division in time $O(M(n))$.

Definition 7.6. Let $f \in K[X]$ and $k \in \mathbb{N}$, the *k'th reversal* of f is

$$\mathrm{rev}_k(f) := X^k f\left(\frac{1}{X}\right).$$

Example 7.7. If $f = f_0 + f_1 X + \cdots + f_n X^n$, then $\mathrm{rev}_n(f) = f_n + f_{n-1}X + \cdots + f_0 X^n$.

Remark 7.8. In general, $\mathrm{rev}_k(f) \in K[X]$ if $k \geq n$.

Let $f, g \in K[X]$ with $\deg(f) = m$, $\deg g = n < m$, and q, r be the quotient and remainder respectively of the division of f by g. Performing the change of variable $X \mapsto 1/X$ and multiplying by X^m the equality $f = qg + r$ gives

$$\begin{aligned} X^m f\left(\frac{1}{X}\right) &= X^n g\left(\frac{1}{X}\right) X^{m-n} q\left(\frac{1}{X}\right) + X^{m-n+1} X^{n-1} r\left(\frac{1}{X}\right) \\ \mathrm{rev}_m f &= \mathrm{rev}_n g \cdot r_{m-n} q + X^{m-n+1} \mathrm{rev}_{n-1} r \end{aligned}$$

so

$$\mathrm{rev}_m f = \mathrm{rev}_n g \cdot \mathrm{rev}_{m-n} q \bmod X^{m-n+1}.$$

Furthermore, since $\deg g = n$, we have $(\mathrm{rev}_n g)_0 \neq 0$, so $\mathrm{rev}_n g$ is invertible modulo X^{m-n+1}. Therefore

$$\mathrm{rev}_{m-n} q = \mathrm{rev}_m f \cdot (\mathrm{rev}_n g)^{-1} \bmod X^{m-n+1}.$$

So what we need is a fast algorithm for inversion modulo X^l: an algorithm which, given $u \in K[X]$ with $u_0 \neq 0$ and $l \in \mathbb{N}$, computes $v \in K[X]$ such that $uv = 1 \bmod X^l$.

Regard $u \in K[X] \subset K[[X]]$ as a formal power series, and consider the map

$$\begin{array}{rccc} \varphi: & K[[X]]^* & \to & K[[X]] \\ & s & \mapsto & u - \frac{1}{s}. \end{array}$$

Let v be a root of φ, we can write

$$v = w + X^l r$$

with $w \in K[X]_{l-1}$, and w, seen as a power series, is invertible. Then

$$\begin{aligned} 0 = \varphi(v) &= u - \frac{1}{w + X^l r} = u - \frac{1}{w}\frac{1}{1 + X^l r/w} \\ &= u - \frac{1}{w} + X^l \frac{r}{v_0^2} - O(X^{l+1}) \end{aligned}$$

so

$$uw = 1 + X^l \frac{r}{w} + O(X^{l+1}) = 1 \bmod X^l.$$

So we have to find an approximation of order l of a root v of φ. For this purpose, we use Newton iteration: we compute successive approximations of the root, starting with

$$v^{(0)} = \frac{1}{u_0}$$

and iterating with

$$\begin{aligned} v^{(k+1)} &= v^{(k)} - \frac{\varphi(v^{(k)})}{\varphi'(v^{(k)})} = v^{(k)} - \frac{u - \frac{1}{v^{(k)}}}{\left(\frac{1}{v^{(k)}}\right)^2} \\ &= 2v^{(k)} - u \cdot (v^{(k)})^2. \end{aligned}$$

This would give us an algorithm, if only we knew when to stop!

Theorem 7.9. *For all $k \geq 0$, $u \cdot v^{(k)} = 1 \bmod X^{2^k}$.*

Proof. Proof by induction: for $k = 0$, we have

$$u \cdot v^{(0)} = u_0 \cdot \frac{1}{u_0} + O(X) = 1 \bmod X.$$

If it is true for $k \geq 0$, then

$$\begin{aligned} 1 - uv^{(k+1)} &= 1 - u(2v^{(k)} - u \cdot (v^{(k)})^2) = 1 - 2uv^{(k)} + (uv^{(k)})^2 = \left(1 - u \cdot v^{(k)}\right)^2 \\ &= O(X^{2^{k+1}}) = 0 \bmod X^{2^{k+1}}. \end{aligned}$$

□

Remark 7.10. This theorem is a particular case of a more general fact: with a starting point sufficiently close to a root, Newton iteration converges quadratically fast.

Algorithm 9 Inversion using Newton iteration

Input: $u \in K[X]$ with $u_0 \neq 0$, $n \in \mathbb{N}$

Output: $v \in K[X]$ with $u \cdot v = 1 \bmod X^n$

1. $v \leftarrow \frac{1}{u_0}$
2. For i from 1 to $\lceil \log(n) \rceil$, do
3. $\quad v \leftarrow 2v - uv^2 \text{ rem } X^l$
4. Return v

Theorem 7.11. *Algo. 9 requires $O(M(n))$ operations in K.*

Proof. Let $T(n)$ be the number of operations required. Then

$$\begin{aligned} T(n) &\leq \sum_{i=1}^{\lceil \log(n) \rceil} 2M(2^i) + c2^i \\ &\leq c2^{\lceil \log(n) \rceil + 1} + 2 \sum_{i=1}^{\lceil \log(n) \rceil} \underbrace{M(2^i)}_{\leq \frac{M(n)}{n/2^i}} \\ &\leq 4cn + 2\frac{M(n)}{n} \underbrace{\sum_{i=1}^{\lceil \log(n) \rceil} 2^i}_{\leq 4n} \\ &\leq 4cn + 8M(n) = O(M(n)). \end{aligned}$$

□

With this taken care of, we can now write down all the steps required to perform a fast division.

Algorithm 10 Fast division

Input: $f, g \in K[X]$, $k \in \mathbb{N}$, with $\deg f = m$, $\deg g < n$, $g \neq 0$

Output: q, r such that $f = qg + r$ and $\deg(r) < \deg(g)$

1. If $m < n$ then return $q = 0$, $r = f$
2. Compute $h = \mathrm{rev}_n(g)^{-1} \bmod X^{m-n+1}$ with Algo. 9
3. $\bar{q} \leftarrow \mathrm{rev}_m(f)h$
4. Return $q = \mathrm{rev}_{m-n}(\bar{q})$ and $r = f - gq$

Theorem 7.12. *Algo. 10 requires $O(M(m))$ operations in K.*

Remark 7.13. This result is the current world record for polynomial division.

Remark 7.14. In particular, if $f, g, q \in K[X]$ with $\deg(f), \deg(g), \deg(q) \leq n$, then we can compute (and reduce) $f, g \in K[X]/\langle q \rangle$ with $O(M(n))$ operations in K.

If $\gcd(f, g) = 1$, then we will see that $f^{-1} \bmod q$ can be computed using $O(M(n)\log(n))$ operation in K (using the fast GCD algorithm).

7.4 Exercises

Exercise 7.1. Assume that the field K is algebraically closed. Find a bound for the complexity of Algo. 8 if we use FFT instead of Karatsuba's algorithm for the multiplication. Is it better?

Exercise 7.2. How would you adapt Algo. 8 to work with any polynomial g (even if its degree is not a power of 2)?

Exercise 7.3.

1. Write an analogue of Algo. 8 for polynomials such that $\deg(f) \leq 3\deg(g)$. What is its complexity?
2. Generalize to any $f, g \in K[X]$. What is the resulting complexity?

8 Computing with homomorphic images

This chapter does not introduce fast algorithms, but serves as a motivation for algorithms in later chapters, namely multipoint evaluation, interpolation and half-GCD.

8.1 The problem of coefficient explosion

Let $A \in \mathbb{Q}^{n \times n}$, and assume that you want to solve the system

$$A \cdot \begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix} = 0.$$

This can be done in $O(n^\omega)$ operations in $\mathbb{Q}$ using fast linear algebra. However, in practice, for large n, the computation will take very long.

The reason is *expression swell*: the algorithms multiplies and adds rational numbers which become larger and larger. Additions and multiplications for rationals are defined as

$$\frac{a}{b} + \frac{c}{d} = \frac{ad + bc}{bd}$$
$$\frac{a}{b} \cdot \frac{c}{d} = \frac{ac}{bd}$$

so after each operation, the coefficients size is roughly doubled. Reducing the fractions helps, but not by a significant factor.

The typical situation is that the input is small (because it comes from actual data) and the output is small (because frequently in applications, meaningful data tends to not be overcomplicated), but intermediate expressions will be meaningless and huge.

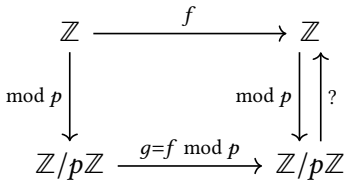
The idea to mitigate this problem is to reduce the problem to domains where objects have a fixed size, so that the actual complexity does not deviate from the predicted number of operations.

8.2 Computations in $\mathbb{Z}$ using modular arithmetic

When doing operations in $\mathbb{Z}$, multiplication doubles the size of the output.

For simplicity, consider a ring homomorphism $f : \mathbb{Z} \to \mathbb{Z}$, and assume that we want to compute $f(x)$, avoiding expression swell inside f.

We want to do the computations in $\mathbb{Z}/p\mathbb{Z}$, for $p \in \mathbb{Z} \setminus \{0\}$.



The diagram commutes, which means that given $x \in \mathbb{Z}$, $g(x \bmod p) = f(x) \bmod p$. But what we want is $f(x)$, not its equivalence class modulo p.

So we want to choose p such that $f(x)$ can be recovered from $f(x) \bmod p$.

There are two interesting scenarios:

1. We know an *a priori* bound $M(x)$ with $|f(x)| \leq M(x)$. Then taking $p > 2M(x)$ will ensure that

$$(f(x) \bmod p) \cap \{-M(x), -M(x)+1, \ldots, M(x)\} = \{f(x)\}.$$

2. We can efficiently check, given $y \in \mathbb{Z}$, whether $f(x) = y$. Then repeat the computation with increasing p until y, defined as

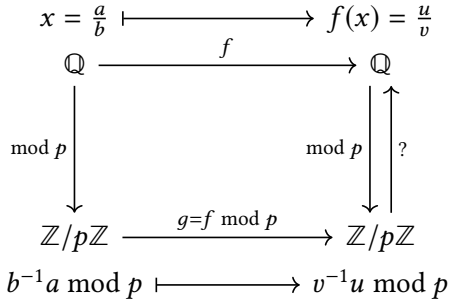
$$\{y\} = (f(x) \bmod p) \cap \left\{-\frac{p}{2}, -\frac{p}{2}+1, \ldots, \frac{p}{2}\right\},$$

is the solution.

8.3 Computations in $\mathbb{Q}$ using rational reconstruction

We now turn back to the problem of $\mathbb{Q}$, where both additions and multiplications double the size of the output. We can do the same thing for a morphism $f : \mathbb{Q} \to \mathbb{Q}$, using modular inverses.

Assume that b and v are coprime to p, we have a commutative diagram:



As in the case of $\mathbb{Z}$, we want to choose p such that $f(a/b)$ can be recovered from $g(b^{-1}a \bmod p)$.

For sufficiency, there are again two scenarios:

1. We know a bound $M(x)$ such that $u^2 + v^2 \leq M(x)^2$. Then taking $p > M(x)^2$ will ensure that

$$\{(w,z) \in \mathbb{Z} \times \mathbb{N} : z^{-1}w = v^{-1}u \bmod p\} \cap \{(w,z) : w^2 + z^2 \leq M(x)\} = \{(u,v)\}.$$

2. We can efficiently check for a given $y \in \mathbb{Q}$ whether $f(x) = y$. Then as in the case of $\mathbb{Z}$, we try increasing values of p until the result is found.

In both cases, in order to determine the intersection point, we need a way, given $y \in \mathbb{Z}/p\mathbb{Z}$ to compute $(u,v) \in \mathbb{Z} \times \mathbb{N}$ such that $v^{-1}u = y \bmod p$ and $u^2 + v^2$ is minimal.

Proposition 8.1. *Consider a run of the Extended Euclid's Algorithm on p and y. Let $(u,v) \in \mathbb{Z}\times\mathbb{N}$ such that $v^{-1}u = y \bmod p$ and $u^2 + v^2$ is minimal. Let g_i, s_i, t_i be values computed at each step of the algorithm, for $i = 1, \ldots, l$:*

$$\begin{aligned}
p = g_1 &= s_1 p + t_1 y = 1 \cdot p + 0 \cdot y \\
y = g_2 &= s_2 p + t_2 y = 0 \cdot p + 1 \cdot y \\
g_3 &= s_3 p + t_3 y \\
&\vdots \\
1 = g_l &= s_l p + t_l y
\end{aligned}$$

Then

$$\{(b,a) \text{ such that } b^{-1}a = y \bmod p\} \supset \{(g_i, t_i) : i \in \{2, \ldots, l\}\} \ni (v,u).$$

Example 8.2. Consider the case $p = 65521$, $y = 29771$, and compute an inverse t of 29771 modulo 65521 using the Extended Euclid's Algorithm, or in other words, a pair $s, t \in \mathbb{Z}$ such that

$$1 = 65521s + 29771t.$$

Here are the intermediate values:

g	s	t
65521	0	1
29771	1	0
5979	-2	1
5855	9	-4
124	-11	5
27	526	-239
16	-2115	961
11	2641	-1200
5	-4756	2161
1	12153	-5522

Then, modulo 65521,

$$29771 = \frac{29771}{1} = -\frac{5979}{2} = \frac{5855}{9} = -\frac{124}{11} = \cdots = \frac{1}{12153}$$

and the minimal pair of numerator and denominator for 29771 is given halfway through the algorithm: it is $(-124, 11)$.

Remark 8.3. It means that the Extended Euclid's Algorithm is useful beyond returning the Bézout coefficients. If we are looking for a rational fraction equal to $x \bmod p$, given a bound on the size of the coefficients, we can find it by examining all lines in the algorithm. And, for the particular case where we want both coefficients to have roughly the same size, the relevant line will be roughly halfway through the algorithm, and can be found using half-GCD algorithms.

8.4 Computation with large moduli using Chinese Remaindering

We saw that computations in $\mathbb{Z}$ and $\mathbb{Q}$ can be done in $\mathbb{Z}/p\mathbb{Z}$, for $p \in \mathbb{N}$ large enough compared to a bound $M(x)$ on the wanted result. But if $M(x)$ is large, p will need to be large, again making the computations expensive.

It is possible to mitigate this problem using the Chinese Remainder Theorem:

$$(n \bmod p) \cap (n \bmod q) = n \bmod \operatorname{lcm}(p, q).$$

So by running the computations modulo p and q, we can reconstruct the result modulo $\operatorname{lcm}(p, q)$.

We still need to be able to find the canonical (small) representative of n modulo $\operatorname{lcm}(p, q)$, given the representatives modulo p and q.

For simplicity, assume that p and q are coprime, so that $\operatorname{lcm}(p, q) = pq$. We are given $n_p, n_q \in \mathbb{Z}$, and we want to find $n \in \mathbb{Z}$ such that

$$\begin{cases} n \bmod p = n_p \bmod p \\ n \bmod q = n_q \bmod q. \end{cases}$$

Since $\gcd(p, q) = 1$, there exists $s, t \in \mathbb{Z}$ such that

$$sq + tq = 1.$$

Let

$$n = n_p + (n_q - n_p)sp \in \mathbb{Z}$$

it is congruent to n_p modulo p and to $n_p + (n_q - n_p) = n_q$ modulo q. So we can just take the canonical representative of n modulo pq.

This can be generalized to more moduli.

Algorithm 11 Chinese Remainder reconstruction

Input:

- $u_1, \ldots, u_n \in \mathbb{Z}$
- $p_1, \ldots, p_n \in \mathbb{Z}$, pairwise coprime

Output: $u \in \mathbb{Z}$ such that $u \bmod p_i = u_i \bmod p_i$ for $i = 1, \ldots, n$

1. $u \leftarrow u_1$
2. $m \leftarrow 1$
3. For k from 2 to n do
4. $\quad m \leftarrow m \cdot p_{k-1}$
5. $\quad s \leftarrow m^{-1} \bmod p_k$
6. $\quad u \leftarrow ((u_k - u)s \text{ rem } p_k)\, m$
7. Return u

Remark 8.4. It means that in the second scenarios, both for $\mathbb{Z}$ and $\mathbb{Q}$, when computing modulo p for increasing values of p, we do not have to throw away results for values of p which were too small. We can use them to reconstruct larger moduli.

Remark 8.5. For example, if we take $p_1, \ldots, p_{20}$ to be the first 20 primes, we can reconstruct results modulo

$$p_1 \cdots p_{20} = 2 \cdot 3 \cdots 71 \simeq 5.6 \cdot 10^{26} \simeq 1.8 \cdot 2^{88}$$

8.5 Computations in $K[X]$ and $K(X)$

In $K[X]$ and $K(X)$, we face the same problem as in $\mathbb{Z}$ and $\mathbb{Q}$ respectively. We can use the same techniques as in the case of integers to reduce to problems over $K[X]/\langle P \rangle$ for some small irreducible polynomial P.

