21

Algorithms for finite fields

This chapter discusses efficient algorithms for factoring polynomials over finite fields, and related problems, such as testing if a given polynomial is irreducible, and generating an irreducible polynomial of given degree.

> *Throughout this chapter, F denotes a finite field of characteristic p and cardinality $q = p^w$.*

In addition to performing the usual arithmetic and comparison operations in F, we assume that our algorithms have access to the numbers p, w, and q, and have the ability to generate random elements of F. Generating such a random field element will count as one "operation in F," along with the usual arithmetic operations. Of course, the "standard" ways of representing F as either $\mathbb{Z}_p$ (if $w = 1$), or as the ring of polynomials modulo an irreducible polynomial over $\mathbb{Z}_p$ of degree w (if $w > 1$), satisfy the above requirements, and also allow for the implementation of arithmetic operations in F that take time $O(\text{len}(q)^2)$ on a RAM (using simple, quadratic-time arithmetic for polynomials and integers).

21.1 Testing and constructing irreducible polynomials

Let $f \in F[\mathtt{X}]$ be a monic polynomial of degree $\ell > 0$. We develop here an efficient algorithm that determines if f is irreducible.

The idea is a simple application of Theorem 20.9. That theorem says that for any integer $k \geq 1$, the polynomial $\mathtt{X}^{q^k} - \mathtt{X}$ is the product of all monic irreducibles whose degree divides k. Thus, $\gcd(\mathtt{X}^q - \mathtt{X}, f)$ is the product of all the distinct linear factors of f. If f has no linear factors, then $\gcd(\mathtt{X}^{q^2} - \mathtt{X}, f)$ is the product of all the distinct quadratic irreducible factors of f. And so on. Now, if f is not irreducible, it must be divisible by some irreducible polynomial of degree at most $\ell/2$, and if g is an irreducible factor of f

of minimal degree, say k, then we have $k \leq \ell/2$ and $\gcd(\mathtt{X}^{q^k} - \mathtt{X}, f) \neq 1$. Conversely, if f is irreducible, then $\gcd(\mathtt{X}^{q^k} - \mathtt{X}, f) = 1$ for all positive integers k up to $\ell/2$. So to test if f is irreducible, it suffices to check if $\gcd(\mathtt{X}^{q^k} - \mathtt{X}, f) = 1$ for all positive integers k up to $\ell/2$ — if so, we may conclude that f is irreducible, and otherwise, we may conclude that f is not irreducible. To carry out the computation efficiently, we note that if $h \equiv \mathtt{X}^{q^k} \pmod{f}$, then $\gcd(h - \mathtt{X}, f) = \gcd(\mathtt{X}^{q^k} - \mathtt{X}, f)$.

The above observations suggest the following algorithm, which takes as input a monic polynomial $f \in F[\mathtt{X}]$ of degree $\ell > 0$, and outputs *true* if f is irreducible, and *false* otherwise:

Algorithm IPT:

$h \leftarrow \mathtt{X} \bmod f$
for $k \leftarrow 1$ to $\lfloor \ell/2 \rfloor$ do
    $h \leftarrow h^q \bmod f$
    if $\gcd(h - \mathtt{X}, f) \neq 1$ then return *false*
return *true*

The correctness of Algorithm IPT follows immediately from the above discussion. As for the running time, we have:

Theorem 21.1. *Algorithm IPT uses $O(\ell^3 \operatorname{len}(q))$ operations in F.*

Proof. Consider an execution of a single iteration of the main loop. The cost of the qth-powering step (using a standard repeated-squaring algorithm) is $O(\operatorname{len}(q))$ multiplications modulo f, and so $O(\ell^2 \operatorname{len}(q))$ operations in F. The cost of the gcd computation is $O(\ell^2)$ operations in F. Thus, the cost of a single loop iteration is $O(\ell^2 \operatorname{len}(q))$ operations in F, from which it follows that the cost of the entire algorithm is $O(\ell^3 \operatorname{len}(q))$ operations in F. □

Algorithm IPT is a "polynomial time" algorithm, since the length of the binary encoding of the input is about $\ell \operatorname{len}(q)$, and so the algorithm runs in time polynomial in its input length, assuming that arithmetic operations in F take time polynomial in $\operatorname{len}(q)$. Indeed, using a standard representation for F, each operation in F takes time $O(\operatorname{len}(q)^2)$ on a RAM, and so the running time on a RAM for the above algorithm would be $O(\ell^3 \operatorname{len}(q)^3)$, that is, cubic in the bit-length of the input.

