Algoritmos de Álgebra Computacional

Curso 2017-2018

Carlos Armero Cantó
Miguel Benito Parejo
Javier Sagredo Tamayo

Algoritmo de Euclides
Algoritmo de Euclides Extendido

Modern Computer Algebra

Joachim von zur Gathen, Jürgen Gerhard

Pág 47 - 48

LEMMA 3.4. *The* gcd *in* $\mathbb{Z}$ *has the following properties, for all* $a, b, c \in \mathbb{Z}$.

(i) $\gcd(a,b) = |a| \iff a \mid b$,

(ii) $\gcd(a,a) = \gcd(a,0) = |a|$ *and* $\gcd(a,1) = 1$,

(iii) $\gcd(a,b) = \gcd(b,a)$ *(commutativity)*,

(iv) $\gcd(a,\gcd(b,c)) = \gcd(\gcd(a,b),c)$ *(associativity)*,

(v) $\gcd(c\cdot a, c\cdot b) = |c|\cdot\gcd(a,b)$ *(distributivity)*,

(vi) $|a| = |b| \Longrightarrow \gcd(a,c) = \gcd(b,c)$.

For a proof, see Exercise 3.3. Because of the associativity, we may write

$$\gcd(a_1,\ldots,a_n) = \gcd(a_1,\gcd(a_2,\ldots,\gcd(a_{n-1},a_n)\ldots)).$$

The following algorithm computes greatest common divisors in an arbitrary Euclidean domain. The nonuniqueness of quotient and remainder complicates the description a bit. For now, "$\gcd(a,b)$" stands for any element which is a gcd of a and b, and we assume some functions quo and rem which associate to any two elements $a, b \in R$ with $b \neq 0$ unique $q = a$ quo b and $r = a$ rem b in R with $a = qb + r$ and $d(r) < d(b)$. Section 3.4 fixes the notation so that gcd has only a single value.

ALGORITHM 3.5 Traditional Euclidean Algorithm.
Input: $f, g \in R$, where R is a Euclidean domain with Euclidean function d.
Output: A greatest common divisor of f and g.

1. $r_0 \longleftarrow f, \quad r_1 \longleftarrow g$
2. $i \longleftarrow 1$
   while $r_i \neq 0$ **do** $r_{i+1} \longleftarrow r_{i-1}$ rem $r_i, \quad i \longleftarrow i+1$
3. **return** r_{i-1}.

For $f = 126$ and $g = 35$, the algorithm works precisely as illustrated at the beginning of this section.

3.2. The Extended Euclidean Algorithm

The following extension of Algorithm 3.5 computes not only the gcd but also a representation of it as a linear combination of the inputs. It generalizes the representation

$$7 = 21 - 1\cdot 14 = 21 - (35 - 1\cdot 21) = 2\cdot(126 - 3\cdot 35) - 35 = 2\cdot 126 - 7\cdot 35,$$

which is obtained by reading the lines of (1) from the bottom up. This important method is called the **Extended Euclidean Algorithm** and works in any Euclidean domain. In various incarnations, it plays a central role throughout this book.

ALGORITHM 3.6 Traditional Extended Euclidean Algorithm.
Input: $f, g \in R$, where R is a Euclidean domain.
Output: $\ell \in \mathbb{N}$, $r_i, s_i, t_i \in R$ for $0 \le i \le \ell+1$, and $q_i \in R$ for $1 \le i \le \ell$, as computed below.

1. $r_0 \longleftarrow f, \quad s_0 \longleftarrow 1, \quad t_0 \longleftarrow 0,$
   $r_1 \longleftarrow g, \quad s_1 \longleftarrow 0, \quad t_1 \longleftarrow 1$
2. $i \longleftarrow 1$
   while $r_i \neq 0$ **do**
   $q_i \longleftarrow r_{i-1}$ quo r_i
   $r_{i+1} \longleftarrow r_{i-1} - q_i r_i$
   $s_{i+1} \longleftarrow s_{i-1} - q_i s_i$
   $t_{i+1} \longleftarrow t_{i-1} - q_i t_i$
   $i \longleftarrow i+1$
3. $\ell \longleftarrow i-1$
   return ℓ, r_i, s_i, t_i for $0 \le i \le \ell+1$, and q_i for $1 \le i \le \ell$

We note that the algorithm terminates because the $d(r_i)$ are strictly decreasing nonnegative integers for $1 \le i \le \ell$, where d is the Euclidean function on R. The elements r_i for $0 \le i \le \ell+1$ are the **remainders** and the q_i for $1 \le i \le \ell$ are the **quotients** in the traditional (Extended) Euclidean Algorithm. The elements r_i, s_i, and t_i form the ***i*th row** in the traditional Extended Euclidean Algorithm, for $0 \le i \le \ell+1$. The central property is that $s_i f + t_i g = r_i$ for all i; in particular, $s_\ell f + t_\ell g = r_\ell$ is a gcd of f and g (see Lemma 3.8 below). We will see later that *all* other intermediate results computed by the algorithm are useful for various tasks in computer algebra.

EXAMPLE 3.7. (i) As in (1), we consider $R = \mathbb{Z}$, $f = 126$, and $g = 35$. The following table illustrates the computation.

i	q_i	r_i	s_i	t_i
0		126	1	0
1	3	35	0	1
2	1	21	1	-3
3	1	14	-1	4
4	2	7	2	-7
5		0	-5	18

We can read off row 4 that $\gcd(126, 35) = 7 = 2 \cdot 126 + (-7) \cdot 35$.

Algoritmo Chino del Resto

Modern Computer Algebra

Joachim von zur Gathen, Jürgen Gerhard

Pág 106

The proof of Theorem 5.2 is constructive and yields the following algorithm.

ALGORITHM 5.4 Chinese Remainder Algorithm (CRA).
Input: $m_0, \ldots, m_{r-1} \in R$ pairwise coprime, $v_0, \ldots, v_{r-1} \in R$, where R is a Euclidean domain.
Output: $f \in R$ such that $f \equiv v_i \bmod m_i$ for $0 \leq i < r$.

1. $m \longleftarrow m_0 \cdots m_{r-1}$
2. **for** $i = 0, \ldots, r-1$ **do**
   compute m/m_i
   call the Extended Euclidean Algorithm 3.14 to compute $s_i, t_i \in R$ with $s_i \dfrac{m}{m_i} + t_i m_i = 1$
   $c_i \longleftarrow v_i s_i \text{ rem } m_i$
3. **return** $\displaystyle\sum_{0 \leq i < r} c_i \frac{m}{m_i}$

We recall that c_i is the remainder in R on dividing $v_i s_i$ by m_i (Section 3.1). To see that the algorithm works correctly, we observe that $c_i m/m_i \equiv 0 \bmod m_j$ for $j \neq i$ and $c_i m/m_i \equiv v_i s_i m/m_i \equiv v_i \bmod m_i$, and hence $f \equiv c_i m/m_i \equiv v_i \bmod m_i$ for $0 \leq i < r$, as claimed.

EXAMPLE 5.5. (i) We let $R = \mathbb{Z}$, $m_i = p_i^{e_i}$ for $0 \leq i < r$, where the $p_i \in \mathbb{N}$ are distinct primes and $e_i \in \mathbb{N}_{>0}$ for $0 \leq i < r$. Then

$$m = \prod_{0 \leq i < r} p_i^{e_i}$$

is the prime decomposition of $m \in \mathbb{Z}$. The CRT tells us that

$$\mathbb{Z}/\langle m \rangle \cong \mathbb{Z}/\langle p_0^{e_0} \rangle \times \cdots \times \mathbb{Z}/\langle p_{r-1}^{e_{r-1}} \rangle,$$

and for arbitrary $v_0, \ldots, v_{r-1} \in \mathbb{Z}$ the CRA computes a solution $f \in \mathbb{Z}$ of the system of congruences

$$f \equiv v_i \bmod p_i^{e_i} \text{ for } 0 \leq i < r.$$

For example, we take $r = 2$, $m_0 = 11$, $m_1 = 13$, and $m = 11 \cdot 13 = 143$, and find for $v_0 = 2$ and $v_1 = 7$ an $f \in \mathbb{Z}$ with $0 \leq f < m$ and

$$f \equiv 2 \bmod 11, \quad f \equiv 7 \bmod 13.$$

There is nothing to do in step 1 of Algorithm 5.4. In step 2, we have to apply the Extended Euclidean Algorithm to 11 and 13 and get $6 \cdot 13 + (-7) \cdot 11 = 1$, that is,

MCD en DFU

Modern Computer Algebra

Joachim von zur Gathen, Jürgen Gerhard

Pág 191

an integral multiple $\alpha_i r_i \in \mathbb{Z}[x]$ which is primitive. This corresponds to taking essentially the primitive part of a polynomial as its normal form in $\mathbb{Q}[x]$; see Exercise 6.5. It is convenient to replace the usual division with remainder by a **pseudo-division** computing $q, r \in \mathbb{Z}[x]$ from $a, b \in \mathbb{Z}[x]$ with

$$\mathrm{lc}(b)^{1+\deg a-\deg b} a = qb + r, \quad \deg r < \deg b$$

(assuming $b \neq 0$). The integer factor multiplied to a ensures that the division can be carried out in $\mathbb{Z}[x]$ (see also Exercise 2.9). This works with any integral domain R instead of $\mathbb{Z}$, and is useful when R is a ring of multivariate polynomials over an integral domain. As in Section 6.2, we assume that we have some normal form normal on R, which we extend to $R[x]$ via $\mathrm{normal}(f) = \mathrm{normal}(\mathrm{lc}(f))f/\mathrm{lc}(f)$.