A good choice for P is $X - a$, with $a \in K$, and then $K[X]/\langle P \rangle = K$. In that case, the operations of reducing modulo $X - a_i$, $a_i \in K$, $i \in \{0, \ldots, n\}$, running the computations in K and reconstructing the resulting polynomial constitute the evaluation/interpolation method seen before.

Remark 8.6. Algo. 11 for polynomials is Newton's interpolation.

9 Fast evaluation and interpolation

Fast multiplication and fast division algorithms are useful because those operations are heavily used in many higher-level algorithms. However, it is frequently not enough, in order to obtain a speed-up, to replace the operations with their fast counterparts.

Example 9.1. Given $n \in \mathbb{N}$ (with n smaller than a machine word), how to compute $n!$?

The usual algorithm uses the formula

$$n! = n \cdot (n-1)!.$$

This algorithm is recursively called linearly-many times, and at each step does one multiplication with a small integer (with a linear cost). Its complexity satisfies

$$T(n) = T(n-1) + O(n),$$

so

$$T(n) = O(n^2).$$

On the other hand, an algorithm using the following formula

$$n! = \left(\frac{n}{2}\right)! \cdot \left(\prod_{k=\frac{n}{2}+1}^{n} k \right)$$

is recursively called log-many times, and at each step adds one large multiplication. Its complexity satisfies

$$T(n) = 2T\left(\frac{n}{2}\right) + M\left(\frac{n}{2}\right)$$

so

$$T(n) = O(M(n)\log(n)).$$

If $M(n) = O(n^2)$, it's worse. If $M(n) = \tilde{O}(n)$, it's better.

The lesson is that in order to take advantage of fast multiplication, algorithms need to be adjusted. It is usually not sufficient to plug fast multiplication into a standard algorithm.

We want to do two things in this chapter:

Evaluation Given $f \in K[X]$ with $\deg(f) < n$ and $a = (a_0, \ldots, a_{n-1}) \in K^n$, compute the multipoint evaluation $f(a_0), \ldots, f(a_{n-1}) \in K$

Interpolation Given $a = (a_0, \dots, a_{n-1}) \in K^n$ with $a_i \neq a_j$ for $i \neq j$, and $b = (b_0, \dots, b_{n-1}) \in K^n$, compute $f \in K[X]$ with $\deg(f) < n$ such that $f(a_i) = b_i$ for all i.

Remark 9.2. If ω is a n'th root of unity in K and $a_i = \omega^i$ $(i = 0, \dots, n-1)$, then we can accomplish both tasks with $O(n \log(n))$ operations in K. But this doesn't work with arbitrary a_i.

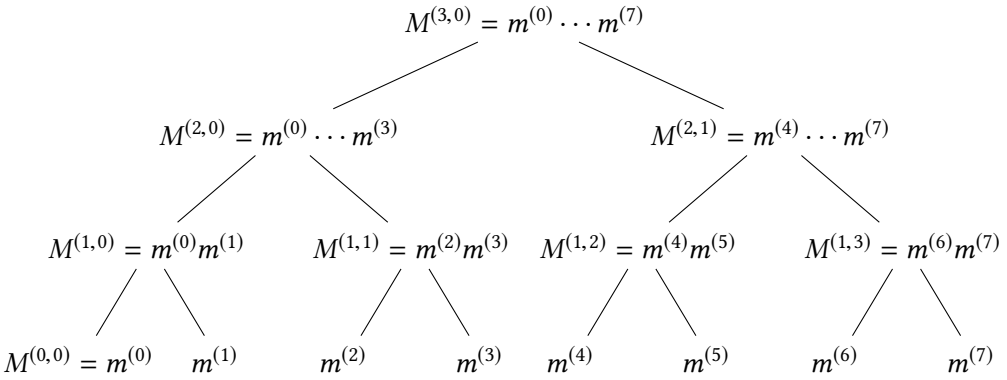
Remark 9.3. The standard algorithms (Horner rule called n times, Newton interpolation) require $O(n^2)$ operations, with no improvements with fast multiplications.

The goal is to find algorithms for both operations using $O(M(n)\log(n))$ operations in K. The main idea, similar to the example of the factorial, is to split the problem into parts of equal size, instead of proceeding point by point.

9.1 Evaluation

Let $m^{(i)} = X - a_i$ and define, for $0 \leq i < k = \log(n)$, $O \leq j < 2^{k-i}$,

$$M^{(i,j)} = m^{(j2^i)} m^{(j2^i+1)} \cdots m^{(j2^i+2^i-1)} = \prod_{l=0}^{2^i-1} m^{(j2^i+l)}.$$



Algorithm 12 Splitting subroutine

Input: $a = (a_0, \dots, a_{n-1}) \in K^n$, $n = 2^k$

Output: $M^{(i,j)}$ as defined above

1. $M^{(0,j)} \leftarrow X - a_j$ $(j \in \{0, \dots, n-1\})$
2. For i from 1 to k do
3. $\quad M^{(i,j)} \leftarrow M^{(i-1,2j)} \cdot M^{(i-1,2j+1)}$ $(j \in \{0, \dots, 2^{k-i} - 1\})$

Theorem 9.4. *Algorithm 12 requires $O(M(n)\log(n))$ operations in K.*

Proof. Let $T(n)$ be the number of operations required in a run of Algorithm 12. Note that

$\deg(M^{(i,j)}) = 2^i$. Then $T(n)$ satisfies

$$\begin{aligned} T(n) &= \sum_{i=1}^{k} \sum_{j=0}^{2^{k-i}-1} M(2^{i-1}) \\ &\leq \sum_{i=1}^{k} M\left(\sum_{j=0}^{2^{k-i}-1} 2^{i-1} \right) \\ &\leq kM(2^{k-1}) \leq kM\left(\frac{n}{2}\right) = O(M(n)\log(n)). \end{aligned}$$

□

Note that $X - a_j$ divides $M^{(k-1,0)}$ for $j \in \{0, \ldots, \frac{n}{2} - 1\}$. So, if we write

$$f = qM^{(k-1,0)} + r$$

with $r = f \text{ rem } M^{(k-1,0)}$, then

$$f(a_j) = r(a_j)$$

for $j \in \{0, \ldots, \frac{n}{2} - 1\}$. Likewise, for $j \in \{\frac{n}{2}, \ldots, n-1\}$,

$$f(a_j) = \left(f \text{ rem } M^{(k-1,1)}\right)(a_j).$$

This suggests the following algorithm for evaluation.

Algorithm 13 Multipoint evaluation

Input: $f \in K[X]$, $\deg(f) < n = 2^k$, $a = (a_0, \ldots, a_{n-1}) \in K^n$

Output: $(f(a_0), \ldots, f(a_{n-1})) \in K^n$

1. If $n = 1$ then return $f(a_0)$
2. Compute $(M^{(i,j)})$ with Algorithm 12 and cache the result
3. $r^{(0)} \leftarrow f \text{ rem } M^{(k-1,0)}$, $r^{(1)} \leftarrow f \text{ rem } M^{(k-1,1)}$
4. Compute recursively $\left(r^0(a_0), \ldots, r^{(0)}(a_{\frac{n}{2}-1})\right)$
5. Compute recursively $\left(r^1(a_{\frac{n}{2}}), \ldots, r^{(1)}(a_{n-1})\right)$
6. Return $\left(r^0(a_0), \ldots, r^{(0)}(a_{\frac{n}{2}-1}), r^1(a_{\frac{n}{2}}), \ldots, r^{(1)}(a_{n-1})\right)$

Theorem 9.5. *Algorithm 13 requires $O(M(n)\log(n))$ operations in K.*

Remark 9.6. This is the best known complexity for multipoint evaluation.

Proof. We only need to compute the $M^{(i,j)}$ once, for a fixed cost of $O(M(n)\log(n))$. Let $T(n)$ be the number of operations required for the rest of the computations, we will prove by induction

that $T(n) \leq cM(n)\log(n)$. The complexity $T(n)$ satisfies the recurrence

$$\begin{aligned} T(n) &= 2T\left(\frac{n}{2}\right) + \overbrace{O(M(n))}^{r^{(0)} \text{ and } r^{(1)}} \\ &\leq 2cM\left(\frac{n}{2}\right)\log\left(\frac{n}{2}\right) + c_2 M(n) \\ &\leq cM(n)\log\left(\frac{n}{2}\right) + c_2 M(n) \\ &\leq cM(n)\log(n) + (c_2 - 2c)M(n) \\ &\leq cM(n)\log(n) \quad \text{by choosing } c \text{ large enough for the second term to be negative.} \end{aligned}$$

□

9.2 Interpolation

Recall that given $a = (a_0, \ldots, a_{n-1}) \in K^n$ with $a_i \neq a_j$ for $i \neq j$, and $b = (b_0, \ldots, b_{n-1}) \in K^n$, we want to compute $f \in K[X]$ with $\deg(f) < n$ such that $f(a_i) = b_i$ for all i.

Recall (Lagrange interpolation).

$$f = \sum_{j=0}^{n-1} b_j L_j$$

where

$$L_j = \prod_{i \neq j} \frac{X - a_i}{a_j - a_i} = \begin{cases} 1 \text{ at } X = a_j \\ 0 \text{ at } X = a_i,\ i \neq j. \end{cases}$$

It can be rewritten as

$$L_j = \frac{M^{(k,0)}}{(X - a_j)S^{(j)}} \text{ where } S^{(j)} = \prod_{i \neq j} a_j - a_i.$$

Observe that

$$\frac{\mathrm{d}M^{(k,0)}}{\mathrm{d}X} = \frac{\mathrm{d}}{\mathrm{d}X}\prod_{i=0}^{n-1}(X - a_i) = \sum_{j=0}^{n-1}\prod_{i \neq j}(X - a_i) = \sum_{j=0}^{n-1}\frac{M^{(k,0)}}{X - a_j},$$

so that $S^{(j)} = \frac{\mathrm{d}}{\mathrm{d}X}M^{(k,0)}|_{X=a_j}$ can be obtained by fast multipoint evaluation applied to $\frac{\mathrm{d}}{\mathrm{d}X}M^{(k,0)}$.

Next we need a fast way to compute linear combinations $\sum_{j=0}^{n-1} c_j \frac{M^{(k,0)}}{X - a_j}$. This is the purpose of the next subroutine.

Algorithm 14 Linear combination subroutine

Input:

- $a = (a_0, \dots, a_{n-1}) \in K^n$ with $a_i \neq a_j$ $(i \neq j)$
- $n = 2^k$
- $c = (c_0, \dots, c_{n-1}) \in K^n$
- $M^{(i,j)}$ as computed by Algo. 12

Output: $\displaystyle\sum_{j=0}^{n-1} c_j \frac{M^{(k,0)}}{X - a_j}$

1. If $n = 1$ then return c_0
2. Compute $r^0 \leftarrow \displaystyle\sum_{j=0}^{n/2-1} c_j \frac{M^{(k-1,0)}}{X - a_j}$ recursively
3. Compute $r^1 \leftarrow \displaystyle\sum_{j=n/2}^{n-1} c_j \frac{M^{(k-1,1)}}{X - a_j}$ recursively
4. Return $M^{(k-1,1)} r^{(0)} + M^{(k-1,0)} r^{(1)}$

Theorem 9.7. *Algo. 14 requires $O(M(n)\log(n))$ operations in K.*

Proof. As usual by induction, with the complexity satisfying the recurrence formula

$$T(n) = 2T\left(\frac{n}{2}\right) + O\left(M\left(\frac{n}{2}\right)\right)$$

□

Algorithm 15 Fast interpolation

Input:

- $a = (a_0, \dots, a_{n-1}) \in K^n$ with $a_i \neq a_j$ $(i \neq j)$
- $n = 2^k$
- $b = (b_0, \dots, b_{n-1}) \in K^n$

Output: $f \in K[X]$ with $\deg(f) < n$ and $f(a_i) = b_i$ for $i = 0, \dots, n-1$

1. Compute $M^{(i,j)}$ using Algo. 12
2. $g \leftarrow \dfrac{\mathrm{d}}{\mathrm{d}X} M^{(k,0)}$
3. Compute $(S^{(0)}, S^{(1)}, \dots) \leftarrow (g(a_0), g(a_1), \dots)$ using Algo. 13
4. Compute $f \leftarrow \displaystyle\sum_{j=0}^{n-1} \frac{b_j}{S^{(j)}} \frac{M^{(k,0)}}{X - a_j}$ using Algo. 14
5. Return f

Theorem 9.8. *Algo. 14 requires $O(M(n)\log(n))$ operations in K.*

Remark 9.9. This is the best known complexity for polynomial interpolation.

Remark 9.10. The algorithms above carry over from $K[X]$ to $R[X]$ provided that $a_i - a_j \in R^*$ for $i \neq j$. Without this condition, the Vandermonde matrix needs not be invertible and the interpolation polynomial may not exist or be unique.

Remark 9.11. There are integer versions of Algo. 13 and 15 (fast simultaneous modular reduction / fast Chinese remaindering), also running in time $O(M(n)\log(n))$.

Remark 9.12. The algorithms presented in this chapter are not faster than Algo. 3 if classical multiplication is used.

10 Fast GCD

In this chapter, the task is, given $f, g \in K[X]$ with degree $< n$, to find $h = \gcd(f, g) \in K[X]$.

10.1 "Slow" GCD: Euclid's algorithm

Algorithm 16 Euclid's algorithm

Input: $f, g \in K[X]$

Output: $\gcd(f, g)$

1. $h \leftarrow f, \bar{h} \leftarrow g$
2. While $\bar{h} \neq 0$, do
3. $\quad \begin{pmatrix} h \\ \bar{h} \end{pmatrix} \leftarrow \begin{pmatrix} \bar{h} \\ h \text{ rem } \bar{h} \end{pmatrix}$
4. Return h

Let r_i, for $i \in \mathbb{N}$, be the value of h in the i'th iteration of the loop, so $r_0 = f, r_1 = g, r_2 = f \text{ rem } g$... Let $q_i = r_{i-1} \text{ quo } r_i$ for $i \in \mathbb{N}$. Then $r_{i+1} = r_{i-1} \text{ rem } r_i = r_{i-1} - q_i r_i$, so

$$\begin{pmatrix} r_i \\ r_{i+1} \end{pmatrix} = \begin{pmatrix} r_{i+1} \\ r_{i-1} - q_i r_i \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 1 & -q_i \end{pmatrix} \cdot \begin{pmatrix} r_{i-1} \\ r_i \end{pmatrix} = Q_i Q_{i-1} \cdots Q_2 Q_1 \cdot \begin{pmatrix} f \\ g \end{pmatrix}$$

where $Q_i = \left(\begin{smallmatrix} 0 & 1 \\ 1 & -q_i \end{smallmatrix}\right)$.

Eventually,

$$\begin{pmatrix} \gcd(f, g) \\ 0 \end{pmatrix} = Q_k Q_{k-1} \cdots Q_2 Q_1 \begin{pmatrix} f \\ g \end{pmatrix}$$

where the first row of $Q_k \cdots Q_1$ contains the Bézout coefficients s, t such that

$$\gcd(f, g) = sf + tg.$$

Remark 10.1. Euclid's algorithm can be extended to keep track of the q_i's and return the Bézout coefficients as well as the GCD.

10.2 Fast GCD

Theorem 10.2. *There exists an algorithm computing the GCD of two polynomials f and g with degree $< n$, together with their Bézout coefficients, using $O(M(n)\log(n))$ operations in K.*

Note that in order to compute $\gcd(f,g)$, it is enough to compute $R := Q_k \cdots Q_1$, and we even get s, t for free in the process. In order to get a fast algorithm for the GCD, we will try to split this product in the middle.

To see how, observe that the trailing coefficients of r_0, r_1 have no influence on the first quotient q_1: if $\deg(r_0) - \deg(r_1) = k$, the degree of q_1 is k and only depends on the first coefficient of r_1 and the first k coefficients of r_0.

Remark 10.3. Generically, and except maybe at the first step, $\deg(r_i) - \deg(r_{i+1}) = 1$.

Definition 10.4. For $f \in K[X]$ with $\deg(f) = n$ and $k \in \mathbb{Z}$, we denote

$$f \upharpoonright k := f \text{ quo } X^{n-k} = f_{n-k} + f_{n-k+1}X + \ldots + f_n X^k$$

Proposition 10.5.

- *If* $k < 0$, $f \upharpoonright k = 0$.
- *If* $k \geq n$, $f \upharpoonright k = f$.
- *If* $i \in \mathbb{N}$, $(X^i f) \upharpoonright k = f \upharpoonright k$.
- *Assume that* $\deg(f) = \deg(g)$. *Then* $f \upharpoonright k = g \upharpoonright k$ *if and only if* $\deg(f - g) < \deg(f) - k$.