Let us now consider the related problem of constructing an irreducible polynomial of specified degree $\ell > 0$. To do this, we can simply use the result of Theorem 20.11, which has the following probabilistic interpretation: if we choose a random, monic polynomial f of degree ℓ over F, then the

probability that f is irreducible is at least $1/2\ell$. This suggests the following probabilistic algorithm:

Algorithm RIP:

repeat
    choose $a_0, \ldots, a_{\ell-1} \in F$ at random
    set $f \leftarrow \mathtt{X}^\ell + \sum_{i=0}^{\ell-1} a_i \mathtt{X}^i$
    test if f is irreducible using Algorithm IPT
until f is irreducible
output f

Theorem 21.2. *Algorithm RIP uses an expected number of $O(\ell^4 \operatorname{len}(q))$ operations in F, and its output is uniformly distributed over all monic irreducibles of degree ℓ.*

Proof. Because of Theorem 20.11, the expected number of loop iterations of the above algorithm is $O(\ell)$. Since Algorithm IPT uses $O(\ell^3 \operatorname{len}(q))$ operations in F, the statement about the running time of Algorithm RIP is immediate. The statement about its output distribution is clear. □

The expected running-time bound in Theorem 21.2 is actually a bit of an over-estimate. The reason is that if we generate a random polynomial of degree ℓ, it is likely to have a small irreducible factor, which will be discovered very quickly by Algorithm IPT. In fact, it is known (see §21.7) that the expected value of the degree of the least degree irreducible factor of a random monic polynomial of degree ℓ over F is $O(\operatorname{len}(\ell))$, from which it follows that the expected number of operations in F performed by Algorithm RIP is actually $O(\ell^3 \operatorname{len}(\ell) \operatorname{len}(q))$.

Exercise 21.1. Let $f \in F[\mathtt{X}]$ be a monic polynomial of degree $\ell > 0$. Also, let $\eta := [\mathtt{X}]_f \in E$, where E is the F-algebra $E := F[\mathtt{X}]/(f)$.

(a) Show how to compute—given as input $\alpha \in E$ and $\eta^{q^m} \in E$ (for some integer $m > 0$)—the value $\alpha^{q^m} \in E$, using just $O(\ell^{2.5})$ operations in F, and space for $O(\ell^{1.5})$ elements of F. Hint: see Theorems 17.1 and 20.7, as well as Exercise 18.4.

(b) Show how to compute—given as input $\eta^{q^m} \in E$ and $\eta^{q^{m'}} \in E$, where m and m' are positive integers—the value $\eta^{q^{m+m'}} \in E$, using $O(\ell^{2.5})$ operations in F, and space for $O(\ell^{1.5})$ elements of F.

(c) Show how to compute—given as input $\eta^q \in E$ and a positive integer m—the value $\eta^{q^m} \in E$, using $O(\ell^{2.5} \operatorname{len}(m))$ operations in F, and

space for $O(\ell^{1.5})$ elements of F. Hint: use a repeated-squaring-like algorithm.

Exercise 21.2. This exercise develops an alternative irreducibility test.

(a) Show that a monic polynomial $f \in F[\mathtt{X}]$ of degree $\ell > 0$ is irreducible if and only if $\mathtt{X}^{q^\ell} \equiv \mathtt{X} \pmod{f}$ and $\gcd(\mathtt{X}^{q^{\ell/s}} - \mathtt{X}, f) = 1$ for all primes $s \mid \ell$.