ALGORITHM 6.61 Primitive Euclidean Algorithm.
Input: Primitive polynomials $f, g \in R[x]$, where R is a UFD, of degrees $n \geq m$.
Output: $h = \gcd(f, g) \in R[x]$.

1. $r_0 \longleftarrow f, \quad r_1 \longleftarrow g, \quad n_0 \longleftarrow n, \quad n_1 \longleftarrow m$

2. $i \longleftarrow 1$
   while $r_i \neq 0$ **do**
   { Pseudodivision }
   $a_{i-1} \longleftarrow \mathrm{lc}(r_i)^{1+n_{i-1}-n_i} r_{i-1}, \quad q_i \longleftarrow a_{i-1} \text{ quo } r_i,$
   $r_{i+1} \longleftarrow \mathrm{pp}(a_{i-1} \text{ rem } r_i), \quad n_{i+1} \longleftarrow \deg r_{i+1}$
   $i \longleftarrow i+1$

3. $\ell \longleftarrow i-1$
   return $\mathrm{normal}(r_\ell)$

THEOREM 6.62.
The algorithm correctly computes the gcd *as specified. We let* $\delta = \max\{n_{i-1} - n_i : 1 \leq i \leq \ell\}$ *be the maximal quotient degree.*

(i) If $R = \mathbb{Z}$ *and* $\|f\|_\infty, \|g\|_\infty \leq A$, *then the max-norm of the intermediate results is at most* $(2(n+1)^n A^{n+m})^{\delta+2}$, *and the algorithm uses* $O(n^3 m \delta^2 \log^2(nA))$ *word operations.*

(ii) If $R = F[y]$ *for a field* F *and* $\deg_y f, \deg_y g \leq d$, *then the degree in* y *of all intermediate results is at most* $(\delta+2)(n+m)d$, *and the time is* $O(n^3 m \delta^2 d^2)$ *operations in* F.

PROOF. Let $\alpha_i = \mathrm{lc}(r_i) \in R$ for all i, and K be the field of fractions of R. As in Exercise 6.47, we find that $r_i/\alpha_i \in K[x]$ is the ith remainder in the Euclidean

Inverso de un elemento en un cuerpo finito

Modern Computer Algebra

Joachim von zur Gathen, Jürgen Gerhard

Pág 73

where $\alpha = (x \bmod (x^3+4x)) \in \mathbb{Z}_5[x]/\langle x^3+4x\rangle$. We have done the calculations in detail in this example to illustrate the principle; later we will suppress the details.

4.2. Modular inverses via Euclid

We have seen in the previous section how modular addition and multiplication works. What about inversion and division? Do expressions like $a^{-1} \bmod m$ and $a/b \bmod m$ make sense, and if so, how can we compute their value? The following theorem gives an answer when the underlying ring R is a Euclidean domain.

THEOREM 4.1.
Let R be a Euclidean domain, $a,m \in R$, and $S = R/mR$. Then $a \bmod m \in S$ is a unit if and only if $\gcd(a,m) = 1$. In this case, the modular inverse of $a \bmod m$ can be computed by means of the Extended Euclidean Algorithm.

PROOF. We have

$$\begin{aligned} a \text{ is invertible modulo } m &\iff \exists s \in R \quad sa \equiv 1 \bmod m \\ &\iff \exists s,t \in R \quad sa+tm = 1 \implies \gcd(a,m) = 1. \end{aligned}$$

If, on the other hand, $\gcd(a,m) = 1$, then the Extended Euclidean Algorithm provides such $s,t \in R$. □

EXAMPLE 4.2. We let $R = \mathbb{Z}$, $m = 29$, and $a = 12$. Then $\gcd(a,m) = 1$, and the Extended Euclidean Algorithm computes $5 \cdot 29 + (-12) \cdot 12 = 1$. Thus $(-12) \cdot 12 \equiv 17 \cdot 12 \equiv 1 \bmod 29$, and hence 17 is the inverse of 12 modulo 29. ◇

EXAMPLE 4.3. Let $R = \mathbb{Q}[x]$, $m = x^3 - x + 2$, and $a = x^2$. The last row in the Extended Euclidean Algorithm for m and a is

$$\Big(\frac{1}{4}x + \frac{1}{2}\Big)(x^3 - x + 2) + \Big(-\frac{1}{4}x^2 - \frac{1}{2}x + \frac{1}{4}\Big)x^2 = 1,$$

and $(-x^2 - 2x + 1)/4$ is the inverse of x^2 modulo $x^3 - x + 2$. ◇

A consequence of Theorem 4.1 is that if $S = \mathbb{Z}_p$ for a prime $p \in \mathbb{N}$ or $S = F[x]/\langle f\rangle$ for a field F and an **irreducible polynomial** $f \in F[x]$, so that f has no nonconstant factor of degree less than $\deg f$, then any element of $S\backslash\{0\}$ is a unit, so that S is a field. We will use the notation $\mathbb{F}_p$ for the finite field $\mathbb{Z}_p$ with p elements throughout the rest of this book. More generally, if $f \in \mathbb{Z}_p[x] = \mathbb{F}_p[x]$ is an irreducible polynomial of degree n, then $\mathbb{F}_p[x]/\langle f\rangle$ is a **finite field** $\mathbb{F}_q$ with $q = p^n$ elements; Section 25.4 gives some basic properties of finite fields. In fact,

Test de irreducibilidad de un polinomio en Fq[x]

Modern Computer Algebra

Joachim von zur Gathen, Jürgen Gerhard

Pág 407

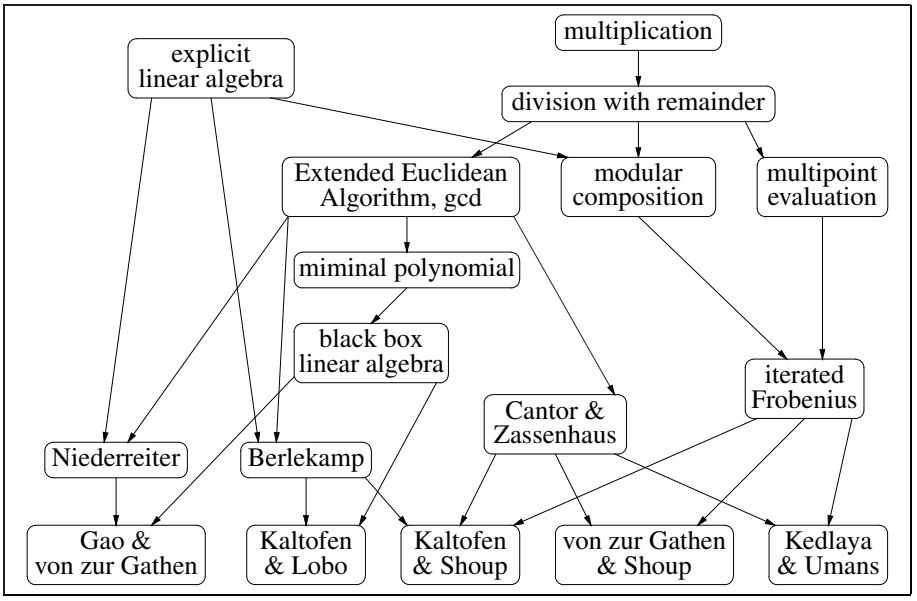

FIGURE 14.10: Algorithms for polynomial factorization over finite fields.

PROOF. It follows immediately from Theorem 14.2 that f satisfies the two conditions if it is irreducible. Conversely, if (i) holds, then Theorem 14.2 implies that the degree of any irreducible factor of f divides n. Let g be such an irreducible factor, and suppose that $d = \deg g < n$. Then d divides n/t for some prime factor t of n, and hence $g \mid x^{q^{n/t}} - x$. This contradicts (ii), and we conclude that $d = n$ and f is irreducible. □

ALGORITHM 14.36 Irreducibility test over finite fields.
Input: $f \in \mathbb{F}_q[x]$ of degree n.
Output: "irreducible" or "reducible".