Definition 10.6. Let $(f,g), (\bar{f},\bar{g}) \in K[X]^2$ with $\deg(f) \geq \deg(g)$, $\deg(\bar{f}) \geq \deg(\bar{g})$ and $k \in \mathbb{Z}$. Then (f,g) and $(\bar{f},\bar{g})$ are said to *coincide up to* k, written $(f,g) \sim_k (\bar{f},\bar{g})$, if

$$\begin{cases} f \upharpoonright k = \bar{f} \upharpoonright k \\ g \upharpoonright (k - (\deg(f) - \deg(g))) = \bar{g} \upharpoonright (k - (\deg(\bar{f}) - \deg(\bar{g}))). \end{cases}$$

Remark 10.7. The relation $\sim_k$ is an equivalence relation (Exercise 10.1).

Proposition 10.8. *If* $(f,g) \sim_k (\bar{f},\bar{g})$ *and* $k \geq \deg(f) - \deg(g)$, *then* $\deg(f) - \deg(g) = \deg(\bar{f}) - \deg(\bar{g})$.

Proof. If $k \geq \deg(f) - \deg(g)$, $g \upharpoonright (k - (\deg(f) - \deg(g)))$ has degree $k - (\deg(f) - \deg(g))$ which is non-negative, and so is non-zero. By hypothesis, it is equal to $\bar{g} \upharpoonright (k - (\deg(\bar{f}) - \deg(\bar{g})))$, and in particular their degrees are equal, hence $\deg(f) - \deg(g) = \deg(\bar{f}) - \deg(\bar{g})$. □

Theorem 10.9. *Let* $(f,g), (\bar{f},\bar{g}) \in (K[X] \setminus \{0\})^2$ *and* $k \in \mathbb{Z}$ *be such that* $(f,g) \sim_{2k} (\bar{f},\bar{g})$ *and* $k \geq \deg(f) - \deg(g) \geq 0$. *Let* $q, \bar{q}, r, \bar{r}$ *be such that*

$$f = qg + r \text{ with } \deg(r) < \deg(g)$$
$$\bar{f} = \bar{q}\bar{g} + \bar{r} \text{ with } \deg(\bar{r}) < \deg(\bar{g}).$$

Then

$$q = \bar{q} \text{ and } \begin{cases} (g,r) \sim_{2(k-\deg(q))} (\bar{g},\bar{r}) \\ \text{or } r = 0 \\ \text{or } k - \deg(q) < \deg(g) - \deg(r) \end{cases}$$

Proof. Without loss of generality we can assume that $\deg(f) = \deg(\bar{f}) > 2k$: otherwise, one can multiply $(f, \bar{f})$ and $(g, \bar{g})$ by suitable powers of X and all hypotheses are still satisfied. Then, by Prop. 10.8, $\deg(g) = \deg(\bar{g})$ and $k \geq \deg(q) = \deg(f) - \deg(g) = \deg(\bar{f}) - \deg(\bar{g}) = \deg(\bar{q})$. Furthermore,

$$\begin{aligned}
\deg(f - \bar{f}) &< \deg(f) - 2k \leq \deg(g) - k \\
\deg(g - \bar{g}) &< \deg(g) - (2k - (\deg(f) - \deg(g))) = \deg(f) - k \\
&\leq \deg(g) - k \leq \deg(g) - \deg(q) \\
\deg(r - \bar{r}) &\leq \max(\deg(r), \deg(\bar{r})) < \deg(g).
\end{aligned}$$

Since

$$f - \bar{f} = q(g - \bar{g}) + (q - \bar{q})\bar{g} + (r - \bar{r}) \tag{10.1}$$

$(q - \bar{q})\bar{g}$ is a sum of terms with degree $< \deg(g)$, hence it also has degree $< \deg(g)$, hence $q = \bar{q}$.

Assume now that $r \neq 0$ and $k - \deg(q) \geq \deg(g) - \deg(r)$. We have to show that

$$g \upharpoonright 2(k - \deg(q)) = \bar{g} \upharpoonright 2(k - \deg(q)) \tag{10.2}$$

$$r \upharpoonright 2(k - \deg(q)) - (\deg(g) - \deg(r)) = \bar{r} \upharpoonright 2(k - \deg(q)) - (\deg(\bar{g}) - \deg(\bar{r})) \tag{10.3}$$

Since (f, g) and $(\bar{f}, \bar{g})$ coincide up to $2k$,

$$g \upharpoonright (2k - \deg(q)) = \bar{g} \upharpoonright (2k - \deg(q))$$

and Eq. (10.2) follows from the fact that $\deg(q) > 0$.

For Eq. (10.3), by Eq. (10.1) we have

$$\begin{aligned}
\deg(r - \bar{r}) &\leq \max(\deg(f - \bar{f}), \deg(q) + \deg(g - \bar{g})) < \deg(q) + \deg(f) - 2k \\
&= \deg(g) - 2(k - \deg(q)) = \deg(r) - 2(k - \deg(q)) - (\deg(g) - \deg(r)).
\end{aligned}$$

Furthermore by assumption

$$\deg(r) \geq \deg(q) + \deg(g) - k \geq \deg(q) + \deg(f) - 2k > \deg(r - \bar{r})$$

so $\deg(r) = \deg(\bar{r})$ and Eq. (10.3) follows from the above bound on $\deg(r - \bar{r})$. □

Theorem 10.9 gives a sufficient condition for two Euclidean quotients to be equal. We now do the same for a sequence of reductions as they happen in the Euclidean algorithm.

Let r_0, r_1 be two polynomials in $K[X]$ such that $\deg(r_0) > \deg(r_1)$, and consider as before a sequence of reductions of length l:

$$\begin{aligned}
r_{i-1} &= q_i r_i + r_{i+1} \text{ for } i = 1, \ldots, l-1 \\
r_{l-1} &= q_l r_l.
\end{aligned}$$

Let $m_i = \deg(q_i)$, $n_i = \deg(r_i)$. For $k \in \mathbb{N}$, define $\eta(k) \in \mathbb{N}$ by

$$\eta(k) = \max\left\{ j \in \{0, \dots, l\} : \sum_{i=1}^{j} m_i \leq k \right\}.$$

Note that if $0 \leq i \leq l$, then $n_i = n_0 - m_1 - \cdots - m_i$, and so

$$n_0 - n_{\eta(k)} = \sum_{i=1}^{\eta(k)} m_i \leq k < \sum_{i=1}^{\eta(k)+1} m_i = n_0 - n_{\eta(k)+1}.$$

Let $\bar{r}_0, \bar{r}_1$ be two polynomials in $K[X]$ such that $\deg(\bar{r}_0) > \deg(\bar{r}_1)$, we define analogously $\bar{l}$, $\bar{m}_i$, $\bar{n}_i$ and $\bar{\eta}(k)$.

Theorem 10.10. *Let $k \in \mathbb{N}$, $h = \eta(k)$ and $\bar{h} = e\bar{t}a(k)$. If (r_0, r_1) and $(\bar{r}_0, \bar{r}_1)$ coincide up to $2k$, then $h = \bar{h}$ and $q_i = \bar{q}_i$ for $1 \leq i \leq h$.*

Proof. We show by induction on j that the following holds for $0 < j \leq h$:

$$\begin{aligned}
&j \leq \bar{h} \\
&q_i = \bar{q}_i \text{ for } 1 \leq i \leq j \\
&\begin{cases} j = h \\ \text{or } (r_j, r_{j+1}) \text{ and } (\bar{r}_j, \bar{r}_{j+1}) \text{ coincide up to } 2(k - n_0 + n_j). \end{cases}
\end{aligned}$$

Then the claim follows by symmetry.

There is nothing to prove for $j = 0$. Assume that the induction hypothesis holds for $0 \leq j - 1 < h$. Then $r_{j-1} \neq 0$ and $k \geq n_0 - n_j \geq n_{j-1} - n_j = \bar{n}_{j-1} - \bar{n}_j$ and so $\bar{r}_j \neq 0$. Theorem 10.9 applied with $k \leftarrow k - n_0 + n_{j-1}$ implies that $q_j = \bar{q}_j$, and either (r_j, r_{j+1}) and $(\bar{r}_j, \bar{r}_{j+1})$ coincide up to $2(k - n_0 + n_j)$, or $r_{j+1} = 0$, or $k - n_0 + n_j < n_j - n_{j+1}$.

In the second case, $j + 1 > l$, $j \geq 1$ so $j = h = 1$. In the third case, $h = \eta(k) = j$ by definition of η.

Finally, since

$$\sum_{i=1}^{j} \deg(\bar{q}_i) = \sum_{i=1}^{j} \deg(q_i) \leq \sum_{i=1}^{h} \deg(q_i) \leq k$$

so $j \leq \bar{\eta}(j) = \bar{h}$. □

We will use this result to describe a divide-and-conquer algorithm for the GCD. For this purpose, we want to divide the problem into two subproblems of approximately the same size, taking into account the degrees of the quotients. This is the reason why we introduced $\eta(k)$ (todo...)

Algorithm 17 Fast Half-GCD (Knuth, Strassen)

Input: $f, g \in K[X]$, $n = n_0 = \deg(f) > \deg(g) = n_1$, $k \in \mathbb{N}$ such that $0 \leq k \leq n$

Output: $h = \eta(k) \in \mathbb{N}$, $q_1, \ldots, q_h \in K[X]$, $R_h = Q_h \cdots Q_1 \in K[X]^{2\times 2}$

Base cases

1. If $r_1 = 0$ or $k < n_0 - n_1$, then return 0, the empty sequence and I_2
2. If $k = 0 = n_0 - n_1$, then return 1, $\frac{\mathrm{LC}(f)}{\mathrm{LC}(g)}$ and $\begin{pmatrix} 0 & 1 \\ 1 & -\frac{\mathrm{LC}(f)}{\mathrm{LC}(g)} \end{pmatrix}$

First half of the reductions

3. $r_0 \leftarrow f$, $r_1 \leftarrow g$
4. $d \leftarrow \lceil k/2 \rceil$
5. Call the algorithm recursively with $r_0 \upharpoonright (2d-2)$, $r_1 \upharpoonright (2d-2-(n_0-n_1))$ and $d-1$, obtaining $\eta(d-1)$, $q_1, \ldots, q_{\eta(d-1)}$ and $R^{(1)} = Q_{\eta(d-1)} \cdots Q_1$

Propagating the reductions to the second half

6. $j \leftarrow \eta(d-1)+1$, $\delta \leftarrow \deg(R^{(1)}_{2,2})$
7. $\begin{pmatrix} r_{j-1} \\ r_j \end{pmatrix} \leftarrow R^{(1)} \cdot \begin{pmatrix} r_0 \\ r_1 \end{pmatrix}$
8. If $r_j = 0$ or $k < \delta + \deg(r_{j-1}) - \deg(r_j)$, then return $j-1$, $q_1, \ldots, q_{j-1}$ and $R^{(1)}$

Second half of the reductions

9. $q_j \leftarrow r_{j-1}$ quo r_j
10. $r_{j+1} \leftarrow r_{j-1}$ rem r_j
11. $d \leftarrow k - \delta - (\deg(r_{j-1}) - \deg(r_j))$
12. Call the algorithm recursively with input $r_j \upharpoonright 2d$, $r_{j+1} \upharpoonright (2d - (\deg(r_j) - \deg(r_{j+1})))$ and d, obtaining $\eta(d)$, $q_{j+1}, \ldots, q_h$ and $R^{(2)} = Q_{\eta(d)+j} \ldots Q_{j+1}$
13. Return $\eta(d)+j$, $q_1, \ldots, q_{\eta(d)+1}$ and $R^{(2)} \begin{pmatrix} 0 & 1 \\ 1 & -q_j \end{pmatrix} R^{(1)}$.

Remark 10.11. To compute the GCD of f and g, call the algorithm with $k = n$.

Theorem 10.12. *Algo. 17 requires $O(M(n)\log(n))$ operations in K if $n \leq 2k$.*

Remark 10.13. This is currently the best known complexity for computing the GCD of two polynomials.

Remark 10.14. The algorithm may be optimized further, by dropping trailing coefficients of the polynomials before Step 7. This improves the constant in the complexity, and it eliminates the requirement that $n \leq 2k$. See [1, Algo. 11.6] for details.

Remark 10.15. Generically, $\deg(r_{i+1}) = \deg(r_i) - 1$ and $\eta(k) = k$, and the algorithm really splits the problem into first the first $l/2$ reductions, and then the last $l/2$. In this case, the constant in the complexity can be further improved.

Remark 10.16. The algorithm can return any line in the sequence of reductions of the Extended Euclid's Algorithm, but not all of them. The version presented above return the h'th line, with

$h = \eta(k)$, which corresponds to the line where the sum of the degrees of the quotients is roughly k. This means that it can directly be used for rational reconstruction purposes.

Remark 10.17. If the field is infinite, expression swell is to be expected. This can be mitigated by clearing constants at every step and using homomorphic images.

Remark 10.18. If $h = \gcd(f, g)$ then $\mathrm{rev}_{\deg(h)} h = \gcd(\mathrm{rev}_{\deg(f)} f, \mathrm{rev}_{\deg(g)} g)$? This may be used to gain an extra speed when the trailing coefficients are in some way simpler than the leading coefficients.

Remark 10.19. There is an integer analog of Algo. 17, also running in $O(M(n)\log(n))$ time.

Remark 10.20. Algo. 17 can be used for fast modular inversion, but it is still slower than Newton inversion.

Remark 10.21. There is no speed-up if classical multiplication is used.

Remark 10.22. Kronecker substitution can be used to obtain a fast GCD algorithm over $K[X_1, \ldots, X_n]$.

10.3 Exercises

Exercise 10.1. Show that the relation $\sim_k$ is an equivalence relation.

11 Fast squarefree decomposition

The task in this chapter is, given $f \in K[X]$, find $g_1, \dots, g_m \in K[X]$ such that the g_i's are squarefree and pairwise coprime, and $f = g_1 g_2^2 \cdots g_m^m$.

11.1 Definitions and naive algorithm

Definition 11.1. $(g_1, \dots, g_m)$ is called a *squarefree decomposition* of f. Its components are uniquely determined up to multiplication by elements of K.

The product $\bar{f} = g_1 \cdots g_m$ is called the *squarefree part* of f.

f is called *squarefree* if $f = \bar{f}$, or equivalently if for all $g \in K[X] \setminus K$, $g^2 \nmid f$, or equivalently if f has no multiple factor in its decomposition as a product of primes.

For the moment, we assume that $\mathbb{Q} \subset K$, *i.e.* $n = 1 + \cdots + 1 \neq 0$ for all $n \in \mathbb{N}$.

Proposition 11.2. *Let $f \in K[X]$, write $f' = \frac{\mathrm{d}}{\mathrm{d}X} f$. Then f is squarefree if and only if* $\gcd(f, f') = 1$. *Furthermore,*

$$\gcd(f, f') \cdot \bar{f} = f.$$

Proof. If f is squarefree, write its prime decomposition $f = p_1 p_2 \dots p_m$ where all p_i's are distinct. Then

$$f' = \sum_{i=1}^{m} p_i' \prod_{j \neq i} p_j,$$

therefore

$$p_i \mid f' \iff p_i \mid p_i' \prod_{j \neq i} p_j \iff p_i \mid p_i'.$$

So $p_i \nmid f'$ and therefore $\gcd(f, f') = 1$.

Now suppose that $f = g_1 g_2^2 \dots g_m^m$ with pairwise coprime squarefree g_i's. Then

$$f' = \sum_{i=1}^{m} i g_i' g_i^{i-1} \prod_{j \neq i} g_j^j.$$

So $g_k^{k-1} \mid f'$, but

$$g_k^k \mid f' \iff g_k \mid i g_k' \prod_{j \neq i} g_j^j \iff g_k \mid g_k'$$

so g_k^k does not divide f'. Therefore

$$\gcd(f, f') = g_2 g_3^2 \dots g_m^{m-1}.$$

□

Algorithm 18 Squarefree decomposition

Input: $f \in K[X]$

Output: $(g_1, \dots, g_m)$ the squarefree decomposition of f

1. $u \leftarrow \gcd(f, f')$
2. If $u = 1$ then return (f)
3. Else
4. Recursively compute the squarefree decomposition $(g_2, g_3, \dots, g_m)$ of u
5. Return $\left(\frac{f}{u g_2 \dots g_m}, g_2, \dots, g_m\right)$

Theorem 11.3. *Algorithm 18 requires $O(mM(n)\log(n))$ operations in K.*

11.2 Fast squarefree decomposition

We now want an algorithm running in time $O(M(n)\log(n))$.