(b) Using part (a) and the result of the previous exercise, show how to determine if f is irreducible using $O(\ell^{2.5}\operatorname{len}(\ell)\omega(\ell) + \ell^2 \operatorname{len}(q))$ operations in F, where $\omega(\ell)$ is the number of distinct prime factors of ℓ.

(c) Show that the operation count in part (b) can be reduced to $O(\ell^{2.5}\operatorname{len}(\ell)\operatorname{len}(\omega(\ell)) + \ell^2 \operatorname{len}(q))$. Hint: see Exercise 3.30.

Exercise 21.3. Design and analyze a *deterministic* algorithm that takes as input a list of irreducible polynomials $f_1, \ldots, f_r \in F[\mathtt{X}]$, where $\ell_i := \deg(f_i)$ for $i = 1, \ldots, r$. Assuming that the degrees $\ell_1, \ldots, \ell_r$ are pairwise relatively prime, your algorithm should output an irreducible polynomial $f \in F[\mathtt{X}]$ of degree $\ell := \prod_{i=1}^r \ell_i$ using $O(\ell^3)$ operations in F.

Exercise 21.4. Design and analyze a probabilistic algorithm that, given a monic irreducible polynomial $f \in F[\mathtt{X}]$ of degree ℓ as input, generates as output a random monic irreducible polynomial $g \in F[\mathtt{X}]$ of degree ℓ (i.e., g should be uniformly distributed over all such polynomials), using an expected number of $O(\ell^{2.5})$ operations in F. Hint: use Exercise 19.8 (or alternatively, Exercise 19.9).

Exercise 21.5. Let $f \in F[\mathtt{X}]$ be a monic irreducible polynomial of degree ℓ, let $E := F[\mathtt{X}]/(f)$, and let $\eta := [\mathtt{X}]_f \in E$. Design and analyze a deterministic algorithm that takes as input the polynomial f defining the extension E, and outputs the values

$$s_j := \mathbf{Tr}_{E/F}(\eta^j) \in F \quad (j = 0, \ldots, \ell - 1),$$

using $O(\ell^2)$ operations in F. Here, $\mathbf{Tr}_{E/F}$ is the trace from E to F (see §20.4). Show that given an arbitrary $\alpha \in E$, along with the values $s_0, \ldots, s_{\ell-1}$, one can compute $\mathbf{Tr}_{E/F}(\alpha)$ using just $O(\ell)$ operations in F.

21.2 Computing minimal polynomials in $F[\mathtt{X}]/(f)$ (III)

We consider, for the third and final time, the problem considered in §18.2 and §19.5: $f \in F[\mathtt{X}]$ is a monic polynomial of degree $\ell > 0$, and $E :=$

$F[\mathtt{X}]/(f) = F[\eta]$, where $\eta := [\mathtt{X}]_f$; we are given an element $\alpha \in E$, and want to compute the minimal polynomial $\phi \in F[\mathtt{X}]$ of α over F. We develop an alternative algorithm, based on the theory of finite fields. Unlike the algorithms in §18.2 and §19.5, this algorithm only works when F is finite and the polynomial f is irreducible, so that E is also a finite field.

From Theorem 20.15, we know that the degree of α over F is the smallest positive integer k such that $\alpha^{q^k} = \alpha$. By successive qth powering, we can compute the conjugates of α, and determine the degree k, using $O(k \operatorname{len}(q))$ operations in E, and hence $O(k\ell^2 \operatorname{len}(q))$ operations in F.

Now, we could simply compute the minimal polynomial ϕ by directly using the formula

$$\phi(\mathtt{Y}) = \prod_{i=0}^{k-1} (\mathtt{Y} - \alpha^{q^i}). \tag{21.1}$$

This would involve computations with polynomials in the variable $\mathtt{Y}$ whose coefficients lie in the extension field E, although at the end of the computation, we would end up with a polynomial all of whose coefficients lie in F. The cost of this approach would be $O(k^2)$ operations in E, and hence $O(k^2\ell^2)$ operations in F.