1. **call** the repeated squaring algorithm 4.8 to compute x^q rem f
   use the modular composition algorithm 12.3 to compute $a = x^{q^n}$ rem f
   if $a \neq x$ **then return** "reducible"

2. **for** all prime divisors t of n **do**

3. use the modular composition algorithm 12.3 to compute $b = x^{q^{n/t}}$ rem f
   if $\gcd(b - x, f) \neq 1$ **then return** "reducible"

4. **return** "irreducible"

Logaritmo discreto

A New Baby-Step Giant-Step Algorithm and Some Applications to Cryptanalysis

Jean Sebastien Coron, David Lefranc and Guillaume Poupard

Pág. 3-4

This paper is organized as follows. After recalling some useful variants of the Baby-Step Giant-Step algorithm in Section 2, we describe in Section 3 our new algorithm for discrete logarithms equal to products of sub-private keys in groups of unknown order. In Section 4, we briefly recall the GPS scheme and the Girault-Lefranc private keys. Then we present two attacks on such private keys: the first one is an application of our new algorithm and the second one makes use of a known variant of the Baby-Step Giant-Step algorithm.

2 Baby-Step Giant-Step Algorithms

In this section, we recall Shanks' *Baby-Step Giant-Step algorithm* [1] and some useful variants. We first specify the notations we use.

Notations Let $G = \langle g \rangle$ be a finite cyclic abelian group generated by the element g and written multiplicatively. Let n be the order of G. As a consequence, we have $G = \left\{g^i;\ i \in [\![0, n-1]\!]\right\}$. For any value v in G, *the discrete logarithm* of v in base g, denoted $\log_g v$, is the unique non-negative integer x less than n such that $v = g^x$.

The *discrete logarithm problem* is to compute $\log_g v$ given g and v.

Let ℓ denote the value $\lceil \log_2 n \rceil$. Then, the binary representation of $x = \log_g v$ requires at most ℓ bits such that we can write

$$x = \sum_{i=0}^{\ell-1} x_i 2^i,$$

where $x_i \in \{0, 1\}$ for $0 \leq i \leq \ell - 1$. The *Hamming weight* of an integer x, denoted $wt(x)$, is equal to the number of 1's in its binary representation.

Let $t < \ell$ be a positive integer. The *Hamming weight t discrete logarithm problem* is to compute $\log_g v$ given g and v with the extra information $wt(\log_g v) = t$.

2.1 The Classical Baby-Step Giant-Step Algorithm

One of the most famous and generic algorithms dealing with the discrete logarithm problem is the so-called Baby-Step Giant-Step algorithm. Introduced by Shanks [1], it is a time-memory trade-off with time complexity $\mathcal{O}\left(\sqrt{n}\right)$ group multiplications.

The algorithm works as follows. Let $m = \lceil n^{1/2} \rceil$. For any given value $v \in \langle g \rangle$, $x = \log_g v$ is less than n so it can be written as $a + b \times m$ with a and b strictly less than m. From the equality $v = g^{\log_g v} = g^{a+b\times m}$, we obtain that $v \times g^{-bm} = g^a$ for some values a and b less than m. Thus, in the two following lists

$$\left(1, g, g^2, \ldots, g^{m-2}, g^{m-1}\right)$$

and

$$\big(v, vg^{-m}, vg^{-2m}, \ldots, vg^{-(m-2)m}, vg^{-(m-1)m}\big),$$

there exists at most two collisions, i.e two couples (g^{a_0},$vg^{-b_0 m}$) and (g^{a_1},$vg^{-b_1 m}$) such that $g^{a_0} = vg^{-b_0 m}$ and $g^{a_1} = vg^{-b_1 m}$. The value x is obtained using the couple with the smallest b_i and $x = a_i + b_i m$. The time complexity of this algorithm mainly relies on the computation of the lists both of which contains m elements. Thus, the time complexity of this algorithm is $\mathcal{O}\big(\lceil n^{1/2} \rceil\big)$ group multiplications. However, in this generic algorithm, the space requirement is also $\mathcal{O}\big(\lceil n^{1/2} \rceil\big)$.

In order to decrease such a large space requirement, Pollard [12] proposed two randomized variants of this algorithm, known as *rho* and *lambda* methods. The generic idea is two find a linear equation over $\log_g v$. The space requirement is then very small and the expected running time of these variants is still $\mathcal{O}\big(\lceil n^{1/2} \rceil\big)$ group multiplications.

2.2 Low Hamming Weight Discrete Logarithms

In 1999, Stinson described some variants [18] of the Baby-Step Giant-Step algorithm in the case of the Hamming weight t discrete logarithm problem, i.e the computation of discrete logarithms for which the Hamming weight is known to be t.

Without loss of generality, let us assume that $\ell = \lceil \log_2 n \rceil$ is even (otherwise we consider $\ell + 1$). This algorithm relies on the concept of *splitting system*.

Definition 1 (Splitting system). *Let t and ℓ be such that $0 < t < \ell$. A (ℓ,t)-splitting system is a pair (X,$\mathcal{B}$) that satisfies:*

- *$|X| = \ell$, and $\mathcal{B}$ is a set of subsets of X, each subset having $\ell/2$ elements.*
- *$\forall\ Y \subset X$ such that $|Y| = t$, $\exists\ B \in \mathcal{B}$ such that $|B \cap Y| = t/2$.*

For example, let t and ℓ be two even integers such that $0 < t < \ell$. Let $X = [\![0, \ell - 1]\!]$ and let $\mathcal{B} = \{B_i\ ;\ 0 \leq i \leq \ell/2 - 1\}$ where for all $0 \leq i \leq \ell/2 - 1$, $B_i = \{i + j \bmod \ell\ ;\ 0 \leq j \leq \ell/2 - 1\}$. The pair ($X$,$\mathcal{B}$) is a ($\ell$,$t$)-splitting system.

Thus, let $v \in \langle g \rangle$ such that $wt\big(\log_g v\big) = t$ (assumed to be even). We now use the above splitting system in the algorithm. Any element of $\langle g \rangle$ is now identified to the set of the positions of the non zero bits involved in the binary representation of its discrete logarithm. Thus, v is identified to a subset $Y \subset X$ of t elements. The goal of the algorithm is to find a decomposition of Y into two subsets of $t/2$ elements using the splitting system.

For all B_i in $\mathcal{B}$
- For all $Y_i^j \subset B_i$ of $t/2$ elements, identify the corresponding value A_i^j in G. Let $\mathcal{L}_1$ be the list of pairs (Y_i^j,A_i^j).
- Consider the set $W_i = [\![0, \ell - 1]\!] \setminus B_i$.

Algoritmo de factorización de un polinomio en cuerpo finito

A computational introduction to number theory and algebra
Victor Shoup
Pág 528, 531

Modern Computer Algebra
Joachim von zur Gathen, Jürgen Gerhard
Pág 387, 385

Proof. The theorem can be restated in terms of the following claim: for each $i = 1, \ldots, r$, we have

- $f_i^{e_i} \mid \mathbf{D}(f)$ if $e_i \equiv 0 \pmod{p}$, and
- $f_i^{e_i - 1} \mid \mathbf{D}(f)$ but $f_i^{e_i} \nmid \mathbf{D}(f)$ if $e_i \not\equiv 0 \pmod{p}$.

To prove the claim, we take formal derivatives using the usual rule for products, obtaining

$$\mathbf{D}(f) = \sum_j e_j f_j^{e_j - 1} \mathbf{D}(f_j) \prod_{k \neq j} f_k^{e_k}. \tag{20.3}$$

Consider a fixed index i. Clearly, $f_i^{e_i}$ divides every term in the sum on the right-hand side of (20.3), with the possible exception of the term with $j = i$. In the case where $e_i \equiv 0 \pmod{p}$, the term with $j = i$ vanishes, and that proves the claim in this case. So assume that $e_i \not\equiv 0 \pmod{p}$. By the previous theorem, and the fact that f_i is irreducible, and in particular, not the pth power of any polynomial, we see that $\mathbf{D}(f_i)$ is non-zero, and (of course) has degree strictly less than that of f_i. From this, and (again) the fact that f_i is irreducible, it follows that the term with $j = i$ is divisible by $f_i^{e_i - 1}$, but not by $f_i^{e_i}$, from which the claim follows. □

This theorem provides the justification for the following square-free decomposition algorithm.