Theorem 11.4. *Let $g_1, \dots, g_m \in K[X]$ be squarefree and pairwise coprime, $g = g_1 \dots g_m$ and*

$$h = \sum_{i=1}^{m} c_i g_i' \frac{g}{g_i} \in K[X] \text{ for some } c_i \in K.$$

Then

$$\gcd(g, h - cg') = \prod_{c_j = c} g_j \text{ for all } c \in K.$$

Proof. Since $g' = \sum_{i=1}^m g_i' \frac{g}{g_i}$, we have

$$h - cg' = \sum_{i=1}^{m} (c_i - c) g_i' \frac{g}{g_i}.$$

$\gcd(g_i, g_i' \frac{g}{g_i}) = 1$, because $g_i' \frac{g}{g_i}$ is the product of g_i' and the g_j's with $j \neq i$, each of them being coprime to g_i. Furthermore, for $j \neq i$, $g_j \mid (c_i - c) g_i' \frac{g}{g_i}$. Therefore

$$\gcd(g_i, h - cg') = \gcd(g_i, (c_i - c) g_i' \frac{g}{g_i}) = \gcd(g_i, c_i - c) = \begin{cases} g_i \text{ if } c = c_i \\ 1 \text{ otherwise,} \end{cases}$$

which concludes the proof. □

Now let $(g_1, \ldots, g_m)$ be the squarefree decomposition of $f \in K[X]$. Let $u = \gcd(f, f')$. For $k = 1, \ldots, m$, let

$$V_k = g_k g_{k+1} \cdots g_m$$
$$W_k = \sum_{i=k}^{m} (i - k + 1) g_i' \frac{V_k}{g_i}.$$

Then:

- $V_k' = \sum_{i=k}^{m} g_i' \frac{V_k}{g_i}$
- By Th. 11.4, $\gcd(V_k, W_k - V_k') = g_k$
- $\frac{W_k - V_k'}{g_k} = \sum_{i=k}^{m} (i - k + 1 - 1) g_i' \frac{V_k}{g_i g_k} = \sum_{i=k+1}^{m} (i - (k+1) + 1) g_i' \frac{V_{k+1}}{g_i} = W_{k+1}.$
- $V_1 = \bar{f} = \frac{f}{u}$
- $\frac{f'}{u} = \sum_{i=1}^{m} i g_i' \frac{f}{g_i u} = \sum_{i=1}^{m} (i - 1 + 1) g_i' \frac{V_1}{g_i} = W_1.$

This motivates the following algorithm.

Algorithm 19 Squarefree decomposition (Yun)

Input: $f \in K[X]$

Output: $(g_1, \ldots, g_m)$ the squarefree decomposition of f

1. $u \leftarrow \gcd(f, f')$
2. $k \leftarrow 1, V_1 \leftarrow \frac{f}{u}, W_1 \leftarrow \frac{f'}{u}$
3. Repeat
4. $\quad g_k \leftarrow \gcd(V_k, W_k - V_k')$
5. $\quad V_{k+1} \leftarrow \frac{V_k}{g_k}$
6. $\quad W_{k+1} \leftarrow \frac{W_k - V_k'}{g_k}$
7. $\quad k \leftarrow k + 1$
8. Until $V_k = 1$
9. Return $(g_1, \ldots, g_k)$

Theorem 11.5. *Algo. 19 requires $O(M(n)\log(n))$ operations in K.*

Remark 11.6. This is the best known complexity for finding the square-free decomposition of a polynomial.

Proof. Let $d_k = \deg(g_k)$ for $k = 1, \ldots, m$. Then $\deg(V_k) = \sum_{i=k}^{m} d_i$ and $\deg(W_k) \leq \deg(V_k) - 1 <$

$\deg(V_k)$. Let $T(n)$ be the number of operations required by Algo. 19. We have

$$\begin{aligned} T(n) &= O(M(n)\log(n)) + \sum_{k=1}^{m} O(M(\deg(V_k))\log(\underbrace{\deg(V_k)}_{\leq n})) \\ &= O(M(n)\log(n)) + O\left(M\left(\sum_{k=1}^{m}\deg(V_k)\right)\log(n)\right). \end{aligned}$$

Since

$$\sum_{k=1}^{m}\deg(V_k) = \sum_{k=1}^{m}\sum_{i=k}^{m} d_i = \sum_{i=1}^{m} i d_i = n,$$

we conclude that $T(n) = O(M(n)\log(n))$. □

11.3 Fast squarefree decomposition in $\mathbb{Z}/p\mathbb{Z}$

Algorithms 18 and 19 do not work if K has positive characteristic, for example if $K = \mathbb{F}_p$ for p prime.

Example 11.7. Consider $f = X^3 + 1 \in \mathbb{F}_3[X]$, with $f' = 3X^2 = 0$, so $\gcd(f, f') = f$. Algo. 18 would not terminate and Algo. 19 would return 1.

In fact f is not squarefree: $f = X^3 + 1 = (X+1)^3$.

Theorem 11.8. *For all $f \in \mathbb{F}_p[X]$,*

$$f' = 0 \iff \exists g \in \mathbb{F}_p[X] \text{ such that } g^p = f.$$

Proof. Write $f = f_0 + f_1 X + \ldots$, and assume that $f' = 0$. So $f_1 + 2f_2 X + \cdots = 0$, so for all $k > 0$, $k f_k = 0$. So for all k, p divides k or $f_k = 0$. Then

$$f = f_0 + f_p X^p + f_{2p} X^{2p} + \cdots = g(X^p) = g(X)^p$$

for $g = f_0 + f_p X + f_{2p} X^2 + \ldots$.

Conversely, if $f = g^p$, $f' = p g' g^{p-1} = 0$. □

Let us examine the output of Algo. 19 more closely. It does terminate also for $K = \mathbb{F}_p$, but as we have seen above the output might be incorrect. Actually, if $(g_1, \ldots, g_m)$ is the squarefree decomposition of $f \in K[X]$, Algo. 18 will return $(h_1, \ldots, h_{p-1})$ where for all $i = 1, \ldots, p-1$,

$$h_i = \prod_{j = i \bmod p} g_j.$$

It is not a problem if $m < p$, and in particular if $\deg(f) < p$.

Otherwise, we still have that

$$\begin{aligned} b &:= \frac{f}{h_1 h_2^2 \dots h_{p-1}^{p-1}} \\ &= \frac{g_1 g_2^2 \dots g_m^m}{g_1 g_2^2 \dots g_{p-1}^{p-1} g_{p+1} g_{p+2}^2 \dots g_{2p-1}^{p-1} \dots} \\ &= (g_p g_{p+1} \dots g_{2p-1})^p (g_{2p} g_{2p+1} \dots g_{3p-1})^{2p} \dots \end{aligned}$$

and it is a p'th power. If $(s_1, \dots, s_l)$ is a squarefree decomposition of $b^{1/p}$, then for all $j = 1, \dots, l$

$$s_j = \prod_{i=jp}^{(j+1)p-1} g_i,$$

therefore

$$\begin{aligned} g_i \mid h_j &\iff i = j \bmod p \\ g_i \mid s_j &\iff j = \left\lfloor \frac{i}{p} \right\rfloor. \end{aligned}$$

Algorithm 20 Squarefree decomposition in $\mathbb{F}_p$

Input: $f \in \mathbb{F}_p[X]$

Output: $(g_1, \dots, g_m)$ the squarefree decomposition of f

1. Call Algo. 19 on f, obtaining $(h_1, \dots, h_{p-1}) = (h_1, \dots, h_k, 1, \dots, 1)$
2. $b \leftarrow \dfrac{f}{h_1 h_2^2 \dots h_{p-1}^{p-1}}$
3. If $b = 1$ then return $(h_1, \dots, h_k)$
4. Recursively compute the squarefree decomposition $(s_1, \dots, s_l)$ of $b^{1/p}$
5. $g_{jp+i} \leftarrow \gcd(h_i, s_j)$ $(i = 1, \dots, p-1, j = 1, \dots, l)$
6. $g_{jp} \leftarrow \dfrac{s_j}{g_{jp+1} \dots g_{jp+p-1}}$ $(j = 1, \dots, l)$
7. $g_i \leftarrow \dfrac{h_i}{g_{p+i} \dots g_{lp+i}}$ $(i = 1, \dots, p-1)$
8. Return $(g_1, g_2, \dots)$

12 The LLL algorithm

Throughout this chapter, $\mathbb{R}^n$ refers to the real euclidean vector space of dimension n, equipped with the scalar product $(v, w) = v_1w_1 + \cdots + v_nw_n$ and the norm $||v|| = \sqrt{(v, v)}$.

12.1 Lattices

Definition 12.1. Let $b^{(1)}, \ldots, b^{(m)} \in \mathbb{R}^n$. The $\mathbb{Z}$-module $L = b^{(1)}\mathbb{Z} + \cdots + b^{(n)}\mathbb{Z} \subset \mathbb{R}^n$ is called a *lattice*. The set $B = \{b^{(1)}, \ldots, b^{(m)}\}$ is called a basis of L.

Example 12.2. $\mathbb{Z}^n$ (with the canonical basis)is a lattice, and so are all its submodules.

Example 12.3. The set $L = \{(u\sqrt{2}, v\sqrt{3}) : u, v \in \mathbb{Z}\}$ is a lattice.

Example 12.4. The set $L = \mathbb{Z}\begin{pmatrix}5\\2\end{pmatrix} + \mathbb{Z}\begin{pmatrix}4\\1\end{pmatrix}$ is a lattice. Observe that $\begin{pmatrix}1\\1\end{pmatrix}$ lies in L and is shorter than both $\begin{pmatrix}5\\2\end{pmatrix}$ and $\begin{pmatrix}4\\1\end{pmatrix}$. An alternative basis is $\begin{pmatrix}1\\1\end{pmatrix}, \begin{pmatrix}0\\3\end{pmatrix}$.

It is not yet the shortest basis for L: it is $\begin{pmatrix}1\\1\end{pmatrix}, \begin{pmatrix}-2\\1\end{pmatrix}$.

The goal in this chapter is, given a lattice $L \subset \mathbb{R}^n$ (by means of its basis), to find:

1. $\operatorname{argmin}_{x \in L \setminus \{0\}} ||x||$, that is a nonzero vector of L with minimal length;
2. a basis of L consisting of vectors that are as short as can be.

Both problems are NP-hard, so with the state of today's knowledge, we cannot hope to describe algorithms which are both efficient and exact. Our goal is to find an efficient approximate algorithm, in the sense that it will find vectors of L which are at most

$$c \times \text{ length of the shortest vector in } L,$$

where the constant c depends only on n (not on L).

12.2 Basis orthogonalization

Algorithm 21 Gram-Schmidt orthogonalization

Input: a basis $b^{(1)}, \dots, b^{(m)}$ of some subspace $U \subset \mathbb{R}^n$

Output: a basis $\bar{b}^{(1)}, \dots, \bar{b}^{(m)}$ of U such that $(\bar{b}^{(i)}, \bar{b}^{(j)}) = 0$ if $i \neq j$

1. For k from 1 to m
2. $$\bar{b}^{(k)} \leftarrow b^{(k)} - \sum_{j=1}^{k-1} \frac{(b^{(k)}, \bar{b}^{(j)})}{(\bar{b}^{(j)}, \bar{b}^{(j)})} \bar{b}^{(j)}$$
3. Return $\bar{b}^{(1)}, \dots, \bar{b}^{(m)}$

Remark 12.5. By construction, we have the following properties:

1. $\bar{b}^{(1)}, \dots, \bar{b}^{(k)}$ generate the same subspace U_k as $b^{(1)}, \dots, b^{(k)}$;
2. $\bar{b}^{(k)}$ is the orthogonal projection of $b^{(k)}$ on $U_{k-1}^{\perp}$;
3. $||\bar{b}^{(k)}|| \leq ||b^{(k)}||$;
4. $(\bar{b}^{(k)}, \bar{b}^{(j)}) = 0$ if $j \neq k$
5. Let $\mu_{k,j} = \frac{(b^{(k)}, \bar{b}^{(j)})}{(\bar{b}^{(j)}, \bar{b}^{(j)})}$, then

$$b^{(k)} = \bar{b}^{(k)} + \sum_{j=1}^{k-1} \mu_{k,j} \bar{b}^{(j)}, \tag{12.1}$$

so

$$\begin{pmatrix} b^{(1)} \\ \vdots \\ b^{(m)} \end{pmatrix} = \begin{pmatrix} 1 & & 0 \\ & \ddots & \\ \mu_{k,j} & & 1 \end{pmatrix} \begin{pmatrix} \bar{b}^{(1)} \\ \vdots \\ \bar{b}^{(m)} \end{pmatrix}$$

6. If $n = m$, $\det(\bar{b}^{(1)}, \dots, \bar{b}^{(m)}) = \det(b^{(1)}, \dots, b^{(m)})$

Remark 12.6. The output of Algo. 21 depends on the order of the input.

Theorem 12.7. *Let $b^{(1)}, \dots, b^{(m)} \in \mathbb{R}^n$ be linearly independent vectors, and L the lattice that they generate. Let $\bar{b}^{(1)}, \dots, \bar{b}^{(m)}$ be the orthogonalized basis computed by Algo. 21. Then for all $x \in L \setminus \{0\}$,*

$$||x|| \geq \min_{1 \leq i \leq m} ||\bar{b}^{(i)}||.$$

Proof. Let $x = \sum_{i=1}^{m} c_i b^{(i)}$ be a non-zero vector in L. Let k be maximal such that $c_k \neq 0$, using Eq. (12.1), we can write

$$x = c_k \bar{b}^{(k)} + \sum_{j<k} \lambda_j \bar{b}^{(j)}$$

for some $\lambda_j \in \mathbb{R}$. Note that all terms in the sum are orthogonal. Therefore

$$||x||^2 = c_k||\bar{b}^{(k)}||^2 + \sum_{j<k} \lambda_j^2||\bar{b}^{(k)}||^2 \geq c_k^2||\bar{b}^{(k)}||^2.$$

Since $c_k \in \mathbb{Z} \setminus \{0\}$,

$$||x||^2 \geq ||\bar{b}^{(k)}||^2.$$

□

12.3 Description of the LLL algorithm

If $\bar{b}^{(1)}, \ldots, \bar{b}^{(m)}$ as computed by Gram-Schmidt's algorithm form a basis of L, then the shortest vector of L will be among the $\bar{b}^{(i)}$. But usually the $\bar{b}^{(i)}$ are outside of L.

The idea to move forward is to approximate the $\bar{b}^{(i)}$ with elements of L.

Definition 12.8. Let $b^{(1)}, \ldots, b^{(m)}$ be linearly independent and $\bar{b}^{(1)}, \ldots, \bar{b}^{(m)}$ be as computed by Algo. 21. Then $(b^{(1)}, \ldots, b^{(m)})$ is called a *reduced* basis if for all $k \in \{1, \ldots, m-1\}$,

$$||\bar{b}^{(k)}||^2 \leq 2||\bar{b}^{(k+1)}||^2.$$

Theorem 12.9. *Let $(b^{(1)}, \ldots, b^{(m)})$ be a reduced basis of a lattice $L \subset \mathbb{R}^n$, and let $x \in L \setminus \{0\}$. Then*

$$||b^{(1)}|| \leq 2^{(m-1)/2}||x||.$$

Proof. Let $\bar{b}^{(1)}, \ldots, \bar{b}^{(m)}$ be as computed by Algo. 21. Then

$$||b^{(1)}||^2 = ||\bar{b}^{(1)}||^2 \leq 2||\bar{b}^{(2)}||^2 \leq \cdots \leq 2^{m-1}||\bar{b}^{(m)}||^2.$$

By Theorem 12.7,

$$x \geq \min ||\bar{b}^{(i)}|| \geq 2^{-(m-1)/2}||\bar{b}^{(1)}|| = 2^{-(m-1)/2}||b^{(1)}||.$$

□

$b^{(1)}, \ldots, b^{(m)}$ will be "close" to $\bar{b}^{(1)}, \ldots, \bar{b}^{(m)}$ if the $|\mu_{k,j}|$ are small. The LLL algorithm proceeds by modifying the input basis in Algo. 21 such as to minimize the numbers, until the basis is reduced.

Algorithm 22 LLL algorithm (A. Lenstra, H. Lenstra, L. Lovacz)

Input: $B = (b^{(1)}, \dots, b^{(m)}) \in \mathbb{Z}^n$, linearly independent over $\mathbb{Q}$, generating a lattice L

Output: a reduced basis of L

1. Compute $\bar{B} = (\bar{b}^{(1)}, \dots, \bar{b}^{(m)})$ and the coefficients $\mu_{k,j}$ by Algo. 21
2. $M \leftarrow \begin{pmatrix} 1 & & 0 \\ & \ddots & \\ \mu_{k,j} & & 1 \end{pmatrix}$
3. $k \leftarrow 2$
4. While $k \leq m$ do
5. For j from $k-1$ down to 1 do
6. $b^{(k)} \leftarrow b^{(k)} - \left\lfloor \mu_{k,j} + \frac{1}{2} \right\rfloor b^{(j)}$
7. Update $\bar{B}$ and M
8. If $k > 1$ and $||\bar{b}^{(k-1)}||^2 > 2||\bar{b}^{(k)}||^2$ then
9. Swap $b^{(k-1)}$ and $b^{(k)}$
10. Update $\bar{B}$ and M
11. $k \leftarrow k - 1$
12. Else
13. $k \leftarrow k + 1$
14. Return $b^{(1)}, \dots, b^{(m)}$

Remark 12.10. For $\mu \in \mathbb{R}$, $\left\lfloor \mu + \frac{1}{2} \right\rfloor$ is the integer nearest to μ. In the literature, it is sometimes denoted $\lfloor \mu \rceil$.