A more efficient approach is the following. Substituting η for $\mathtt{Y}$ in the identity (21.1), we have

$$\phi(\eta) = \prod_{i=0}^{k-1} (\eta - \alpha^{q^i}).$$

Using this formula, we can compute (given the conjugates of α) the value $\phi(\eta) \in E$ using $O(k)$ operations in E, and hence $O(k\ell^2)$ operations in F. Now, $\phi(\eta)$ is an element of E, and for computational purposes, it is represented as $[g]_f$ for some polynomial $g \in F[\mathtt{X}]$ of degree less than ℓ. Moreover, $\phi(\eta) = [\phi]_f$, and hence $\phi \equiv g \pmod{f}$. In particular, if $k < \ell$, then $g = \phi$; otherwise, if $k = \ell$, then $g = \phi - f$. In either case, we can recover ϕ from g with an additional $O(\ell)$ operations in F.

Thus, given the conjugates of α, we can compute ϕ using $O(k\ell^2)$ operations in F. Adding in the cost of computing the conjugates, this gives rise to an algorithm that computes the minimal polynomial of α using $O(k\ell^2 \operatorname{len}(q))$ operations in F.

In the worst case, then, this algorithm uses $O(\ell^3 \operatorname{len}(q))$ operations in F. A reasonably careful implementation needs space for storing a constant number of elements of E, and hence $O(\ell)$ elements of F. For very small values of q, the efficiency of this algorithm will be comparable to that of

the algorithm in §19.5, but for large q, it will be much less efficient. Thus, this approach does not really yield a better algorithm, but it does serve to illustrate some of the ideas of the theory of finite fields.

21.3 Factoring polynomials: the Cantor–Zassenhaus algorithm

In the remaining sections of this chapter, we develop efficient algorithms for factoring polynomials over the finite field F.

The algorithm we discuss in this section is due to Cantor and Zassenhaus. It has two stages:

Distinct Degree Factorization: The input polynomial is decomposed into factors so that each factor is a product of distinct irreducibles of the same degree (and the degree of those irreducibles is also determined).

Equal Degree Factorization: Each of the factors produced in the distinct degree factorization stage are further factored into their irreducible factors.

The algorithm we present for distinct degree factorization is a deterministic, polynomial-time algorithm. The algorithm we present for equal degree factorization is a *probabilistic* algorithm that runs in expected polynomial time (and whose output is always correct).

21.3.1 Distinct degree factorization

The problem, more precisely stated, is this: given a monic polynomial $f \in F[\mathtt{X}]$ of degree $\ell > 0$, produce a list of polynomial/integer pairs (g, k), where

- each g is a product of distinct monic irreducible polynomials of degree k, and
- the product of all the polynomials g in the list is equal to f.

This problem can be easily solved using Theorem 20.9, using a simple variation of the algorithm we discussed in §21.1 for irreducibility testing. The basic idea is this. We can compute $g := \gcd(\mathtt{X}^q - \mathtt{X}, f)$, so that g is the product of all the distinct linear factors of f. We can remove the factor g from f, but after doing so, f may still contain some linear factors (if the original polynomial was not square-free), and so we have to repeat the above step until no linear factors are discovered. Having removed all linear factors from f, we next compute $\gcd(\mathtt{X}^{q^2} - \mathtt{X}, f)$, which will be the product of all the distinct quadratic irreducibles dividing f, and we can remove these from f—although $\mathtt{X}^{q^2} - \mathtt{X}$ is the product of all linear and quadratic irreducibles,

since we have already removed the linear factors from f, the gcd will give us just the quadratic factors of f. As above, we may have to repeat this a few times to remove all the quadratic factors from f. In general, for $k = 1, \ldots, \ell$, having removed all the irreducible factors of degree less than k from f, we compute $\gcd(\mathtt{X}^{q^k} - \mathtt{X}, f)$ to obtain the product of all the distinct irreducible factors of f of degree k, repeating as necessary to remove all such factors.