Algorithm SFD**.** On input f, where $f \in F[X]$ is a monic polynomial of degree $\ell > 0$, compute a square-free decomposition of f as follows:

```
initialize an empty list L
s ← 1
repeat
    j ← 1, g ← f / gcd(f, D(f))
    while g ≠ 1 do
        f ← f/g, h ← gcd(f, g), m ← g/h
        if m ≠ 1 then append (m, js) to L
        g ← h, j ← j + 1
    if f ≠ 1 then // f is a pth power
        // compute a pth root as in (20.2)
        f ← f^{1/p}, s ← ps
until f = 1
output L
```

Theorem 20.5. *Algorithm* SFD *correctly computes a square-free decomposition of f using $O(\ell^2 + \ell(w-1)\operatorname{len}(p)/p)$ operations in F.*

Proof. Let $f = \prod_i f_i^{e_i}$ be the factorization of the input f into irreducibles. Let S

and quadratic irreducibles, since we have already removed the linear factors from f, the gcd will give us just the quadratic factors of f. In general, for $k = 1, \ldots, \ell$, having removed all the irreducible factors of degree less than k from f, we compute $\gcd(X^{q^k} - X, f)$ to obtain the product of all the irreducible factors of f of degree k, and then remove these from f.

The above discussion leads to the following algorithm for distinct degree factorization.

Algorithm DDF**.** On input f, where $f \in F[X]$ is a monic square-free polynomial of degree $\ell > 0$, do the following:

```
initialize an empty list L
h ← X mod f
k ← 0
while f ≠ 1 do
    h ← h^q mod f, k ← k + 1
    g ← gcd(h − X, f)
    if g ≠ 1 then
        append (g, k) to L
        f ← f/g
        h ← h mod f
output L
```

The correctness of Algorithm DDF follows from the discussion above. As for the running time:

Theorem 20.6. *Algorithm* DDF *uses* $O(\ell^3 \operatorname{len}(q))$ *operations in* F.

Proof. Note that the body of the main loop is executed at most ℓ times, since after ℓ iterations, we will have removed all the factors of f. Thus, we perform at most ℓ qth-powering steps, each of which takes $O(\ell^2 \operatorname{len}(q))$ operations in F, and so the total contribution to the running time of these is $O(\ell^3 \operatorname{len}(q))$ operations in F. We also have to take into account the cost of the gcd and division computations. The cost per loop iteration of these is $O(\ell^2)$ operations in F, contributing a term of $O(\ell^3)$ to the total operation count. This term is dominated by the cost of the qth-powering steps, and so the total cost of Algorithm DDF is $O(\ell^3 \operatorname{len}(q))$ operations in F. □

factorization in step 2, and the 32 green elements of type square/nonsquare or nonsquare/square are splitting polynomials. The $1+32=33$ other elements are unlucky. In general, only $2q^d-2$ of the q^{rd} elements of R are "blue", so that for larger values of q^d it is very unlikely that the algorithm hits upon one of them by chance.

The representation at left is the one we can compute with. The magic of an isomorphism transforms this chaotic conglomerate into the disciplined diagram at right, about which we can easily reason, derive algorithms and prove their properties; these algorithms then are executed in the messy real world at left. $\diamond$

Algorithm 14.8 gives a factorization into two factors. If we need just one irreducible factor, we can apply the algorithm recursively to the smaller factor (Exercise 14.15). However, we will usually want all r factors, and this can be done by running the algorithm recursively on each factor.

ALGORITHM 14.10 Equal-degree factorization.
Input: A squarefree monic polynomial $f \in \mathbb{F}_q[x]$ of degree $n>0$, for an odd prime power q, and a divisor d of n, so that all irreducible factors of f have degree d.
Output: The monic irreducible factors of f in $\mathbb{F}_q[x]$.

1. **if** $n=d$ **then return** f
2. **call** the equal-degree splitting algorithm 14.8 with input f and d repeatedly until it returns a proper factor $g \in \mathbb{F}_q[x]$ of f
3. **call** the algorithm recursively with input g and with input f/g
   return the results of the two recursive calls

THEOREM 14.11.
A squarefree polynomial of degree $n=rd$ with r irreducible factors of degree d can be completely factored with an expected number of $O((d\log q+\log n)\mathsf{M}(n)\log r)$ or $O^{\sim}(n^2\log q)$ operations in $\mathbb{F}_q$.

PROOF. The workings of Algorithm 14.10 can be illustrated by means of a labeled tree (see Figure 14.6 for an example). The node labels are factors of f, with f at the root and the irreducible factors of f at the leaves. If the call to Algorithm 14.8 in step 2 returns "failure" at a particular node, then the corresponding node has precisely one child with the same label. Otherwise, it has two children labeled g and f/g. The product over all labels at one level of the tree is a divisor of f, and hence the total degree over all nodes at each level is at most n. The cost at a node of degree m is $O((d\log q+\log m)\mathsf{M}(m))$ operations in $\mathbb{F}_q$, by Theorem 14.9, and by the superlinearity of M (Section 8.3), the total cost at each level of the tree is $O((d\log q+\log n)\mathsf{M}(n))$ operations.

We assume that q is odd, and write $e = (q^d - 1)/2$. For any $\beta \in R_i^\times = \mathbb{F}_{q^d}^\times$, we have $\beta^e \in \{1, -1\}$, and both possibilities occur equally often, by Lemma 14.7 with q^d instead of q. If we choose $a \in \mathbb{F}_q[x]$ with $\deg a < n$ and $\gcd(a, f) = 1$ uniformly at random, then $\chi_1(a), \ldots, \chi_r(a)$ are independent uniformly distributed elements of $\mathbb{F}_{q^d}^\times$, and $\varepsilon_i = \chi_i(a^e) \in R_i$ is 1 or -1, each with probability $1/2$. Therefore

$$\chi(a^e - 1) = (\varepsilon_1 - 1, \ldots, \varepsilon_r - 1),$$

and $a^e - 1$ is a splitting polynomial unless $\varepsilon_1 = \cdots = \varepsilon_r$. The latter occurs with probability $2 \cdot (1/2)^r = 2^{-r+1} \leq 1/2$.

ALGORITHM 14.8 Equal-degree splitting.

Input: A squarefree monic polynomial $f \in \mathbb{F}_q[x]$ of degree $n > 0$, where q is an odd prime power, and a divisor $d < n$ of n, so that all irreducible factors of f have degree d.

Output: A proper monic factor $g \in \mathbb{F}_q[x]$ of f, or "failure".

1. choose $a \in \mathbb{F}_q[x]$ with $\deg a < n$ at random
   if $a \in \mathbb{F}_q$ **then return** "failure"

2. $g_1 \longleftarrow \gcd(a, f)$
   if $g_1 \neq 1$ **then return** g_1

3. **call** the repeated squaring algorithm 4.8 in $R = \mathbb{F}_q[x]/\langle f \rangle$ to compute $b = a^{(q^d-1)/2} \text{ rem } f$

4. $g_2 \longleftarrow \gcd(b - 1, f)$
   if $g_2 \neq 1$ and $g_2 \neq f$ **then return** g_2 **else return** "failure"

THEOREM 14.9.

Algorithm 14.8 works correctly as specified. It returns "failure" with probability less than $2^{1-r} \leq 1/2$, where $r = n/d \geq 2$, and takes $O((d \log q + \log n)\mathsf{M}(n))$ or $O^\sim(n^2 \log q)$ operations in $\mathbb{F}_q$.

PROOF. The failure probability has been given above as 2^{1-r} if $\gcd(a, f) = 1$. For general a, where step 2 might find a factor, the failure probability is less than 2^{1-r}. The cost for the gcds in steps 2 and 4 is $O(\mathsf{M}(n) \log n)$, and computing b in step 3 takes at most $2\log_2(q^d) \in O(d \log q)$ multiplications modulo f or $O(\mathsf{M}(n) d \log q)$ operations in $\mathbb{F}_q$. □

The usual trick of running the algorithm k times makes the failure probability less than $2^{(1-r)k} \leq 2^{-k}$.

Algoritmo de factorización de un polinomio en cuerpo finito

Modern Computer Algebra

Joachim von zur Gathen, Jürgen Gerhard

Pág 389-390, 387, 385

Figure 14.6 illustrates the process of a typical execution of Algorithm 14.10 in the form of a tree. The leaves correspond to the isolated irreducible factors. The labels are the indices of the irreducible factors of the polynomial at that node. The numbers in the left column are the levels. For example, the rightmost label at level 2 corresponds to the factor $f_1 f_2 f_6$ of f. The depth 6 is less than our upper bound $\lceil 2\log_2 10 \rceil + 2 = 9$ on the average value. $\diamond$

When q is large enough, there is a way to replace almost all powerings with exponent $(q^d - 1)/2$ in step 3 of Algorithm 14.8 by cheaper powerings with exponent $(q-1)/2$, leading to an expected time of $O(d\,\mathsf{M}(n)\log q + \mathsf{M}(n)\log(qn)\log r)$ operations in $\mathbb{F}_q$ for a variant of Algorithm 14.10 (Exercise 14.17).

14.4. A complete factoring algorithm

It remains to deal with polynomials that are not squarefree. Section 14.6 describes the squarefree factorization stage in detail, but we now discuss a simpler approach by modifying the distinct-degree factorization stage.