First consider the first loop of the algorithm (Steps 5 to 7).

Lemma 12.11. *Let $k \leq m$, and let $B = (b^{(1)}, \dots, b^{(m)})$, $\bar{B} = (\bar{b}^{(1)}, \dots, \bar{b}^{(m)})$ and $M = (\mu_{i,j})$ be the values of the corresponding variables before Step 5. Let $C = (c^{(1)}, \dots, c^{(m)})$, $\bar{C} = (\bar{c}^{(1)}, \dots, \bar{c}^{(m)})$ and $N = (\nu_{i,j})$ be the values of B, $\bar{B}$ and M before Step 8 (after exiting the first loop). Then*

- *$\bar{C} = \bar{B}$;*
- *the rows $1, \dots, k-1$ in M and N are equal;*
- *for all $j \in \{1, \dots, k-1\}$, $|\nu_{k,j}| \leq \frac{1}{2}$.*

Proof. We prove it by downwards induction on $j \in \{1, \dots, k\}$ that, after the pass through the loop with value j,

1. for all $l \in \{j, \dots, k\}$, $\bar{c}^{(l)} = \bar{b}^{(l)}$;
2. the rows $1, \dots, k-1$ in M and N are equal;
3. for all $l \in \{j, \dots, k-1\}$, $|\nu_{k,l}| \leq \frac{1}{2}$.

The initial case $j = k$ is trivial.

Assume that the properties hold for $j + 1 \leq k$, and consider Step 7 for j. The recomputation does not change $\bar{B}$:

- $\bar{b}^{(j)}$ for $j < k$ has no reason to change;

- $\bar{b}^{(k)}$ is the projection of $b^{(k)}$ onto the subspace orthogonal to $U_{k-1} = \mathrm{Span}(b^{(1)}, \dots, b^{(k-1)})$, so it is also the orthogonal projection of $b^{(k)} + v$ for any $v \in U_{k-1}$;
- $\bar{b}^{(j)}$ for $j > k$ is unchanged because all the previous $\bar{b}^{l}$ are unchanged.

As a consequence, the rows $1, \dots, k-1$ of M and N are equal.

Now consider the k'th row of M. For brevity, let $\lambda = \left\lfloor \mu_{k,j} + \frac{1}{2} \right\rfloor$. We have:

$$\begin{aligned}
\bar{c}^{(k)} = \bar{b}^{(k)} &= b^{(k)} - \sum_{l=1}^{k-1} \mu_{k,l} \bar{b}^{(l)} \\
&= b^{(k)} - \lambda b^{(j)} - \sum_{l=1}^{k-1} \mu_{k,l} \bar{b}^{(l)} + \lambda b^{(j)} \\
&= c^{(k)} - \sum_{\substack{l=1 \\ l \neq j}}^{k-1} \mu_{k,l} \bar{b}^{(l)} - \mu_{k,j} \bar{b}^{(j)} + \lambda b^{(j)} \\
&= c^{(k)} - \sum_{l=1}^{j-1} \left(\mu_{k,l} - \lambda \mu_{j,l} \right) \bar{b}^{(l)} - \left(\mu_{k,j} - \lambda \right) \bar{c}^{(j)} - \sum_{l=j+1}^{k-1} \mu_{k,l} \bar{c}^{(l)}
\end{aligned}$$

So the $\mu_{k,l}$ with $l > j$ are unchanged, and $\mu_{k,j}$ becomes $\mu_{k,j} - \left\lfloor \mu_{k,j} + \frac{1}{2} \right\rfloor$, which lies in $\left[-\frac{1}{2}, \frac{1}{2} \right]$. □

Now we consider the second modification done by the algorithm.

Lemma 12.12. *At Step 10, after swapping $b^{(k-1)}$ and $b^{(k)}$:*

- *in the basis $\bar{B}$, the only changes are in the entries $\bar{b}^{(k-1)}$ and $\bar{b}^{(k)}$;*
- *in the matrix M, the only changes are in the rows $k-1$ and below (so the rows $k-1, k, \dots, m$).*

Proof. For the first statement, the basis vectors $\bar{b}^{(j)}$, with $j < k-1$, are unchanged because they do not depend on $b^{(l)}$ for $l \geq k-1 > j$. And the basis vectors $\bar{b}^{(j)}$, with $j > k$, are unchanged because they are orthogonal projections onto $U_j^{\perp}$, which does not change when we permute elements of the basis.

As a consequence, the only modified entries in the matrix M are those which depend on the vectors $b^{(k-1)}$ and $b^{(k)}$ (rows $k-1$ and k) and on the vectors $\bar{b}^{(k-1)}$ and $\bar{b}^{(k)}$ (columns $k-1$ and k). Since the matrix is triangular by construction, all entries in the rows $1, \dots, k-2$ of those two columns are zero. □

This equips us to prove the main loop invariant of the algorithm. An immediate corollary will be that the output of the algorithm is necessarily a reduced basis of the lattice. Then all that will remain to do is to bound the number of loops to prove that the algorithm terminates and give complexity estimates.

Proposition 12.13. *The following invariant holds every time Algorithm 22 enters the loop 4:*

$$|\mu_{j,i}| \leq \frac{1}{2} \text{ for } 1 \leq i < j < k$$
$$||\tilde{b}^{(i-1)}||^2 \leq 2||\tilde{b}^{(i)}||^2 \text{ for } 1 < i < k.$$

Proof. If $k = 1$ there is nothing to prove, so the invariants are true on the very first loop of the algorithm. We will prove that they remain true from one loop to the next.

Assume that the invariants hold at the beginning of a loop, with value k. By Lemma 12.11, before Step 8

$$|\mu_{k,i}| \leq \frac{1}{2} \text{ for } 1 \leq i < k,$$

and, since the rows $1, \ldots, k-1$ of M are unchanged,

$$|\mu_{j,i}| \leq \frac{1}{2} \text{ for } 1 \leq i < j < k.$$

Now consider the two cases in Step 8. If no swap is needed, the second invariant already holds for $i = k$, and by hypothesis is already held for $1 < i < k$, so it holds for all i such that $1 < i < k+1$. And from the above, the first invariant also holds for all i, j such that $1 \leq i < j < k+1$.

If a swap was needed, then by Lemma 12.12, the basis elements $\tilde{b}^{(1)}, \ldots, \tilde{b}^{(k-2)}$ and the rows $1, \ldots, k-2$ of the matrix M are unchanged by the swap, so the first invariant still holds for all i, j such that $1 \leq i < j < k-1$ and the second invariant still holds for all i, j such that $1 < i < k$. □

Corollary 12.14. *If Algorithm 22 terminates, its output is a reduced basis.*

12.4 Termination and complexity

Theorem 12.15. *Algorithm 22 terminates in at most $O(n^2 \log(A))$ iterations of the "while" loop, where A is a bound on $||b^{(1)}||, \ldots, ||b^{(m)}||$.*

Proof. For $k \in \{1, \ldots, m\}$, let

$$B_k = \begin{pmatrix} b^{(1)} \\ \vdots \\ b^{(k)} \end{pmatrix} \in \mathbb{Z}^{k \times n}, \tilde{B}_k = \begin{pmatrix} \tilde{b}^{(1)} \\ \vdots \\ \tilde{b}^{(k)} \end{pmatrix} \in \mathbb{Z}^{k \times n}$$

and let $d_k = \det(B_k \cdot B_k^T)$. Since $\tilde{B}_k$ is the matrix of an orthogonal basis,

$$\tilde{B}_k \tilde{B}_k^T = \begin{pmatrix} ||\tilde{b}^{(1)}||^2 & & \\ & \ddots & \\ & & ||\tilde{b}^{(k)}||^2 \end{pmatrix}$$

Let M_k be the matrix formed with the first k rows and columns of M, so that

$$B_k = M_k \bar{B}_k$$

and therefore

$$B_k B_k^T = M_k \bar{B}_k \bar{B}_k^T M_k^T$$

so

$$d_k = \det(\bar{B}_k \bar{B}_k^T) = \prod_{i=1}^{k} ||\bar{b}^{(i)}||^2$$

This immediately implies (via Lemma 12.11) that in Steps 5 to 7, d_k does not change.

Consider now Step 10 when rows $i-1$ and i were just swapped. If $k \neq i-1$, the effect of the swap is just a permutation of the vectors of the basis, and the determinant of the permutation matrix is ± 1, so d_k is unchanged.

If $k = i-1$, let $(b_*^{(1)}, \ldots, b_*^{(k)})$, $(\bar{b}_*^{(1)}, \ldots, \bar{b}_*^{(k)})$ and d_k^* be the new value of $(b^{(1)}, \ldots, b^{(k)})$, $(\bar{b}^{(1)}, \ldots, \bar{b}^{(k)})$ and d_k respectively after the recomputation:

$$\begin{aligned} d_{i-1}^* &= \prod_{j=1}^{i-1} ||\bar{b}_*^{(j)}||^2 = \prod_{j=1}^{i-2} ||\bar{b}_*^{(j)}||^2 \cdot ||\bar{b}_*^{(i-1)}||^2 \\ &= d_{i-1} \cdot \frac{||\bar{b}_*^{(i-1)}||^2}{||\bar{b}^{(i-1)}||^2}. \end{aligned}$$

Since $\bar{b}_*^{(i-1)}$ is the orthogonal projection of

$$b_*^{(i-1)} = b^{(i)} = \bar{b}^{(i)} + \sum_{j=1}^{i-1} \mu_{k,j} \bar{b}^{(j)}$$

onto $U_{k-2}^{\perp}$,

$$\bar{b}_*^{(i-1)} = \bar{b}^{(i)} + \mu_{i,i-1} \bar{b}^{(i-1)},$$

so

$$\begin{aligned} ||\bar{b}_*^{(i-1)}||^2 &\leq \underbrace{||\bar{b}^{(i)}||^2}_{< \frac{1}{2}||\bar{b}^{(i-1)}||^2} + \underbrace{\mu_{i,i-1}^2}_{\leq \frac{1}{4}} ||\bar{b}^{(i-1)}||^2 \\ &< \frac{3}{4} ||\bar{b}^{(i-1)}||^2. \end{aligned}$$

Therefore

$$d_{i-1}^* < \frac{3}{4} d_{i-1}.$$

Now consider

$$D := \prod_{k=1}^{m} d_k = \prod_{k=1}^{m} \prod_{j=1}^{k} ||\bar{b}^{(k)}||^2 = \prod_{k=1}^{m} ||\bar{b}^{(k)}||^{2(m-k+1)} \in \mathbb{N} \setminus \{0\}.$$

From the above, every swap decreases D by a factor of $\frac{3}{4}$, and remains fixed at every other point of the algorithm. This shows that the algorithm terminates: otherwise, successive values of D would form a strictly decreasing sequence of positive integers.

Furthermore, we can bound the initial value of D with

$$D = ||\bar{b}^{(1)}||^{2m} \cdots ||\bar{b}^{(m)}||^2 \leq ||b^{(1)}||^{2m} \cdots ||b^{(m)}||^2 \leq A^{m(m-1)}.$$

So the number of swaps is bounded by

$$\log_{4/3} A^{m(m-1)} = O(m^2 \log(A)) = O(n^2 \log(A))$$

and it also bounds the number of iterations: if S is the number of swaps (where k is decremented), there will be exactly $S + m$ iterations without a swap (where k is incremented) and so $2S + m$ iterations in total. □

Theorem 12.16. *Algorithm 22 requires $O(n^4 \log(A))$ operations in $\mathbb{Z}$.*

Proof. It is a consequence of the previous theorem, noting that Steps 7 and 10 each require $O(n^2)$ operations in $\mathbb{Z}$, only recomputing those values which may have been changed, and that the first step requires $O(n^3)$ operations. □

Theorem 12.17. *The size of the integers constructed during a run of Algorithm 22 is bounded by $O(n \log(A))$. The bit complexity of Algorithm 22 is $O(n^4 \log(A) M(n \log(A)))$.*

Remark 12.18. Algorithm22 can be extended to vectors that are not linearly independent, or to vectors that live in some computable ring R with $\mathbb{Z} \subset R \subset \mathbb{R}$.

Remark 12.19. Algorithm 22 find a vector in L that is no more than $2^{(n-1)/2}$ times as long as the shortest. In practice, it usually finds the shortest.

Remark 12.20. Several non-equivalent notions of *"reduced"* exist in the literature.

12.5 Applications

12.5.1 Factorization in $\mathbb{Q}[X]$

We will see in Chapter 14 an application of the LLL algorithm to factorization of polynomials with rational coefficients.

12.5.2 Integer relations

Let $\xi_1, \dots, \xi_m \in \mathbb{R}$. An *integer relation* between $\xi_1, \dots, \xi_m$ is a vector $(e_1, \dots, e_m) \in \mathbb{Z}^m$ such that

$$e_1\xi_1 + \cdots + e_m\xi_m = 0.$$

Such relations can be found using lattice reduction: let $\varepsilon > 0$ (small), $w \in \mathbb{N}_{>0}$ (large) and let $x_1, \dots, x_m \in \mathbb{Q}$ be approximations of $\xi_1, \dots, \xi_m$ with precision $\varepsilon > 0$, that is

$$|x_i - \xi_i| < \varepsilon \text{ for all } i.$$

Then, if $(e_1, \dots, e_m)$ is an integer relation, so is $(we_1, \dots, we_m)$

$$\begin{aligned} |we_1x_1 + \cdots + we_mx_m| &= |we_1(x_1 - \xi_1) + \cdots + we_m(x_m - \xi_m)| \\ &\leq (|e_1| + \cdots + |e_m|)\, w\varepsilon. \end{aligned}$$

Consider the lattice

$$L = \begin{pmatrix} wx_1 \\ 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix}\mathbb{Z} + \begin{pmatrix} wx_2 \\ 0 \\ 1 \\ \vdots \\ 0 \end{pmatrix}\mathbb{Z} + \cdots + \begin{pmatrix} wx_m \\ 0 \\ 0 \\ \vdots \\ 1 \end{pmatrix}\mathbb{Z} \subset \mathbb{Q}^{m+1}$$

Then the vector

$$\begin{pmatrix} we_1x_1 + \cdots + we_mx_m \\ e_1 \\ \cdots \\ e_m \end{pmatrix} \in L$$

will be short if $(e_1, \dots, e_m)$ is an integer relation, ε is small enough and w is large enough.

More precisely, using the previous bounds, it can be shown that an integer relation with $|e_i| < M$ will be found if

$$\varepsilon < 2^{m/2} \min_{i=1,\dots,m} \frac{|x_i|}{mM}$$

and

$$w > \frac{M\sqrt{m}}{2^{-m/2} \min_{i=1,\dots,m} |x_i| - mM\varepsilon}.$$

12.5.3 Minimal polynomial

If α is an approximation of an algebraic number of degree d, its minimal polynomial can be found by applying the previous technique to $1, \alpha, \alpha^2, \dots, \alpha^d$.

12.5.4 Rational and algebraic reconstruction

Let $p \in \mathbb{Z}$, and $\mathbb{Z}_p = \{\frac{u}{v} \mid v \text{ does not divide } p\} \subset \mathbb{Q}$. Let $\varphi : \frac{u}{v} \in \mathbb{Z}_p \mapsto v^{-1}u \in \mathbb{F}_p$. A number $r = \frac{u}{v}$ can be reconstructed form its image $\varphi(r)$ using the Half-GCD algorithm (see Chapter 8). Alternatively, one can use LLL and the fact that $\begin{pmatrix} u \\ v \end{pmatrix}$ is a short vector of

$$\begin{pmatrix} \varphi(r) \\ 1 \end{pmatrix} \mathbb{Z} + \begin{pmatrix} p \\ 0 \end{pmatrix} \mathbb{Z}.$$

The problem can be generalized to algebraic reconstruction. Let $m \in \mathbb{Q}[X]$ be an irreducible polynomial, and consider the number field $\mathbb{Q}(\alpha) = \mathbb{Q}[X]/m$. Let $\mathbb{Z}_p(\alpha)$ be the subset of $\mathbb{Q}(\alpha)$ consisting of numbers whose denominator is not divisible by p. If m has degree d, elements of $\mathbb{Z}_p(\alpha)$ have the form

$$z = \frac{u_0 + u_1\alpha + \cdots + u_{d-1}\alpha^{d-1}}{v} \text{ where } u_0, \ldots, u_{d-1} \in \mathbb{Z} \text{ and } v \in \mathbb{N} \setminus p\mathbb{N}.$$

If $\varphi(m)$ has a root $\bar{\alpha}$ in $\mathbb{F}_p$, then φ can be extended to $\mathbb{Z}_p(\alpha)$ by setting $\varphi(\alpha) = \bar{\alpha}$.