The above discussion leads to the following algorithm for distinct degree factorization, which takes as input a monic polynomial $f \in F[\mathtt{X}]$ of degree $\ell > 0$:

Algorithm DDF:

$h \leftarrow \mathtt{X} \bmod f$
$k \leftarrow 1$
while $f \neq 1$ do
    $h \leftarrow h^q \bmod f$
    $g \leftarrow \gcd(h - \mathtt{X}, f)$
    while $g \neq 1$ do
        output (g, k)
        $f \leftarrow f/g$
        $h \leftarrow h \bmod f$
        $g \leftarrow \gcd(h - \mathtt{X}, f)$
    $k \leftarrow k + 1$

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

Theorem 21.3. *Algorithm DDF uses $O(\ell^3 \operatorname{len}(q))$ operations in F.*

Proof. Note that the body of the outer 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 computations. We perform one gcd computation in every iteration of the main loop, for a total of ℓ such computations. We also perform an "extra" gcd computation whenever we discover a non-trivial factor of f; however, since we only discover at most ℓ such non-trivial factors, we perform at most ℓ such "extra" gcd computations. So the total number of gcd computations is at most 2ℓ, and as each of these takes $O(\ell^2)$ operations in F, they contribute a term of $O(\ell^3)$ to the total operation count. This

term is dominated by the cost of the qth-powering steps (as is the cost of the division step in the inner loop), and so the total cost of Algorithm DDF is $O(\ell^3 \operatorname{len}(q))$ operations in F. $\square$

21.3.2 Equal degree factorization

The problem, more precisely stated, is this: given a monic polynomial $g \in F[\mathtt{X}]$ of degree $\ell > 0$, and an integer $k > 0$, such that g is of the form

$$g = g_1 \cdots g_r$$

for distinct monic irreducible polynomials $g_1, \ldots, g_r$, each of degree k, compute these irreducible factors of g. Note that given g and k, the value of r is easily determined, since $r = \ell/k$.

We begin by discussing the basic mathematical ideas that will allow us to efficiently split g into two non-trivial factors, and then we present a somewhat more elaborate algorithm that completely factors g.

By the Chinese remainder theorem, we have an F-algebra isomorphism

$$\theta : E_1 \times \cdots \times E_r \to E,$$

where for $i = 1, \ldots, r$, E_i is the extension field $F[\mathtt{X}]/(g_i)$ of degree k over F, and E is the F-algebra $F[\mathtt{X}]/(g)$.

Recall that $q = p^w$. We have to treat the cases $p = 2$ and $p > 2$ separately. We first treat the case $p = 2$. Let us define the polynomial

$$M_k := \sum_{j=0}^{wk-1} \mathtt{X}^{2^j} \in F[\mathtt{X}]. \tag{21.2}$$

(The algorithm in the case $p > 2$ will only differ in the definition of M_k.)

For $\alpha \in E$, if $\alpha = \theta(\alpha_1, \ldots, \alpha_r)$, then we have

$$M_k(\alpha) = \theta(M_k(\alpha_1), \ldots, M_k(\alpha_r)).$$

Note that each E_i is an extension of $\mathbb{Z}_2$ of degree wk, and that

$$M_k(\alpha_i) = \sum_{j=0}^{wk-1} \alpha_i^{2^j} = \mathbf{Tr}_{E_i/\mathbb{Z}_2}(\alpha_i),$$

where $\mathbf{Tr}_{E_i/\mathbb{Z}_2} : E_i \to \mathbb{Z}_2$ is the trace from E_i to $\mathbb{Z}_2$, which is a surjective, $\mathbb{Z}_2$-linear map (see §20.4).

Now, suppose we choose $\alpha \in E$ at random. Then if $\alpha = \theta(\alpha_1, \ldots, \alpha_r)$, the α_i will be independently distributed, with each α_i uniformly distributed

over E_i. It follows that the values $M_k(\alpha_i)$ will be independently and uniformly distributed over $\mathbb{Z}_2$. Thus, if $a := \mathrm{rep}(M_k(\alpha))$ (i.e., $a \in F[\mathtt{X}]$ is the polynomial of degree less than ℓ such that $M_k(\alpha) = [a]_g$), then $\gcd(a, g)$ will be the product of those factors g_i of g such that $M_k(\alpha_i) = 0$. We will fail to get a non-trivial factorization only if the $M_k(\alpha_i)$ are either all 0 or all 1, which for $r \geq 2$ happens with probability at most $1/2$ (the worst case being when $r = 2$).