The algorithm proceeds as follows. For $i = 1, 2, \ldots$, the (squarefree) product g of all irreducible factors of f of degree i is computed, via one distinct-degree factorization step. Then g is factored into irreducibles, by calling the equal-degree factorization algorithm. For each irreducible factor g_j obtained, we determine its multiplicity e in f by trial division, and then remove g_j^e.

ALGORITHM 14.13 Polynomial factorization over finite fields.
Input: A nonconstant polynomial $f \in \mathbb{F}_q[x]$, where q is an odd prime power.
Output: The monic irreducible factors of f and their multiplicities.

1. $h_0 \longleftarrow x, \quad v_0 \longleftarrow \dfrac{f}{\mathrm{lc}(f)}, \quad i \longleftarrow 0, \quad U \longleftarrow \emptyset$
   repeat

2. $i \longleftarrow i + 1$
   { one distinct-degree factorization step }
   call the repeated squaring algorithm 4.8 in $R = \mathbb{F}_q[x]/\langle f \rangle$ to compute $h_i = h_{i-1}^q \bmod f$
   $g \longleftarrow \gcd(h_i - x, v_{i-1})$

3. **if** $g \neq 1$ **then**
   { equal-degree factorization }
   call Algorithm 14.10 with input g and i to compute the monic irreducible factors $g_1, \ldots, g_s \in \mathbb{F}_q[x]$ of g

4. $v_i \longleftarrow v_{i-1}$
   { determine multiplicities }
   for $j = 1, \ldots, s$ **do**
   $e \longleftarrow 0$
   while $g_j \mid v_i$ **do** $v_i \longleftarrow \dfrac{v_i}{g_j}, \quad e \longleftarrow e+1$
   $U \longleftarrow U \cup \{(g_j, e)\}$
5. **until** $v_i = 1$
6. **return** U

As in the distinct-degree factorization algorithm 14.3, the algorithm may be aborted as soon as $\deg v_i < 2(i+1)$, and h_i need only be computed modulo v_{i-1} in step 2.

THEOREM 14.14.
Algorithm 14.13 correctly computes the irreducible factorization of f. If $\deg f = n$, then it takes an expected number of $O(n\mathsf{M}(n)\log(qn))$ or $O^{\sim}(n^2 \log q)$ arithmetic operations in $\mathbb{F}_q$.

PROOF. Let $f = \mathrm{lc}(f)\prod_{1\le i\le r} f_i^{e_i}$ be the irreducible factorization of f, with distinct monic irreducible polynomials $f_1, \ldots, f_r \in \mathbb{F}_q[x]$ and positive integers $e_1, \ldots, e_r$. We prove that the invariants

$$h_i \equiv x^{q^i} \bmod f, \quad v_i = \mathrm{lc}(f) \prod_{\deg f_k > i} f_k^{e_k}$$

hold each time before the algorithm passes through step 2. The first invariant is shown as in the proof of Theorem 14.4. The second one is clear for $i = 0$, and we may assume that $i \ge 1$. By Theorem 14.2, $x^{q^i} - x$ is the product of all distinct monic irreducible polynomials in $\mathbb{F}_q[x]$ of degree dividing i, and hence in particular it is squarefree. Thus, since $v_{i-1} \mid f$ and by the induction hypothesis, the polynomial

$$g = \gcd(h_i - x, v_{i-1}) = \gcd(x^{q^i} - x, v_{i-1}) = \prod_{\deg f_k = i} f_k$$

is a squarefree polynomial with irreducible factors of degree i only, and $g_1, \ldots, g_s$ are in fact the irreducible factors of g at the end of step 3 if $g \ne 1$. These are then removed with the correct multiplicity from v_i in step 4, and the invariants hold again before the next pass through step 2.

The cost for one execution of step 2 is $O(\mathsf{M}(n)\log(qn))$ operations in $\mathbb{F}_q$. There are at most n iterations of the outer loop, and hence the overall cost for step 2 is $O(n\mathsf{M}(n)\log(nq))$ operations. If $m_i = \deg g$, then step 3, the only probabilistic part

factorization in step 2, and the 32 green elements of type square/nonsquare or nonsquare/square are splitting polynomials. The $1+32=33$ other elements are unlucky. In general, only $2q^d-2$ of the q^{rd} elements of R are "blue", so that for larger values of q^d it is very unlikely that the algorithm hits upon one of them by chance.

The representation at left is the one we can compute with. The magic of an isomorphism transforms this chaotic conglomerate into the disciplined diagram at right, about which we can easily reason, derive algorithms and prove their properties; these algorithms then are executed in the messy real world at left. $\diamond$

Algorithm 14.8 gives a factorization into two factors. If we need just one irreducible factor, we can apply the algorithm recursively to the smaller factor (Exercise 14.15). However, we will usually want all r factors, and this can be done by running the algorithm recursively on each factor.

ALGORITHM 14.10 Equal-degree factorization.
Input: A squarefree monic polynomial $f \in \mathbb{F}_q[x]$ of degree $n>0$, for an odd prime power q, and a divisor d of n, so that all irreducible factors of f have degree d.
Output: The monic irreducible factors of f in $\mathbb{F}_q[x]$.

1. **if** $n=d$ **then return** f
2. **call** the equal-degree splitting algorithm 14.8 with input f and d repeatedly until it returns a proper factor $g \in \mathbb{F}_q[x]$ of f
3. **call** the algorithm recursively with input g and with input f/g
   return the results of the two recursive calls

THEOREM 14.11.
A squarefree polynomial of degree $n=rd$ with r irreducible factors of degree d can be completely factored with an expected number of $O((d\log q+\log n)\mathsf{M}(n)\log r)$ or $O^{\sim}(n^2\log q)$ operations in $\mathbb{F}_q$.

PROOF. The workings of Algorithm 14.10 can be illustrated by means of a labeled tree (see Figure 14.6 for an example). The node labels are factors of f, with f at the root and the irreducible factors of f at the leaves. If the call to Algorithm 14.8 in step 2 returns "failure" at a particular node, then the corresponding node has precisely one child with the same label. Otherwise, it has two children labeled g and f/g. The product over all labels at one level of the tree is a divisor of f, and hence the total degree over all nodes at each level is at most n. The cost at a node of degree m is $O((d\log q+\log m)\mathsf{M}(m))$ operations in $\mathbb{F}_q$, by Theorem 14.9, and by the superlinearity of M (Section 8.3), the total cost at each level of the tree is $O((d\log q+\log n)\mathsf{M}(n))$ operations.

We assume that q is odd, and write $e = (q^d - 1)/2$. For any $\beta \in R_i^\times = \mathbb{F}_{q^d}^\times$, we have $\beta^e \in \{1, -1\}$, and both possibilities occur equally often, by Lemma 14.7 with q^d instead of q. If we choose $a \in \mathbb{F}_q[x]$ with $\deg a < n$ and $\gcd(a, f) = 1$ uniformly at random, then $\chi_1(a), \ldots, \chi_r(a)$ are independent uniformly distributed elements of $\mathbb{F}_{q^d}^\times$, and $\varepsilon_i = \chi_i(a^e) \in R_i$ is 1 or -1, each with probability $1/2$. Therefore

$$\chi(a^e - 1) = (\varepsilon_1 - 1, \ldots, \varepsilon_r - 1),$$

and $a^e - 1$ is a splitting polynomial unless $\varepsilon_1 = \cdots = \varepsilon_r$. The latter occurs with probability $2 \cdot (1/2)^r = 2^{-r+1} \leq 1/2$.

ALGORITHM 14.8 Equal-degree splitting.

Input: A squarefree monic polynomial $f \in \mathbb{F}_q[x]$ of degree $n > 0$, where q is an odd prime power, and a divisor $d < n$ of n, so that all irreducible factors of f have degree d.

Output: A proper monic factor $g \in \mathbb{F}_q[x]$ of f, or "failure".

1. choose $a \in \mathbb{F}_q[x]$ with $\deg a < n$ at random
   if $a \in \mathbb{F}_q$ **then return** "failure"

2. $g_1 \longleftarrow \gcd(a, f)$
   if $g_1 \neq 1$ **then return** g_1

3. **call** the repeated squaring algorithm 4.8 in $R = \mathbb{F}_q[x]/\langle f \rangle$ to compute $b = a^{(q^d-1)/2} \text{ rem } f$

4. $g_2 \longleftarrow \gcd(b - 1, f)$
   if $g_2 \neq 1$ and $g_2 \neq f$ **then return** g_2 **else return** "failure"

THEOREM 14.9.

Algorithm 14.8 works correctly as specified. It returns "failure" with probability less than $2^{1-r} \leq 1/2$, where $r = n/d \geq 2$, and takes $O((d \log q + \log n)\mathsf{M}(n))$ or $O^\sim(n^2 \log q)$ operations in $\mathbb{F}_q$.