Example 12.21. Consider the field $\mathbb{Q}[i] = \mathbb{Q}[X]/\langle X^2 + 1\rangle$, and φ the reduction modulo 5. In $\mathbb{F}_5$, $X^2 + 1$ has 2 and 3 as roots. So we can set $\varphi(i) = 2$ and obtain a ring morphism $\mathbb{Z}_5(i) \to \mathbb{F}_5$. Explicitely, this morphism is defined as

$$\varphi : \frac{a + bi}{d} \mapsto d^{-1}a + 2d^{-1}b \bmod 5.$$

In this situation, given $\varphi(z)$, we can reconstruct z by using LLL. Indeed,

$$\begin{pmatrix} u_0 \\ u_1 \\ \vdots \\ u_{d-1} \\ v \end{pmatrix}$$

is a short vector of the lattice

$$\begin{pmatrix} \varphi(z) \\ 0 \\ \vdots \\ 0 \\ 1 \end{pmatrix} \mathbb{Z} + \begin{pmatrix} 0 \\ \varphi(z)/\bar{\alpha} \\ \vdots \\ 0 \\ 1 \end{pmatrix} \mathbb{Z} + \cdots + \begin{pmatrix} 0 \\ 0 \\ \vdots \\ \varphi(z)/\bar{\alpha}^{d-1} \\ 1 \end{pmatrix} \mathbb{Z} + \begin{pmatrix} p \\ 0 \\ \vdots \\ 0 \\ 0 \end{pmatrix} \mathbb{Z} + \cdots + \begin{pmatrix} 0 \\ 0 \\ \vdots \\ p \\ 0 \end{pmatrix} \mathbb{Z} \subset \mathbb{Z}^{d+1}$$

where the divisions by $\bar{\alpha}$ are performed in $\mathbb{F}_p$ and an arbitrary representative in $\mathbb{Z}$ is chosen.

12.5.5 Linear systems over $\mathbb{Z}$

Let $b^{(1)}, \dots, b^{(m)} \in \mathbb{Z}^n$ and consider

$$L = \left\{ x = (x_1, \dots, x_n) \in \mathbb{Z}^m \mid x_1 b^{(1)} + \dots x_m b^{(m)} \right\} \subset \mathbb{Z}^m.$$

L is a lattice, and solving the linear system requires finding generators of this lattice.

Those generators can be found using LLL. Let $w \in \mathbb{N}$, and consider the lattice

$$L^* = \begin{pmatrix} wb^{(1)} \\ 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix} \mathbb{Z} + \begin{pmatrix} wb^{(2)} \\ 0 \\ 1 \\ \vdots \\ 0 \end{pmatrix} \mathbb{Z} + \dots + \begin{pmatrix} wb^{(m)} \\ 0 \\ 0 \\ \vdots \\ 1 \end{pmatrix} \mathbb{Z} \subset \mathbb{Z}^{n+m}.$$

This lattice contains the wanted generators, which will have the form

$$\begin{pmatrix} 0 \\ \vdots \\ 0 \\ x_1 \\ \vdots \\ x_m \end{pmatrix}.$$

If w is large enough, those vectors will be short and appear in a reduced basis of L^*.

12.5.6 Knapsack-type problems

The knapsack problem (KP) is the problem of deciding, given a list of items with their weight together with a minimal weight w and a maximal weight W, whether it is possible to pack items with a total weight at least w and at most W. This problem is NP-complete. Many cryptosystems depend on problems with a similar type as KP.

We consider here the subset sum problem (SSP): given $a_1, \dots, a_n, a \in \mathbb{N}$, does there exist $I \subset \{1, \dots, n\}$ such that $\sum_{i \in I} a_i = a$? Equivalently, we want to decide whether there exists $x_1, \dots, x_n \in \{0, 1\}$ such that $a_1 x_1 + \dots + a_n x_n = a$.

This is also a NP-complete problem.

Remark 12.22. SSP is equivalent to the KP under polynomial-time reductions, because they are both NP-complete. There is also a direct (and trivial) reduction from SSP to KP, by taking $w = W = a$.

A pragmatic attack on SSP is to apply LLL to the lattice

$$L = \begin{pmatrix} -a_1 w \\ 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix} \mathbb{Z} + \begin{pmatrix} -a_2 w \\ 0 \\ 1 \\ \vdots \\ 0 \end{pmatrix} \mathbb{Z} + \cdots + \begin{pmatrix} -a_n w \\ 0 \\ 0 \\ \vdots \\ 1 \end{pmatrix} \mathbb{Z} + \begin{pmatrix} aw \\ 0 \\ 0 \\ \vdots \\ 0 \end{pmatrix} \mathbb{Z} \subset \mathbb{Z}^{n+1}$$

with some large w, and hope for solutions $(0, x_1, \ldots, x_n)$ to show up in a reduced basis.

A lot of cryptosystems based on the knapsack problem or related NP-complete problems can be broken in this way.

13 Factorization in $\mathbb{F}_p[X]$

Given $f \in \mathbb{F}_p[X]$, we want to find $p_1, \ldots, p_n \in \mathbb{F}_p[X]$ irreducible and pairwise distinct, and $e_1, \ldots, e_n \in \mathbb{N}$, such that $f = p_1^{e_1} \cdots p_n^{e_n}$.

13.1 Preliminaries

Remark 13.1. Using the squarefree decomposition algorithm as a preprocessor, we may assume that $e_1, \ldots, e_n = 1$.

Recall the following theorem:

Theorem 13.2. *1. For all $a \in \mathbb{F}_p$, $a^p - a = 0$.*

2. Let $q = p^n$ and let K be a field with q elements. Then K is the splitting field of $X^q - X$ over $\mathbb{F}_p$, that is the smallest field extension of $\mathbb{F}_p$ containing in which $X^q - X$ splits into a product of linear factors. Furthermore, $X^q - X$ is squarefree in $\bar{\mathbb{F}_p}$, so

$$X^q - X = \prod_{a \in K} (X - a).$$

3. If K is a field extension of $\mathbb{F}_p$, then

$$\mathbb{F}_p = \{a \in K : a^p - a = 0\}.$$

4. If K is a finite field and L is a field extension of K, then there exists l such that $|L| = |K|^l$.

13.2 Berlekamp's algorithm

The idea is to reduce factorization to linear algebra.. Consider the $\mathbb{F}_p$-vector space $R := \mathbb{F}_p[X]/\langle f \rangle$. Then the map

$$\begin{array}{rccc} L: & R & \to & R \\ & a & \mapsto & a^p - a \end{array}$$

is $\mathbb{F}_p$-linear.

If $f = p_1 \cdots p_m$ for irreducible pairwise distinct p_i's, then

$$R \simeq \mathbb{F}_p[X]/\langle p_1 \rangle \times \cdots \times \mathbb{F}_p[X]/\langle p_m \rangle.$$

Let $a \in \ker(L)$ then for all $i \in \{1, \dots, m\}$:

$$a^p = a \bmod f \implies a^p = a \bmod p_i$$
$$\implies a \operatorname{rem} p_i \in \mathbb{F}_p \text{ (for any representative of } a \in R = \mathbb{F}_p[X]/\langle f \rangle).$$

And conversely, if for all $i \in \{1, \dots, m\}$, $a^p - a$ is divisible by p_i, then $a^p - a$ is divisible by f and $a \in \ker(L)$. So

$$\beta := \ker(L) \simeq \mathbb{F}_p^m.$$

Remark 13.3. The kernel β of L contains $\mathbb{F}_p$ (as constant polynomials modulo f in R). Under the isomorphism above, it corresponds to the line generated by $(1, \dots, 1)$, that is vectors where all coordinates are equal.

If $a \in R$ is such that $(a \bmod p_1, a \bmod p_2, \dots)$ has some, but not all, components 0, then $\gcd(a, f) = \prod_{p_i | a} p_i$ is a proper divisor of f.

Since each $b \in \ker(L)$ corresponds to some vector in $\mathbb{F}_p^m$, we can simply search through them. But there are many such vectors, p^m of them to be precise.

Instead, fix $b \in \ker(L) \setminus \mathbb{F}_p$, and suppose that it corresponds to a vector

$$(b_1, \dots, b_m) \in \mathbb{F}_p^m$$

with no zero component. Then $b - b_1$ corresponds to the vector

$$b - b_1(1, \dots, 1) = (0, b_2 - b_1, \dots, b_m - b_1).$$

It is therefore enough to search through $b + \alpha$ with $b \in \ker(L)$ fixed and α running through $\mathbb{F}_p$.

Algorithm 23 Factorization in $\mathbb{F}_p[X]$ (Berlekamp)

Input: $f \in \mathbb{F}_p[X]$, $\deg(f) = n > 0$, f squarefree

Output: a proper divisor g of f, or f itself if f is irreducible

1. For $i = 0 \dots n-1$, compute $X^{ip} \operatorname{rem} f = \sum_{j=0}^{n-1} q_{i,j} X^j$
   and set $Q^T \leftarrow ((q_{i,j}))_{i,j=0}^{n-1}$
2. Compute a basis $B = \{b^{(1)}, \dots, b^{(m)}\} \subset \mathbb{F}_p^n$ of $\ker(Q - I_n)$
3. If $m = 1$ then return f
4. For $i = 1, \dots, m$
5. $\quad b \leftarrow b^{(i)}(1, X, \dots, X^{n-1}) \in \mathbb{F}_p[X]$
6. $\quad$ If $\deg(b) = 0$, continue with next i
7. $\quad$ Else, for $\alpha = 0, \dots, p-1$ do
8. $\quad\quad g \leftarrow \gcd(f, b + \alpha)$
9. $\quad\quad$ If $g \neq 1$ then break and return g

Theorem 13.4. *Algorithm 23 is correct.*

Theorem 13.5. *Algorithm 23 requires $O(n^3 + pM(n)\log(n))$ operations in $\mathbb{F}_p$.*

Remark 13.6. If p is large, Algorithm 23 is slow.

13.3 Cantor-Zassenhaus algorithm

We would like to reduce the complexity in p to $\log(p)$.

The idea of the algorithm in this section will be to proceed in two stages:

1. split f into $f = g_1 \cdots g_k$ where g_i is the product of irreducible factors of degree i (*distinct degree splitting*);
2. split each g_i into a product of irreducibles (*equal degree splitting*).

Theorem 13.7. *In $\mathbb{F}_p[X]$, the following identity holds:*

$$X^{p^n} - X = \prod_{\substack{Q \text{ irred.} \\ \text{monic} \\ \deg(Q) | n}} Q.$$

Proof. We first prove that the rhs divides the lhs: if $Q \in \mathbb{F}_p[X]$ is irreducible monic such that $\deg(Q) = d \mid n$, let k be such that $n = dk$. Then $K := \mathbb{F}_p[X]/\langle q \rangle$ is a field with p^d elements. Thus, for all $a \in K$, $a^{p^d} = a$, and so

$$a^{p^n} = a^{p^{kd}} = a^{(p^d)^k} = (\cdots (a^{p^d})^{p^d} \cdots)^{p^d} = a.$$

So $X^{p^n} - X$ is 0 in K, which implies that Q divides $X^{p^n} - X$.

Now we prove that the lhs divides the rhs. Let K be the splitting field of $X^{p^n} - X$. Let $Q \mid X^{p^n} - X$ with Q irreducible with degree d. We have to show that $d \mid n$. Consider the field $\mathbb{F}_p[X]/\langle Q \rangle$. It has p^d elements and it is a subfield of K which has p^n elements, so indeed $d \mid n$. □

With this theorem, we can present an algorithm for distinct degree splitting.

Algorithm 24 Distinct degree splitting

Input: $f \in \mathbb{F}_p[X]$, $\deg(f) = n > 0$, f squarefree

Output: $g_1, \ldots, g_k \in \mathbb{F}_p[X]$ such that $f = g_1 \cdots g_k$ and g_i consists only of factors of degree i

1. $h \leftarrow X$
2. For $k = 1, 2, \ldots$ while $f \neq 1$, do
3. $\quad h \leftarrow h^p \text{ rem } f$ # $h = X^{p^k} \bmod f$
4. $\quad g_k \leftarrow \gcd(h - X, f)$
5. $\quad f \leftarrow \frac{f}{g_k}$
6. Return $g_1, \ldots, g_k$

Theorem 13.8. *Algo. 24 is correct and requires $O(kM(n)\log(np))$ operations in $\mathbb{F}_p$.*

Proof. Let P be an irreducible factor of f, with degree i. By Th. 13.7, P divides $X^{p^i} - X$ and P does not divide $X^{p^j} - X$ for $j < i$. So P still divides f at step $k = i$ in the algorithm, and it divides g_i.

Since the g_k's are made with gcd's with f, they cannot contain any factor which does not divide f.

The complexity is clear once we note that all polynomials involved have degree at most pn. □

Lemma 13.9. *Let p be prime, $q = p^k$ and consider the field $\mathbb{F}_q$. Let $S = \{a^2 : a \in \mathbb{F}_q^\times\}$ be the set of squares in $\mathbb{F}_q^\times$. Then:*

1. $S = \{a \in \mathbb{F}_q^\times : a^{(q-1)/2} = 1\}$
2. $|S| = \frac{q-1}{2}$.

Theorem 13.10. *Let $p \neq 2$ and $f \in \mathbb{F}_p[X]$. Let Q be an irreducible factor of f with degree $d < n$. Then*

$$\operatorname{Prob}_{\substack{A \in \mathbb{F}_p[X] \\ \gcd(A,f)=1 \\ \deg(A)<n}} \left(A^{(p^d-1)/2} - 1 = 0 \bmod Q \right) = \frac{1}{2}.$$

Proof. Let $e = \frac{1}{2}(p^d - 1)$, it is an integer because $p \neq 2$. Let $A \in \mathbb{F}_p[X]$ with degree $< n$ and coprime to f, then it is coprime to Q. Let $K = \mathbb{F}_p[X]/\langle Q \rangle$, it is a field with p^d elements and A is invertible in that field. By Lemma 13.9, we have

$$A^e = A^{\frac{1}{2}(p^d-1)} = \begin{cases} 1 & \text{if } A = B^2 \text{ for some } B \in \mathbb{F}_p[X] \\ -1 & \text{if } A \neq B^2 \text{ for all } B \in \mathbb{F}_p[X] \end{cases}$$

and the cardinality of the set of squares in $K^\times$ is half the cardinality of $K^\times$. □

Algorithm 25 Factorization subroutine

Input: $f \in \mathbb{F}_p[X]$, $p > 2$, $\deg(f) = n > 0$, and $d \in \mathbb{N}$, $d < n$ such that f is the product of m pairwise distinct irreducible polynomials of degree d

Output: h proper factor of f, or FAIL

1. Choose $A \in \mathbb{F}_p[X]$ with $1 \leq \deg(A) < n$ at random
2. $h \leftarrow \gcd(A, f)$
3. If $h \neq 1$ then return h
4. $h \leftarrow A^{(p^d-1)/2} \operatorname{rem} f$
5. $h \leftarrow \gcd(h - 1, f)$
6. If $h \neq 1$ and $h \neq f$ then return h
7. Return FAIL

Theorem 13.11. *Algo. 25 requires $O((d \log(p) + \log(n))M(n))$ operations in $\mathbb{F}_p$ and returns FAIL with probability $< \frac{1}{2^{m-1}} \leq \frac{1}{2}$ if f is not irreducible.*

Proof. Let $g_1, \ldots, g_m$ be the irreducible factors of f. The algorithm returns FAILif, at step 6, either

1. $h = f$, meaning that $A^{(p^d-1)/2} = 1 \bmod g_i$ for all i, or
2. $h = 1$, meaning that $A^{(p^d-1)/2} = -1 \bmod g_i$ for all i.

By Th. 13.10, either event has probability 2^{-m}, so together they have probability 2^{-m+1}. The probability of failure is strictly smaller than that because there is a small chance of success at step 3. □

Algorithm 26 Equal degree splitting

Input: $f \in \mathbb{F}_p[X]$, $p > 2$, $\deg(f) = n > 0$, and $d \in \mathbb{N}$, $d < n$ such that f is the product of m pairwise distinct irreducible polynomials of degree d

Output: the irreducible factors of f

1. If $n = d$ then return f
2. Call Algo. 25 on f, returning h
3. If h is FAILthen
4. Compute the factors of f recursively
5. Else
6. Compute the factors of h and f/h recursively
7. Return the factors

Theorem 13.12. *The* expected *number of operations in $\mathbb{F}_p$ in a run of Algo. 26 is $O\left((d\log(p) + \log(n))\log(n/d)M(n)\right)$.*

Proof. Each recursion level requires $O((d\log(p) + \log(n))M(n))$ operations by the theorem on the complexity of Algo. 25, and because

$$\log(a)M(a) + \log(b)M(b) \leq \log(a+b)M(a+b)$$

and $\deg(h) + \deg(f/h) = \deg(f) = n$.