PROOF. The failure probability has been given above as 2^{1-r} if $\gcd(a, f) = 1$. For general a, where step 2 might find a factor, the failure probability is less than 2^{1-r}. The cost for the gcds in steps 2 and 4 is $O(\mathsf{M}(n) \log n)$, and computing b in step 3 takes at most $2\log_2(q^d) \in O(d \log q)$ multiplications modulo f or $O(\mathsf{M}(n) d \log q)$ operations in $\mathbb{F}_q$. □

The usual trick of running the algorithm k times makes the failure probability less than $2^{(1-r)k} \leq 2^{-k}$.

Algoritmo de factorización de Berlekamp en cuerpo finito

Modern Computer Algebra

Joachim von zur Gathen, Jürgen Gerhard

Pág 403

ALGORITHM 14.31 Berlekamp's algorithm.
Input: A monic squarefree polynomial $f \in \mathbb{F}_q[x]$ of degree $n > 0$, where q is an odd prime power.
Output: Either a proper factor g of f, or "failure".

1. **call** the repeated squaring algorithm 4.8 in $\mathbb{F}_q[x]/\langle f\rangle$ to compute x^q rem f
2. **for** $i = 0,\ldots,n-1$ compute x^{qi} rem $f = \sum_{0\le j<n} q_{ij}x^j$
   $Q \longleftarrow (q_{ij})_{0\le i,j<n}$
3. use Gaussian elimination on $Q - I \in \mathbb{F}_q^{n\times n}$, where I is the $n \times n$ identity matrix, to compute the dimension $r \in \mathbb{N}$ and a basis $b_1 \bmod f,\ldots,b_r \bmod f$ of the Berlekamp algebra $\mathcal{B}$, with $b_1,\ldots,b_r \in \mathbb{F}_q[x]$ of degree less than n
   if $r = 1$ **then return** f
4. choose independent uniformly random elements $c_1,\ldots,c_r \in \mathbb{F}_q$
   $a \longleftarrow c_1b_1 + \cdots + c_rb_r$
5. $g_1 \longleftarrow \gcd(a, f)$
   if $g_1 \neq 1$ and $g_1 \neq f$ **then return** g_1
6. **call** the repeated squaring algorithm 4.8 in $R = \mathbb{F}_q[x]/\langle f\rangle$ to compute $b = a^{(q-1)/2}$ rem f
7. $g_2 \longleftarrow \gcd(b-1, f)$
   if $g_2 \neq 1$ and $g_2 \neq f$ **then return** g_2 **else return** "failure"

The Gaussian elimination in step 3 can be done with $O(n^3)$ operations in $\mathbb{F}_q$. In Section 12.1 we have seen faster methods, and we may use any feasible matrix multiplication exponent ω, so that $n \times n$ matrices can be multiplied with $O(n^\omega)$ operations. The reader may very well think of the classical $\omega = 3$.

THEOREM 14.32.
Algorithm 14.31 works correctly as specified and returns "failure" with probability at most $1/2$. *It uses* $O(n^\omega + \mathsf{M}(n)\log q)$ *operations in* $\mathbb{F}_q$ *if* $\omega > 2$.

PROOF. Correctness is clear from the discussion preceding the algorithm. In order to analyze the failure probability, we note that a is a uniformly random element of $\mathcal{B}$, so that $u_i \equiv a \bmod f_i$ for $1 \le i \le r$ are independent random elements of $\mathbb{F}_q$ (via its embedding in $\mathbb{F}_q[x]/\langle f\rangle$). If some u_i is zero and some u_j nonzero, a factor is returned in step 5. With probability q^{-r}, all u_i's are zero. All u_i's are nonzero with probability $(1-q^{-1})^r$, and then each $v_i = u_i^{(q-1)/2}$ is 1 or -1 with probability 2^{-1} for either case, and all v_i's are equal with probability $2\cdot 2^{-r}$. Thus failure occurs

Algoritmo de factorización en Z[x]

Modern Computer Algebra

Joachim von zur Gathen, Jürgen Gerhard

Pág 436-437

15.6. Factoring using Hensel lifting: Zassenhaus' algorithm

Let $R = \mathbb{Z}$ and $f \in \mathbb{Z}[x]$ be a squarefree primitive polynomial of degree n. We will now replace step 3 of the big prime algorithm 15.2 by factoring modulo a "small" prime $p \in \mathbb{N}$ and subsequent Hensel lifting. As in Algorithm 15.2, we will choose p in such a way that $f \bmod p$ is squarefree and of the same degree as f, so that p does not divide $\operatorname{res}(f, f')$. The bound on the resultant from Theorem 6.23 implies that this is an easy task; the details are given later in Corollary 18.12.

ALGORITHM 15.19 Factorization in $\mathbb{Z}[x]$ (prime power version).

Input: A squarefree primitive polynomial $f \in \mathbb{Z}[x]$ of degree $n \geq 1$ with $\operatorname{lc}(f) > 0$ and max-norm $\|f\|_\infty = A$.

Output: The irreducible factors $\{f_1, \ldots, f_k\} \subseteq \mathbb{Z}[x]$ of f.

1. **if** $n = 1$ **then return** $\{f\}$
   $b \longleftarrow \operatorname{lc}(f)$, $\quad B \longleftarrow (n+1)^{1/2} 2^n A b$,
   $C \longleftarrow (n+1)^{2n} A^{2n-1}$, $\quad \gamma \longleftarrow \lceil 2 \log_2 C \rceil$

2. **repeat** choose a prime $p \leq 2\gamma \ln \gamma$, $\quad \overline{f} \longleftarrow f \bmod p$
   until $p \nmid b$ and $\gcd(\overline{f}, \overline{f}') = 1$ in $\mathbb{F}_p[x]$
   $l \longleftarrow \lceil \log_p (2B+1) \rceil$

3. { modular factorization }
   compute $h_1, \ldots, h_r \in \mathbb{Z}[x]$ of max-norm at most $p/2$ that are nonconstant, monic, and irreducible modulo p, such that $f \equiv b h_1 \cdots h_r \bmod p$

4. { Hensel lifting }
   call Algorithm 15.17 to compute a factorization $f \equiv b g_1 \cdots g_r \bmod p^l$ with monic polynomials $g_1, \ldots, g_r \in \mathbb{Z}[x]$ of max-norm at most $p^l/2$ such that $g_i \equiv h_i \bmod p$ for $1 \leq i \leq r$

5. { initialize the index set T of modular factors still to be treated, the set G of factors found, and the polynomial f^* still to be factored }
   $T \longleftarrow \{1, \ldots, r\}$, $\quad s \longleftarrow 1$, $\quad G \longleftarrow \emptyset$, $\quad f^* \longleftarrow f$

6. { factor combination }
   while $2s \leq \#T$ **do**

7. **for** all subsets $S \subseteq T$ of cardinality $\#S = s$ **do**

8. compute $g^*, h^* \in \mathbb{Z}[x]$ with max-norm at most $p^l/2$ satisfying $g^* \equiv b \prod_{i \in S} g_i \bmod p^l$ and $h^* \equiv b \prod_{i \in T \setminus S} g_i \bmod p^l$

9. **if** $\|g^*\|_1 \|h^*\|_1 \leq B$ **then**
   $T \longleftarrow T \setminus S$, $\quad G \longleftarrow G \cup \{\operatorname{pp}(g^*)\}$,
   $f^* \longleftarrow \operatorname{pp}(h^*)$, $\quad b \longleftarrow \operatorname{lc}(f^*)$
   break the loop 7 and **goto** 6

10. $s \longleftarrow s+1$

11. **return** $G \cup \{f^*\}$

There are several ways to find suitable primes in step 2: we might try the small primes $2,3,5,\ldots$ one after the other, or we might use a single precision prime just below the processor's word length from a precalculated list. Both approaches work well in practice, but do not yield a generally valid result since for some particular input all primes from any fixed list might divide the discriminant. Another alternative which provably works is to choose p randomly; the required number theoretic arguments will be discussed in Section 18.4.

THEOREM 15.20.
Algorithm 15.19 works correctly. We have $\gamma \in O(n\log(nA))$, and the expected cost of steps 2 and 3 is

$$O\Big(\gamma\log^2\gamma\,\mathrm{loglog}\,\gamma + (\mathsf{M}(n^2)+\mathsf{M}(n)\log\gamma)\log n\cdot\mathsf{M}(\log\gamma)\,\mathrm{loglog}\,\gamma\Big)$$

or $O^\sim(n^2+n\log A)$ word operations. Step 4 takes $O(\mathsf{M}(n)\mathsf{M}(n+\log A)\log n)$ or $O^\sim(n^2+n\log A)$ word operations. The cost for one execution of steps 8 and 9 is

$$O((\mathsf{M}(n)\log n + n\log(n+\log A))\mathsf{M}(n+\log A)) \text{ or } O^\sim(n^2+n\log A)$$

word operations, and there are at most 2^{n+1} iterations.