It remains to show that the expected depth of the recursion tree is $O(\log(n/d)) = O(\log(m))$. Any two irreducible factors g_i, g_j have a chance $\geq \frac{1}{2}$ to be separated in step 2. So with probability $< \frac{1}{2^k}$ they are still not separated at the k'th recursion level. With probability $< \frac{m^2}{2^k}$ there is at least one unseparated pair of factors at level k.

Let $p^{(k)} = \text{Prob}(\text{depth } > k)$, we know so far that

$$p^{(k)} < \frac{m^2}{2^k}.$$

The expected depth of the recursion tree is then

$$\begin{aligned}\sum_{k\geq 1} k\left(p^{(k-1)}-p^{(k)}\right) &= \sum_{k\geq 0}(k+1)p^{(k)} - \sum_{k\geq 0} kp^{(k)}\\ &= \sum_{k\geq 0} p^{(k)}\\ &= \underbrace{\sum_{k\leq 2\log m} \underbrace{p^{(k)}}_{\leq 1}}_{\leq 2\log(m)} + \underbrace{\sum_{k>2\log(m)} \underbrace{p^{(k)}}_{<\frac{m^2}{2^k}}}_{<m^2\sum(\frac{1}{2})^k}\\ &\leq 2\log(m) + \underbrace{m^2\frac{2}{2^{2\log(m)}}}_{=2} = O(\log(m)).\end{aligned}$$

□

Algorithm 27 Factorization in $\mathbb{F}_p[X]$ (Cantor-Zassenhaus)

Input: $f \in \mathbb{F}_p[X]$, $p > 2$, $\deg(f) = n > 0$, f squarefree

Output: the irreducible factors of f

1. Compute the distinct degree splitting $f = g_1 \cdots g_k$ with Algo. 24
2. For d from 1 to k do
3. If $g_d \neq 1$ then compute the irreducible factors of g_d by Algo. 26
4. Return the factors

Theorem 13.13. *Algo. 27 requires an expected number of $O(nM(n)\log(pn))$ operations in $\mathbb{F}_p$.*

Remark 13.14. Algorithms 24, 26 and 27 can be generalized to arbitrary finite fields (*i.e.* fields with p^n elements where p is any prime).

Remark 13.15. Maple and Mathematica use variants of Algorithm 27 for polynomial factorization.

14 Factorization in $\mathbb{Q}[X]$

The task is the same as before: given $f \in \mathbb{Q}[X]$, find $g_1, \dots, g_m \in \mathbb{Q}[X]$ irreducible such that $f = g_1 \cdots g_m$.

In this chapter, we present 4 algorithms:

- an algorithm reconstructing a factorization in $\mathbb{Q}[X]$ from one in $\mathbb{F}_p[X]$ or $\mathbb{Z}/p^l\mathbb{Z}[X]$, with exponential cost in number of operations in $\mathbb{F}_p$, $\mathbb{Z}/p^l\mathbb{Z}$ and $\mathbb{Z}$;
- an algorithm computing a factorization in $\mathbb{Z}/p^l\mathbb{Z}[X]$ in polynomial time;
- LLL's algorithm, performing the reconstruction in polynomial time, but slower in practice than the exponential algorithm;
- von Hoeij's algorithm, performing the reconstruction in polynomial time and more efficient in practice.

14.1 Reconstructing from a factorization in $\mathbb{F}_p[X]$

Without loss of generality, we can assume that:

1. f is squarefree, *i.e.* the g_i's are pairwise distinct
2. $f \in \mathbb{Z}[X]$ (because factoring f is equivalent to factoring cf for any $c \neq 0$, so we can clear denominators)
3. $g_1, \dots, g_m \in \mathbb{Z}[X]$ (Gauss's lemma)

Definition 14.1. Let $h = \sum h_i X^i \in \mathbb{Z}[X]$, the *content* of h is

$$\mathrm{cont}(h) = \gcd(h_i).$$

The *primitive part* of h is

$$\mathrm{pp}(h) = \frac{h}{\mathrm{cont}(h)} \in \mathbb{Z}[X].$$

Proposition 14.2. *If $h_1, h_2 \in \mathbb{Z}[X]$, then*

$$\mathrm{cont}(h_1 h_2) = \pm \mathrm{cont}(h_1)\mathrm{cont}(h_2)$$
$$\mathrm{pp}(h_1 h_2) = \pm \mathrm{pp}(h_1)\mathrm{pp}(h_2).$$

Lemma 14.3 (Gauss)**.** *If $f \in \mathbb{Z}[X]$ factors as $g_1 \cdots g_m$ over $\mathbb{Q}[X]$, then there exist $c_1, \dots, c_m \in \mathbb{Z}$ such that $c_1 g_1, \dots, c_m g_m \in \mathbb{Z}[X]$ and $f = c_1 g_1 \cdots c_m g_m$.*

Proof. It is enough to prove it for a product of two polynomials. Assume that $f = g_1 g_2$ with

$$g_i = \frac{h_i}{d_i}, h_i \in \mathbb{Z}[X], d_i \in \mathbb{N}, \text{ minimal.}$$

The minimality of d_i ensures that d_i does not divide $\text{cont}(g_i)$. Since $f = g_1 g_2$, $d_1 d_2$ divides $\text{cont}(g_1)\text{cont}(g_2)$, so d_1 divides $\text{cont}(g_2)$ and d_2 divides $\text{cont}(g_1)$. So we can write

$$f = \frac{\text{cont}(g_1)\text{cont}(g_2)}{d_1 d_2}\text{pp}(h_1)\text{pp}(h_2) = \frac{\text{cont}(g_1)}{d_2} g_1 \frac{\text{cont}(g_2)}{d_1} g_2.$$

□

If $f = g_1 g_2$ in $\mathbb{Z}[X]$, then $f = g_1 g_2$ in $(\mathbb{Z}/q)[X]$ for all $q \in \mathbb{N}$. But the converse is false: consider $f = X^2 + 1 \in \mathbb{Z}[X]$, it is irreducible over $\mathbb{Q}$ but $X^2 + 1 = (X + 2)(X + 3) in \mathbb{F}_5[X]$. Still, all irreducible factors of $f \in \mathbb{Z}[X]$ will be products of the irreducible factors of f mod q.

The idea is thus to try all combinations!

For $f = f_0 + \cdots + f_n X^n \in \mathbb{Z}[X]$, define $|f| = \sqrt{f_0^2 + \cdots + f_n^2}$.

Theorem 14.4 (Landau - Mignotte). *Let $f, g \in \mathbb{Z}[X]$ with $\deg(f) = n$, $\deg(g) = k$ and $g \mid f$, then*

$$\max_i |g_i| \leq |g| \leq 2^k |f| \leq \sqrt{n+1}\, 2^k \max_i |f_i|.$$

Algorithm 28 Partial factorization over $\mathbb{Z}[X]$

Input: $f \in \mathbb{Z}[X]$, squarefree with degree $\deg(f) = n > 0$

Output: an irreducible factor g of f, of f itself if irreducible

1. Choose a prime $p > B := 2\sqrt{n+1}\, 2^n \max_i |f_i|$ so that $\bar{f} := f \bmod p \in \mathbb{F}_p[X]$ is also squarefree
2. Compute the monic irreducible factors $\bar{g}^{(1)}, \ldots, \bar{g}^{(m)} \in \mathbb{F}_p[X]$ of $\bar{f}$
3. For k from 1 to $m/2$, do
4. For all $G \subset \{\bar{g}^{(1)}, \ldots, \bar{g}^{(m)}\}$ with $|G| = k$, do
5. $h \leftarrow \mathsf{LC}(f) \prod_{g \in G} g$
6. If $g \mid f$ in $\mathbb{Q}[X]$, then return g
7. Return f

Remark 14.5. The conversions from $\mathbb{F}_p$ to $\mathbb{Z}$ are done by taking the representative in $\{-\lceil p/2 \rceil, \ldots, \lceil p/2 \rceil\}$.

Remark 14.6. The algorithm is easily modified such as to produce a full factorisation.

Remark 14.7. In the worst case, up to $2^{n/2}$ attempts are needed before a factor g is found. Indeed, for all $n = 2^k$, there are irreducible polynomials in $\mathbb{Z}[X]$ which split into linear factors modulo any prime.

Remark 14.8. Typically the performance is not that bad.

14.2 p-adic numbers and Hensel's lifting

The main disadvantage of the previous algorithm is that it requires to do computations in large prime fields $\mathbb{F}_p$, where $p \gg 2^{64}$, which are quite slow. To overcome this issue, we use Hensel's lifting.

Definition 14.9. Let $p \in \mathbb{Z}$ be a prime number. The set

$$\mathbb{Z}_p = \left\{ \sum_{n=0}^{\infty} a_n p^n : 0 \leq a_n < p \right\},$$

equipped with the natural addition and multiplication (convolution) is called the ring of *p-adic integers*.

Proposition 14.10. *1. $\mathbb{Z}_p$ is indeed a ring;*

2. $\mathbb{Z}_p \simeq \{\frac{u}{v} \in \mathbb{Q} : p \nmid v\}$;

3. There is a natural inclusion $\mathbb{Z} \hookrightarrow \mathbb{Z}_p$.

Proof. Exercise. □

p-adic numbers can be equipped with a distance and a topology which make them approximations of integers. Moderately more precisely, we say that two p-adic numbers $u, v \in \mathbb{Z}_p$ are "close" iff $u - v = 0 \mod p^l$ for a large exponent l. Equivalently, it means that the first l coefficients of u and v are the same.

Definition 14.11. Let $u = \sum_{n \in \mathbb{N}} a_n p^n$ be a p-adic number. The truncated expansion $\sum_{n=0}^{l-1} a_n p^n \in \mathbb{Z}$ is called a *p-adic approximation* of u of order l.

The idea of the next algorithm will be to find a good p-adic approximation for the coefficients of the factors of f.

Consider the equation

$$f = g \cdot h$$

for given $f \in \mathbb{Z}[X] \hookrightarrow \mathbb{Z}_p[X]$ and unknown g, h. Factorizing f in $\mathbb{Z}_p[X]$ is equivalent to solving the previous equation for g, h in $\mathbb{Z}_p[X]$, which amounts to finding a root of

$$\begin{array}{rccc} F: & \mathbb{Z}_p[X]^2 & \to & \mathbb{Z}_p[X] \\ & g, h & \mapsto & f - g \cdot h. \end{array}$$

We can use Newton iteration for that. Suppose that $g^{(k)}, h^{(k)} \in \mathbb{Z}[X]$ are such that

$$f - g^{(k)} h^{(k)} = 0 \mod p^k.$$

Then there exists $g_k, h_k \in \mathbb{Z}[X]$ with coefficients in $\{0, \ldots, p-1\}$ such that

$$g^{(k+1)} = g^{(k)} + g_k p^k \mod p^{k+1}$$
$$h^{(k+1)} = h^{(k)} + h_k p^k \mod p^{k+1}.$$

Expanding $F(g^{(k)} + g_k p^k, h^{(k)} + h_k p^k)$ as a power series around $(g^{(k)}, h^{(k)})$ yields:

$$F(g^{(k+1)}, h^{(k+1)}) = F(g^{(k)}, h^{(k)}) + \underbrace{\partial_1 F(g^{(k)}, h^{(k)})}_{-h^{(k)}} p^k g_k + \underbrace{\partial_2 F(g^{(k)}, h^{(k)})}_{-g^{(k)}} p^k h_k + p^{2k}(\dots)$$

so, truncating and grouping,

$$0 = (f - g^{(k)}h^{(k)}) - h^{(k)}p^k g_k - g^{(k)}p^k h_k \bmod p^{k+1}.$$

All three terms of this sum are divisible by p^k, so

$$u := \frac{f - g^{(k)}h^{(k)}}{p^k} \bmod p = h^{(0)}g_k + g^{(0)}h_k \bmod p.$$

If $\gcd(g^{(0)}, h^{(0)}) = 1$ in $\mathbb{Z}_p[X]$ then there are unique $s, t \in \mathbb{F}_p[X]$ of degree $< \deg h^{(0)}$ and $< \deg g^{(0)}$ respectively, such that

$$1 = g^{(0)}s + h^{(0)}t \in \mathbb{F}_p[X],$$

and so

$$u = g^{(0)}us + h^{(0)}ut.$$

Let $q = us \operatorname{quo} h^{(0)}$, adding $0 = q(g^{(0)}h^{(0)} - h^{(0)}g^{(0)})$ to the previous equation yields

$$u = g^{(0)} \cdot \left(us - qh^{(0)}\right) + h^{(0)} \cdot \left(ut + qg^{(0)}\right),$$

where:

- u has degree $\leq n$
- $us - qh^{(0)} = us \operatorname{rem} h^{(0)}$ so the first summand has degree $\leq n - 1$
- as a consequence, the second summand has degree $\leq n$, and so $ut + qg^{(0)}$ has degree $\leq \deg(g^{(0)})$.

For any $\alpha \in \mathbb{Z}$, we have the same equality

$$u = g^{(0)} \cdot \underbrace{\left(us - (q-\alpha)h^{(0)}\right)}_{=:h_k} + h^{(0)} \cdot \underbrace{\left(ut + (q-\alpha)g^{(0)}\right)}_{=:g_k},$$

where h_k and g_k are uniquely determined by α, and α is the coefficient of degree n in h_k. If $\mathrm{LC}(h^{(0)}) = 1$, then we can always choose $\alpha = 0$, and it will ensure that $\mathrm{LC}(h^{(k)}) = 1$ for all k.

Algorithm 29 Approximated factorization over $\mathbb{Z}_p[X]$ (Hensel)

Input:
- $f \in \mathbb{Z}[X]$, $\deg f = n$, p prime, $l \in \mathbb{N}$
- $g^{(0)}, h^{(0)} \in \mathbb{F}_p[X]$ coprime such that $f = g^{(0)}h^{(0)} \bmod p$ and $h^{(0)}$ is monic

Output: $g^{(l)}, h^{(l)} \in \mathbb{Z}[X]$, $h^{(l)}$ monic, such that $f = g^{(l)}h^{(l)} \bmod p^{l+1}$

1. Compute $s, t \in \mathbb{F}_p[X]$ with $1 = g^{(0)}s + h^{(0)}t$
2. $g \leftarrow g^{(0)}$, $h \leftarrow h^{(0)}$
3. For k from 1 to l do
4. $\quad u \leftarrow (f - gh)/p^k$
5. $\quad q = us \text{ quo } h^{(0)}$
6. $\quad h \leftarrow h + p^k(us - qh^{(0)})$
7. $\quad g \leftarrow g + p^k(ut + qg^{(0)})$
8. Return (g, h)

Theorem 14.12. *Algo. 29 requires $O(lM(n))$ operations in $\mathbb{F}_p$ or $\mathbb{Z}$.*

Remark 14.13. Algo. 29 can be generalized to an arbitrary number of factors.

We can then use the same idea as Algo. 28, replacing the factorisations in $\mathbb{F}_p$ for large p with factorizations in $\mathbb{Z}/p^l\mathbb{Z}$ where p is small and l is large enough (*i.e.* such that $p^l > B$). This gets rid of the first problem with Algo. 28, which was the computations in a large prime firld.

14.3 LLL-powered factorization

In this section, the goal is to describe an alternative to Algo. 29 with polynomial worst-case complexity.

Theorem 14.14 (Hadamard). *Let $b^{(1)}, \dots, b^{(n)} \in \mathbb{R}^n$ and $B \in \mathbb{R}$ be such that $\max_i ||b_i^{(k)}|| < B$ for $k = 1, \dots, n$. Then*

$$|\det(b^{(1)}, \dots, b^{(n)})| \leq ||b^{(1)}|| \cdots ||b^{(1)}|| \leq n^{n/2}B^n.$$

Proof. If the $b^{(i)}$ are linearly dependent, there is nothing to prove. Assume that they are not, and let $\bar{b}^{(1)}, \dots, \bar{b}^{(n)}$ be their Gram-Schmidt orthogonalization (as computed by Algo. 21). Then

$$\begin{aligned}
|\det(b^{(1)}, \dots, b^{(n)})| &= |\det(\bar{b}^{(1)}, \dots, \bar{b}^{(n)})| \\
&= \sqrt{\det(\bar{b}^{(1)}, \dots, \bar{b}^{(n)}) \det(\bar{b}^{(1)}, \dots, \bar{b}^{(n)})^T} \\
&= \sqrt{\det(\mathrm{Diag}(||\bar{b}^{(1)}||^2, \dots, ||\bar{b}^{(n)}||^2))} \\
&= ||\bar{b}^{(1)}||^2 \cdots ||\bar{b}^{(n)}||^2 \\
&\leq ||b^{(1)}||^2 \cdots ||b^{(n)}||^2 \leq (\sqrt{n}B)^n.
\end{aligned}$$

□

Recall that for a polynomial $f = f_0 + \cdots + f_n X^n \in \mathbb{Z}[X]$, we defined $|f| = \sqrt{f_0^2 + \cdots + f_n^2}$.