PROOF. The correctness proof of Theorem 15.3 carries over with the following modifications. We replace the congruence in (2) by $f^* \equiv b\prod_{i\in T} g_i \bmod p^l$. In one part of that proof we assume that the condition in step 9 is false for all subsets $S \subseteq T$ of cardinality s, but that f^* has an irreducible factor $g \in \mathbb{Z}[x]$ with $\mu(g) = s$, and the fact that $\mathbb{F}_p[x]$ is a UFD yields a set $S \subseteq T$ of cardinality s such that the condition in step 9 is true for that particular subset. Now $\mathbb{Z}_{p^l}[x]$ is not a UFD in general (it even has nonzero zero divisors), and we have to replace the argument by unique factorization in $\mathbb{F}_p[x]$ plus an appeal to the uniqueness of Hensel lifting (Theorem 15.14). Namely, let $h = f^*/g$ and $S \subseteq T$ with $\#S = s$ be such that $\mathrm{lc}(h)g \equiv b\prod_{i\in S} h_i \bmod p$ and $\mathrm{lc}(g)h \equiv b\prod_{i\in T\setminus S} h_i \bmod p$. Now for that same subset S, let $g^* \equiv b\prod_{i\in S} g_i \bmod p^l$ and $h^* \equiv b\prod_{i\in T\setminus S} g_i \bmod p^l$. Thus $bf^* \equiv \mathrm{lc}(h)g\cdot\mathrm{lc}(g)h \bmod p^l$ and $bf^* \equiv g^*h^* \bmod p^l$ are both liftings of the same factorization of bf^* modulo p, and the uniqueness of Hensel lifting (Theorem 15.14) implies that $\mathrm{lc}(h)g \equiv g^* \bmod p^l$ and $\mathrm{lc}(g)h \equiv h^* \bmod p^l$. Now $B < p^l/2$, by the choice of l, and as in the proof of Theorem 15.3, we arrive at the contradiction that $\|g^*\|_1\|h^*\|_1 \le B$ holds in step 9.

Corollary 18.12 says that with $O(\gamma\log^2\gamma\,\mathrm{loglog}\,\gamma)$ word operations, we can find a random p in step 2, and that p divides $\mathrm{disc}(f)$ with probability at most $1/2$.

common root of $\varphi_1,\dots,\varphi_n$ obtains a better one a^* as $a^* = a - J^{-1}(a)f(a)$, where $J = (\partial\varphi_i/\partial y_j)_{1\le i,j\le n} \in R[y_1,\dots,y_n]^{n\times n}$ is the **Jacobian** of φ.

We have seen in Example 15.8 that Hensel lifting generalizes Newton iteration. But Hensel lifting can also be considered as a special case of multivariate Newton iteration, as follows. We regard the coefficients of g and h in a factorization $f = gh$ as indeterminates. Comparing the coefficients of $x^{n-1},\dots,x,1$ on both sides, we obtain n equations for the n unknown coefficients, and a common solution to those equations corresponds to a factorization of f. Invertibility of J is equivalent to coprimality of g and h (Exercise 15.21).

15.5. Multifactor Hensel lifting

In this section, we will employ the Hensel step algorithm 15.10 to lift a factorization into arbitrarily many factors. Let R be a ring, $p \in R$, and $g,h \in R[x]$. We say that g and h are **Bézout-coprime** modulo p if there exist $s,t \in R[x]$ such that $sg + th \equiv 1 \bmod p$. If $R/\langle p\rangle$ is a field, then this just means that $g \bmod p$ and $h \bmod p$ are coprime in the usual sense.

ALGORITHM 15.17 Multifactor Hensel lifting.

Input: An element p in a ring R (commutative, with 1), $f \in R[x]$ of degree $n \ge 1$ such that $\mathrm{lc}(f)$ is a unit modulo p, monic nonconstant polynomials $f_1,\dots,f_r \in R[x]$ that are pairwise Bézout-coprime modulo p and satisfy $f \equiv \mathrm{lc}(f)f_1\cdots f_r \bmod p$, and $l \in \mathbb{N}$.

Output: Monic polynomials $f_1^*,\dots,f_r^* \in R[x]$ with $f \equiv \mathrm{lc}(f)f_1^*\cdots f_r^* \bmod p^l$ and $f_i^* \equiv f_i \bmod p$ for all i.

1. **if** $r = 1$ **then** compute $f_1^* \in R[x]$ with $f \equiv \mathrm{lc}(f)f_1^* \bmod p^l$ and **return** f_1^*
2. $k \longleftarrow \lfloor r/2\rfloor, \quad d \longleftarrow \lceil \log_2 l\rceil$
3. compute $g_0,h_0 \in R[x]$ such that $g_0 \equiv \mathrm{lc}(f)f_1\cdots f_k \bmod p$ and $h_0 \equiv f_{k+1}\cdots f_r \bmod p$
4. compute $s_0,t_0 \in R[x]$ such that $s_0g_0 + t_0h_0 \equiv 1 \bmod p$, $\deg s_0 < \deg h_0$, and $\deg t_0 < \deg g_0$, using the Extended Euclidean Algorithm if $R/\langle p\rangle$ is a field and Exercise 15.29 otherwise
5. **for** $j = 1,\dots,d$ **do**
6. **call** the Hensel step algorithm 15.10 with $m = p^{2^{j-1}}$ to lift the congruences $f \equiv g_{j-1}h_{j-1}$ and $s_{j-1}g_{j-1} + t_{j-1}h_{j-1} \equiv 1$ modulo $p^{2^{j-1}}$ to congruences $f \equiv g_jh_j$ and $s_jg_j + t_jh_j \equiv 1$ modulo p^{2^j}
7. $g \longleftarrow g_d, \quad h \longleftarrow h_d$

8. **call** the algorithm recursively to compute $f_1^*, \ldots, f_k^* \in R[x]$ satisfying $g \equiv \mathrm{lc}(g) f_1^* \cdots f_k^* \bmod p^l$ and $f_i^* \equiv f_i \bmod p$ for $1 \le i \le k$
9. **call** the algorithm recursively to compute $f_{k+1}^*, \ldots, f_r^* \in R[x]$ satisfying $h \equiv f_{k+1}^* \cdots f_r^* \bmod p^l$ and $f_i^* \equiv f_i \bmod p$ for $k < i \le r$
10. **return** $f_1^*, \ldots, f_r^*$

THEOREM 15.18.
Algorithm 15.17 works correctly as specified.

(i) If $R = \mathbb{Z}$, $p \in \mathbb{N}$ is prime, $\|f\|_\infty < p^l$, and $\|f_i\|_\infty < p$ for all i, then the algorithm takes

$$O\Big((\mathsf{M}(n)\mathsf{M}(l\mu) + \mathsf{M}(n)\log n \cdot \mathsf{M}(\mu) + n\mathsf{M}(\mu)\log\mu)\log r\Big)$$

or $O^\sim(nl\mu)$ word operations, where $\mu = \log p$.

(ii) If $R = F[y]$ for a field F, p is irreducible, $\deg_y f < l \deg_y p$, and $\deg_y f_i < \deg_y p$ for all i, then the algorithm takes

$$O\Big((\mathsf{M}(n)\mathsf{M}(l\mu) + \mathsf{M}(n)\log n \cdot \mathsf{M}(\mu) + n\mathsf{M}(\mu)\log\mu)\log r\Big)$$

or $O^\sim(nl\mu)$ arithmetic operations in F, where $\mu = \deg_y p$.

PROOF. Correctness follows from Theorem 15.11 by induction on j and on r. We only prove the running time estimate (i); case (ii) is similar and left as Exercise 15.22. Let $n_i = \deg f_i$ for $1 \le i \le r$, and let $T(n_1, \ldots, n_r)$ denote the running time of the algorithm in word operations. For $r = 1$, we first compute in step 1 $\mathrm{lc}(f)^{-1} \bmod p$ by the Extended Euclidean Algorithm, taking $O(\mathsf{M}(\mu)\log\mu)$ word operations, by Corollary 11.11. Then we use the Newton iteration algorithm 9.10 to lift the inverse modulo p^l, in time $O(\mathsf{M}(l\mu))$, by Theorem 9.12. Multiplying all coefficients of f by $\mathrm{lc}(f)^{-1}$ modulo p^l takes $O(n\mathsf{M}(l\mu))$ word operations. We conclude that $T(n) \in O(n\mathsf{M}(l\mu) + \mathsf{M}(\mu)\log\mu)$.

Now let $r \ge 2$. By Lemma 10.4, the cost for step 3 is $O(\mathsf{M}(n)\log r)$ additions and multiplications in the field $\mathbb{F}_p$. Step 4 takes $O(\mathsf{M}(n)\log n)$ additions and multiplications in $\mathbb{F}_p$ plus $O(n)$ divisions, by Corollary 11.9. By Corollary 11.11, the cost for steps 3 and 4 in word operations is therefore $O(\mathsf{M}(n)\log n \cdot \mathsf{M}(\mu) + n\mathsf{M}(\mu)\log\mu)$.

By Exercise 9.7, we can reduce all coefficients of f modulo $p, p^2, p^4, \ldots, p^{2^{d-1}}$ with $O(n\mathsf{M}(l\mu))$ word operations. Theorem 15.11 implies that the cost for the jth execution of step 6 is $O(\mathsf{M}(n)\mathsf{M}(2^j\mu))$ word operations. Summing this over j, the superlinearity of M (Section 8.3) and the geometric sum $\sum_{1 \le j \le d} 2^j \le 4l$ imply that the overall cost for steps 5 and 6 is $O(\mathsf{M}(n)\mathsf{M}(l\mu))$ word operations.

also be regarded as lifting the factorization

$$x^4 - 1 \equiv (x-2)(x^3 + 2x^2 - x - 2) \bmod 5$$

to a factorization modulo 625. In the above setting, we then have $f = x^4 - 1$, $p = 5$, $g = x^3 + 2x^2 - x - 2$, and $h = x - 2$. The polynomials g and h are coprime modulo 5, and with the Extended Euclidean Algorithm in $\mathbb{F}_5[x]$, we get $s = -2$ and $t = 2x^2 - 2x - 1$ such that $sg + th \equiv 1 \bmod 5$. With $m = p$, we obtain

$$\begin{aligned} e &= f - gh = 5x^2 - 5, \\ \hat{g} &= g + te = 10x^4 - 9x^3 - 13x^2 + 9x + 3, \\ \hat{h} &= h + se = -10x^2 + x + 8, \end{aligned}$$

and in fact

$$f - \hat{g}\hat{h} = 25 \cdot (4x^6 - 4x^5 - 8x^4 + 7x^3 + 5x^2 - 3x - 1) \equiv 0 \bmod 25,$$

so that $f \equiv \hat{g}\hat{h} \bmod 25$. $\diamond$

The above example shows one drawback to our first approach: the degrees of $\hat{g}$ and $\hat{h}$ are higher than those of g and h, in particular their sum exceeds the degree of f. This may happen because the multiples of m are zero divisors modulo m^2, and hence the product of the leading coefficients of two polynomials may vanish modulo m^2.

To overcome this problem, we use division with remainder in $R[x]$. Since R is not a field, this is not always possible. The following lemma states that division with remainder by monic polynomials always works.

LEMMA 15.9. *(i) Let $f, g \in R[x]$, with g nonzero and monic. Then there exist unique polynomials $q, r \in R[x]$ with $f = qg + r$ and $\deg r < \deg g$.*

(ii) If f, g, q, r are as in (i) and $f \equiv 0 \bmod m$ for some $m \in R$, then $q \equiv r \equiv 0 \bmod m$.

Part (i) has been proven in Section 2.4, and the proof of (ii) is Exercise 15.12. We do not need the coefficients of the new polynomials exactly, but only modulo m^2. This means that over a Euclidean domain R we can reduce them accordingly and keep their sizes small. Here are the formulas that work.

ALGORITHM 15.10 Hensel step.

Input: An element m in a (commutative) ring R, and polynomials $f, g, h, s, t \in R[x]$ such that

$$f \equiv gh \bmod m \text{ and } sg + th \equiv 1 \bmod m,$$

$\mathrm{lc}(f)$ is not a zero divisor modulo m, h is monic, $\deg f = n = \deg g + \deg h$, $\deg s < \deg h$, and $\deg t < \deg g$.

Output: Polynomials $g^*, h^*, s^*, t^* \in R[x]$ such that

$$f \equiv g^* h^* \bmod m^2 \text{ and } s^* g^* + t^* h^* \equiv 1 \bmod m^2,$$

h^* is monic, $g^* \equiv g \bmod m$, $h^* \equiv h \bmod m$, $s^* \equiv s \bmod m$, and $t^* \equiv t \bmod m$, $\deg g^* = \deg g$, $\deg h^* = \deg h$, $\deg s^* < \deg h^*$, and $\deg t^* < \deg g^*$.

1. compute $e, q, r, g^*, h^* \in R[x]$ such that $\deg r < \deg h$ and

$$\begin{array}{rclcrcl} e & \equiv & f - gh \bmod m^2, & & se & \equiv & qh + r \bmod m^2, \\ g^* & \equiv & g + te + qg \bmod m^2, & & h^* & \equiv & h + r \bmod m^2 \end{array} \tag{4}$$

2. compute $b, c, d, s^*, t^* \in R[x]$ such that $\deg d < \deg h^*$ and

$$\begin{array}{rclcrcl} b & \equiv & sg^* + th^* - 1 \bmod m^2, & & sb & \equiv & ch^* + d \bmod m^2, \\ s^* & \equiv & s - d \bmod m^2, & & t^* & \equiv & t - tb - cg^* \bmod m^2 \end{array} \tag{5}$$

3. **return** g^*, h^*, s^*, t^*

EXAMPLE 15.8 (continued). We let $m = 5$ and f, g, h, s, t, e in $\mathbb{Z}[x]$ be as in Example 15.8. Division of se by h with remainder yields $q = -10x + 5$ and $r = -5$ with $se \equiv qh + r \bmod 25$. Moreover, we compute

$$g^* \equiv g + te + qg \equiv x^3 + 7x^2 - x - 7 \bmod 25, \quad h^* \equiv h + r \equiv x - 7 \bmod 25.$$

Then $f \equiv g^* h^* \bmod 25$, and the degrees of g^*, h^* are the same as those of g and h; the polynomials are simpler than $\hat{g}, \hat{h}$ as calculated before. As in Example 9.24, we obtain that 7 is a solution to $x^4 - 1 \equiv 0 \bmod 25$ that is congruent to the starting solution 2 modulo 5.

To obtain s^*, t^*, which we need for the next iteration, we compute

$$b \equiv sg^* + th^* - 1 \equiv -5x^2 - 10x - 5 \bmod 25.$$

Polynomial division yields $c = 10x - 10$ and $d = -10$ with $sb \equiv ch^* + d \bmod 25$. Now

$$s^* \equiv s - d \equiv 8 \bmod 25, \quad t^* \equiv t - tb - cg^* \equiv -8x^2 - 12x - 1 \bmod 25.$$

Then indeed $s^* g^* + t^* h^* \equiv 1 \bmod 25$, and the degrees of s^*, t^* agree with those of s, t, respectively. $\diamond$

THEOREM 15.11.

Algorithm 15.10 works correctly as specified. It uses $O(\mathsf{M}(n)\mathsf{M}(\log m))$ word operations if $R = \mathbb{Z}$, $m > 1$, and all inputs have max-norm less than m^2, and $O(\mathsf{M}(n)\mathsf{M}(\deg_y m))$ operations in the field F if $R = F[y]$ and the degree in y of all inputs is less than $2\deg_y m$.

Algoritmo de primalidad de AKS

A computational introduction to number theory and algebra

Victor Shoup

Pág 550

On input n, where n is an integer and $n > 1$, do the following:

1. if n is of the form a^b for integers $a > 1$ and $b > 1$ then
   return *false*
2. find the smallest integer $r > 1$ such that either
   $\gcd(n, r) > 1$
   or
   $\gcd(n, r) = 1$ and
   $[n]_r \in \mathbb{Z}_r^*$ has multiplicative order $> 4\,\mathrm{len}(n)^2$
3. if $r = n$ then return *true*
4. if $\gcd(n, r) > 1$ then return *false*
5. for $j \leftarrow 1$ to $2\,\mathrm{len}(n)\lfloor r^{1/2} \rfloor + 1$ do
   if $(X + j)^n \not\equiv X^n + j \pmod{X^r - 1}$ in the ring $\mathbb{Z}_n[X]$ then
   return *false*
6. return *true*

Fig. 21.1. Algorithm AKS

21.2.1 Running time analysis

The question of the running time of Algorithm AKS is settled by the following fact:

Theorem 21.2. *For integers $n > 1$ and $m \geq 1$, the least prime r such that $r \nmid n$ and the multiplicative order of $[n]_r \in \mathbb{Z}_r^*$ is greater than m is $O(m^2\,\mathrm{len}(n))$.*

Proof. Call a prime r "good" if $r \nmid n$ and the multiplicative order of $[n]_r \in \mathbb{Z}_r^*$ is greater than m, and otherwise call r "bad." If r is bad, then either $r \mid n$ or $r \mid (n^d - 1)$ for some $d = 1, \ldots, m$. Thus, any bad prime r satisfies

$$r \mid n \prod_{d=1}^{m} (n^d - 1).$$

If all primes r up to some given bound $x \geq 2$ are bad, then the product of all primes up to x divides $n \prod_{d=1}^{m} (n^d - 1)$, and so in particular,

$$\prod_{r \leq x} r \leq n \prod_{d=1}^{m} (n^d - 1),$$