Theorem 14.15. *Let $f, g \in \mathbb{Z}[X]$, $\deg f = n > 0$, $\deg g = m > 0$. Suppose that $u \in \mathbb{Z}[X]$ with $\deg u > 0$ is such that $u \mid f$, $u \mid g \in \mathbb{F}_q[X]$ for some $q > 0$ with $|f|^m |g|^n < q$. Then $\gcd(f, g)$ is nontrivial in $\mathbb{Z}[X]$.*

Proof. We will show that $\operatorname{res}(f, g) = 0$, which implies the result, with

$$|\operatorname{res}(f,g)| = \left| \det \begin{pmatrix} f_0 & & & g_0 & & \\ \vdots & \ddots & & \vdots & \ddots & \\ f_n & & f_0 & g_m & & g_0 \\ & \ddots & \vdots & & \ddots & \vdots \\ & & f_n & & & g_m \end{pmatrix} \right|$$

The previous theorem yields that

$$|\operatorname{res}(f,g)| \leq |f|^m |g|^n < q.$$

It suffices to show $\operatorname{res}(f, g) = 0 \bmod q$. But this is true since $u \mid \gcd(f, g) \in \mathbb{F}_q[X]$. □

Now suppose that $u \in \mathbb{Z}[X]$ with $\deg(u) = d > 0$ is monic and $f \in \mathbb{Z}[X]$ with $\deg(f) = n$ is such that $u \mid f \bmod q$ for some q.

If we can find $g \in \mathbb{Z}[X]$ with $\deg(g) = k < n$ and $u \mid g \bmod q$ and $|g|^n < q|f|^{-k}$, then the theorem implies that $\gcd(f, g) \in \mathbb{Z}[X]$ is a proper factor of f.

Let $u = u_0 + \cdots + u_d X^d$ and consider

$$L := \underbrace{\begin{pmatrix} u_0 \\ u_1 \\ \vdots \\ u_d \\ 0 \\ 0 \\ \vdots \\ 0 \end{pmatrix}}_{u} \mathbb{Z} + \underbrace{\begin{pmatrix} 0 \\ u_0 \\ \vdots \\ u_{d-1} \\ u_d \\ 0 \\ \vdots \\ 0 \end{pmatrix}}_{Xu} \mathbb{Z} + \cdots + \underbrace{\begin{pmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ u_0 \\ u_1 \\ \vdots \\ u_d \end{pmatrix}}_{X^{k-d}u} \mathbb{Z} + \begin{pmatrix} q \\ 0 \\ \vdots \\ 0 \\ 0 \\ \vdots \\ 0 \end{pmatrix} \mathbb{Z} + \begin{pmatrix} 0 \\ q \\ \vdots \\ 0 \\ 0 \\ \vdots \\ 0 \end{pmatrix} \mathbb{Z} + \cdots + \begin{pmatrix} 0 \\ \vdots \\ 0 \\ q \\ 0 \\ \vdots \\ 0 \end{pmatrix} \mathbb{Z} \subset \mathbb{Z}^{k+1}.$$

Proposition 14.16. *For $g = g_0 + \cdots + g_k X^k$, we have $\begin{pmatrix} g_0 \\ \vdots \\ g_k \end{pmatrix} \in L \iff u \mid g \bmod q$.*

Proof. "$\Rightarrow$" is obvious. For "$\Leftarrow$", assume that $u \mid g \bmod q$. Then $g = au + bq \in \mathbb{Z}[X]$ for some $a, b \in \mathbb{Z}[X]$ such that $\deg a < \deg g - \deg u = k - d$ and $\deg b \leq k$, with $b = \bar{a}u + \bar{b}$ for $\bar{a}, \bar{b} \in \mathbb{Z}[X]$ (u monic), with $\deg \bar{b} < \deg u = d$. Then

$$g = \underbrace{(a + \bar{a})}_{\deg \leq k-d} u + q \underbrace{\bar{b}}_{\deg < \deg u} \in \mathbb{Z}[X]$$

so $(g_0, \ldots, g_k) \in L$. □

If $g \mid f$ in $\mathbb{Z}[X]$ then by Thm. 14.4 (Landau-Mignotte), $|g| < 2^k|f|$. By Thm. 12.15 (LLL), we can find $\bar{g} \in L$ with

$$|\bar{g}| < 2^{k/2} \cdot 2^k \cdot |f| = 2^{3k/2}|f|.$$

If $(2^{3k/2}|f|)^n < q|f|^{-k}$, for example if $q > 2^{3n^2/2+n}|f|^{2n}$, then the above implies that $\gcd \bar{g}, f$ will be a nontrivial factor of f.

Algorithm 30 LLL factorization

Input: $f \in \mathbb{Z}[X]$ squarefree, $\deg(f) = n$

Output: a proper factor g of f (or f itself if f is irreducible)

1. Find $p \in \mathbb{Z}$ such that $f \bmod p$ is squarefree
2. $B \leftarrow 2 \cdot 2^{3n^2/2+n}|f|^{2n}$, $l \leftarrow \left\lceil \log_p B \right\rceil$
3. Compute $\bar{g}^{(1)}, \ldots, \bar{g}^{(m)} \in \mathbb{F}_p[X]$ the monic irreducible factors of $f \bmod p$ (Algo. 27)
4. Compute $g^{(1)}, \ldots, g^{(m)} \in \mathbb{Z}[X]$ such that $f = \mathrm{LC}(f)\bar{g}^{(1)} \cdots \bar{g}^{(m)} \bmod p^l$ (Algo. 29)
5. Let $u \in \{g^{(1)}, \ldots, g^{(m)}\}$ with maximal degree
6. For k from $\deg(u) + 1$ to $n - 1$ do
7. Define $L \subseteq \mathbb{Z}^{k+1}$ as above
8. Compute a short vector $\bar{g} \in L$ (Algo 22 [LLL]), view $\bar{g}$ as an element of $\mathbb{Z}[X]$
9. $h \leftarrow \gcd(\bar{g}, f)$ in $\mathbb{Q}[X]$
10. If $\deg h > 0$ then return h
11. Return f

Theorem 14.17. *Algo. 30 requires $O(n^6(n + \log(|f|)))$ operations in $\mathbb{Z}$ or $\mathbb{F}_p$.*

Proof. The most expensive step is 8. By Th. 12.16, it requires $O(k^4 \log |u|)$ operations in $\mathbb{Z}$. The size of u is bounded by

$$\begin{aligned}\log |u| &\leq \log \left(2^{3n^2/2+n}|f|^{2n}\right) \\ &= \left(\frac{3}{2}n^2 + n + 2n \log |f|\right) \\ &= O(n(n + \log |f|)).\end{aligned}$$

The cost for the whole loop is therefore bounded by

$$\left(\sum_{k=1}^{n} k^4\right) \log |u| \leq O(n^5)O(n(n + \log |f|)) = O(n^6(n + \log |f|)).$$

□

Remark 14.18.
1. The bit complexity of Algo. 30 is $O(n^6(n+\log|f|))M(n^2(n+\log|f|))(\log n+\log\log|f|)$.
2. A full factorization can be found with the same complexity.
3. In practice, Algo. 30 is worse than Algo. 28 and it is not used by any computer algebra system.

14.4 Van Hoeij's algorithm

In this section, *with* loss of generality but for simplificity, we assume that f is monic.

Definition 14.19. Let $X_1,\dots,X_n$ be indeterminates and $d\in\mathbb{N}$.

1. A polynomial $p\in K[X_1,\dots,X_n]$ is called *symmetric* if for all $\pi\in\mathfrak{S}_n$,
$$p(X_1,\dots,X_n)=p(X_{\pi(1)},\dots,X_{\pi(n)}).$$
The set of symmetric polynomials is denoted by $K[X_1,\dots,X_n]^{\mathfrak{S}_n}$.
2. The *elementary symmetric polynomial* of degree d is
$$e_d(X_1,\dots,X_n)=\sum_{0\le i_1<i_2<\cdots<i_d\le n} X_{i_1}X_{i_2}\cdots X_{i_d}.$$
3. We define the polynomial
$$p_d(X_1,\dots,X_n)=X_1^d+\cdots+X_n^d.$$

Example 14.20.
- $e_1(x,y,z)=x+y+z$
- $e_2(x,y,z)=xy+xz+yz$
- $e_3(x,y,y)=xyz$

Proposition 14.21.
$$K[X_1,\dots,X_n]^{\mathfrak{S}_n}=K[e_1,\dots,e_n]=K[p_1,\dots,p_n].$$

Proof. The second equality comes from the Newton identities
$$ke_k=\sum_{i=1}^{k}(-1)^{i-1}e_{k-i}p_i$$
$$p_k=\sum_{i=1}^{k}(-1)^{i-1}e_ip_{k-i}.$$
□

Definition 14.22. For $f \in K[X]$ monic with $\alpha_1, \dots, \alpha_n \in \bar{K}$ such that

$$f = (X - \alpha_1)(X - \alpha_2) \cdots (X - \alpha_n)$$

and $i \in \mathbb{N}$, the *i'th trace* of f is

$$\mathrm{Tr}_i(f) = \alpha_1^i + \cdots + \alpha_n^i.$$

Theorem 14.23. *Let $f \in \mathbb{Z}[X]$ be monic, K an extension of $\mathbb{Q}$, $g \in K[X]$ monic with degree d such that $g \mid f$ in $K[X]$. Then*

$$g \in \mathbb{Z}[X] \iff \mathrm{Tr}_i(g) \in \mathbb{Z} \text{ for all } i \in \{1, \dots, d\}.$$

Proof. "$\Rightarrow$": Let $\alpha_1, \dots, \alpha_d \in \bar{K}$ be the roots of g. Then

$$\begin{aligned} g &= (X - \alpha_1)(X - \alpha_2) \cdots (X - \alpha_d) \\ &= X^d - e_1(\alpha_1, \dots, \alpha_d)X^{d-1} + \cdots + (-1)^d e_d(\alpha_1, \dots, \alpha_d) \in \mathbb{Z}[X]. \end{aligned}$$

So each $e_i(\alpha_1, \dots, \alpha_d)$ lies in $\mathbb{Z}$. So by Newton's identities, so does each $p_i(\alpha_1, \dots, \alpha_d) = \mathrm{Tr}_i(g)$.

"$\Leftarrow$": If $\mathrm{Tr}_i(g) = p_i(\alpha_1, \dots, \alpha_d) \in \mathbb{Z}$ then, again by Newton's identities, $e_i(\alpha_1, \dots, \alpha_d) \in \mathbb{Q}$, so $g \in \mathbb{Q}[X]$. But g is monic and divides $f \in \mathbb{Z}[X]$, so by Gauss's lemma, $g \in \mathbb{Z}[X]$. □

Let $g_1, \dots, g_m \in \mathbb{Z}[X]$ be monic, irreducible modulo p and such that

$$f = g_1 \cdots g_m \bmod p^l.$$

Every irreducible factor $g \in \mathbb{Z}[X]$ of f can be written as

$$g = g_1^{e_1} \cdots g_m^{e_m} \bmod p^l$$

for some $(e_1, \dots, e_m) \in \{0, 1\}^m$.

The idea for the next algorithm is to use LLL to search for the exponent vector $(e_1, \dots, e_m)$ instead of the coefficient vector of some factor of degree d.

Typically, we can expect that:

1. $m \ll d$
2. $|e_i| \leq 1 \ll |g_k|$.

Searching for $(e_1, \dots, e_m)$ is a linear optimization problem. Indeed, for all polynomials u, v, $\mathrm{Tr}_i(uv) = \mathrm{Tr}_i(u) + \mathrm{Tr}_i(v)$, so

$$\mathrm{Tr}_i(g) = e_1 \mathrm{Tr}_i(g_1) + \cdots + e_m \mathrm{Tr}_i(g_m) \bmod p^l.$$

However, a priori we do not know $\mathrm{Tr}_i(g)$ exactly, but only bounds for it.

Suppose $B \in \mathbb{N}$ is such a bound. Let $t_{i,j} := \mathrm{Tr}_i(g_j) \bmod p^l$, and consider the lattice $L \subseteq \mathbb{Q}^{n+m}$ generated by

$$\begin{pmatrix} t_{11}/B \\ t_{21}/B \\ \vdots \\ t_{n1}/B \\ 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix}, \begin{pmatrix} t_{12}/B \\ t_{22}/B \\ \vdots \\ t_{n2}/B \\ 0 \\ 1 \\ \vdots \\ 0 \end{pmatrix}, \ldots, \begin{pmatrix} t_{1m}/B \\ t_{2m}/B \\ \vdots \\ t_{nm}/B \\ 0 \\ 0 \\ \vdots \\ 1 \end{pmatrix}, \begin{pmatrix} p^{l/B} \\ 0 \\ \vdots \\ 0 \\ 0 \\ 0 \\ \vdots \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ p^{l/B} \\ \vdots \\ 0 \\ 0 \\ 0 \\ \vdots \\ 0 \end{pmatrix}, \ldots, \begin{pmatrix} 0 \\ 0 \\ \vdots \\ p^l/B \\ 0 \\ 0 \\ \vdots \\ 0 \end{pmatrix}.$$

Then for every factor $g = g_1^{e_1} \cdots g_m^{e_m}$ of f, there is a vector

$$e = \begin{pmatrix} \bullet \\ \vdots \\ \bullet \\ e_1 \\ \vdots \\ e_m \end{pmatrix} \in L \text{ with } |e| \leq \sqrt{n+m}.$$

Conversely, by Th. 14.23, any $e \in L$ with $|e| \leq \sqrt{n+m}$ gives rise to a factor g of f if p^l is sufficiently big.

Algorithm 31 Factorization in $\mathbb{Z}[X]$ (van Hoeij)

Input: $f \in \mathbb{Z}[X]$ squarefree, $\deg(f) = n$

Output: $g_1, \ldots, g_r \in \mathbb{Z}[X]$ the monic irreducible factors of f

1. Find $p \in \mathbb{Z}$ such that $f \bmod p$ is squarefree
2. Compute $\bar{g}_1, \ldots, \bar{g}_m \in \mathbb{F}_p[X]$ the monic irreducible factors of $f \bmod p$ (Algo. 27)
3. Let B be a bound on $\mathrm{Tr}_i(g)$ for any irreducible factor g of f and $i = 1, \ldots, n$
4. For $l = 2, 3, 4, \ldots$ do
5. Compute $\bar{g}^{(1)}, \ldots, \bar{g}^{(m)} \in \mathbb{Z}[X]$ monic such that $f = \bar{g}^{(1)} \cdots \bar{g}^{(m)} \bmod p^l$
6. Let $L \subseteq \mathbb{Q}^{n+m}$ be defined as above and compute a reduced basis $b^{(1)}, \ldots, b^{(s)}$ of L (Algo. 22 [LLL])
7. $\{e^{(1)}, \ldots, e^{(r)}\} \leftarrow \{\pi(b^{(i)}) : |b^{(i)}| \leq \sqrt{n+m}\} \cap \{0,1\}^m$ where $\pi : \mathbb{Q}^{n+m} \to \mathbb{Q}^m$ drops the top n coordinates
8. For i from 1 to r, do $g^{(i)} \leftarrow \bar{g}_1^{e_1^{(i)}} \cdots \bar{g}_m^{e_m^{(i)}}$
9. If $f = g^{(1)} \cdots g^{(r)}$ in $\mathbb{Z}[X]$ then return $g^{(1)}, \ldots, g^{(r)}$.

Here are some possible improvements:

1. Before entering the loop, perform a partial factor combination as in Algo. 28 (say, up to subsets of size 3) and only treat the remaining $\bar{g}_i$ as described above.

2. Instead of

$$\begin{pmatrix} t_{1i}/B \\ t_{2i}/B \\ \vdots \\ t_{ni}/B \\ 0 \\ 0 \\ \vdots \\ 1 \\ \vdots \\ 0 \end{pmatrix} \in \mathbb{Q}^{n+m}$$

as generators of the lattice, use

$$\begin{pmatrix} A \cdot \begin{pmatrix} t_{1i} \\ \vdots \\ \vdots \\ t_{ni} \end{pmatrix} /B \\ 0 \\ \vdots \\ 1 \\ \vdots \\ 0 \end{pmatrix} \in \mathbb{Q}^{s+m}$$

where $A \in \mathbb{Z}^{s \times n}$ is a random matrix with small entries, and s is small but slowly increasing with l.

3. Even better, for $l > 2$, use

$$\begin{pmatrix} A \begin{pmatrix} t_{11} & \dots & t_{1m} \\ \vdots & & \vdots \\ t_{n1} & \dots & t_{nm} \end{pmatrix} \cdot \pi(b^{(i)})/B \\ \\ \pi(b^{(i)}) \end{pmatrix} \in \mathbb{Q}^{s+m}$$

where the $b^{(i)}$ are the vectors of the reduced basis from the previous iteration.

Remark 14.24. 1. With the improvements above, Algo. 31 is the fastest known algorithm for factorization in $\mathbb{Q}[X]$ in practice.

2. It is not easy to give a complexity bound.