That is our basic splitting strategy. The algorithm for completely factoring g works as follows. The algorithm proceeds in stages. At any stage, we have a partial factorization $g = \prod_{h \in H} h$, where H is a set of non-constant, monic polynomials. Initially, $H = \{g\}$. With each stage, we attempt to get a finer factorization of g by trying to split each $h \in H$ using the above splitting strategy—if we succeed in splitting h into two non-trivial factors, then we replace h by these two factors. We continue in this way until $|H| = r$.

Here is the full equal degree factorization algorithm. It takes as input a monic polynomial $g \in F[\mathtt{X}]$ of degree $\ell > 0$, and an integer $k > 0$, such that g is the product of $r := \ell/k$ distinct monic irreducible polynomials, each of degree k. With M_k as defined in (21.2), the algorithm runs as follows:

Algorithm EDF:

$H \leftarrow \{g\}$
while $|H| < r$ do
    $H' \leftarrow \emptyset$
    for each $h \in H$ do
        choose $\alpha \in F[\mathtt{X}]/(h)$ at random
        $d \leftarrow \gcd(\mathrm{rep}(M_k(\alpha)), h)$
        if $d = 1$ or $d = h$
            then $H' \leftarrow H' \cup \{h\}$
            else $H' \leftarrow H' \cup \{d, h/d\}$
    $H \leftarrow H'$
output H

The correctness of the algorithm is clear from the above discussion. As for its expected running time, we can get a quick-and-dirty upper bound as follows:

- For a given h, the cost of computing $M_k(\alpha)$ for $\alpha \in F[\mathtt{X}]/(h)$ is $O(k \deg(h)^2 \operatorname{len}(q))$ operations in F, and so the number of operations in F performed in each iteration of the main loop is at most a constant

times

$$k \operatorname{len}(q) \sum_{h \in H} \deg(h)^2 \leq k \operatorname{len}(q) \Bigg(\sum_{h \in H} \deg(h) \Bigg)^2 = k\ell^2 \operatorname{len}(q).$$

- The expected number of iterations of the main loop until we get some non-trivial split is $O(1)$.
- The algorithm finishes after getting $r - 1$ non-trivial splits.
- Therefore, the total expected cost is $O(rk\ell^2 \operatorname{len}(q))$, or $O(\ell^3 \operatorname{len}(q))$, operations in F.

This analysis gives a bit of an over-estimate—it does not take into account the fact that we expect to get fairly "balanced" splits. For the purposes of analyzing the overall running time of the Cantor–Zassenhaus algorithm, this bound suffices; however, the following analysis gives a tight bound on the complexity of Algorithm EDF.

Theorem 21.4. *In the case $p = 2$, Algorithm EDF uses an expected number of $O(k\ell^2 \operatorname{len}(q))$ operations in F.*

Proof. We may assume $r \geq 2$. Let L be a random variable that denotes the number of iterations of the main loop of the algorithm.

We claim that $E[L] = O(\operatorname{len}(r))$. To prove this claim, we make use of the fact (see Theorem 6.25) that

$$\mathsf{E}[L] = \sum_{t \geq 1} \mathsf{P}[L \geq t].$$

For $i = 1, \ldots, r$ and $j = i+1, \ldots, r$, define L_{ij} to be the number of iterations of the main loop in which the factors g_i and g_j remain unseparated at the beginning of the loop. Now, if g_i and g_j have not been separated at the beginning of one loop iteration, then they will be separated at the beginning of the next with probability $1/2$. It follows that

$$\mathsf{P}[L_{ij} \geq t] \leq 2^{-(t-1)}.$$

Also note that $L \geq t$ implies that $L_{ij} \geq t$ for some i, j, and hence

$$\mathsf{P}[L \geq t] \leq \sum_{i=1}^{r} \sum_{j=i+1}^{r} \mathsf{P}[L_{ij} \geq t] \leq r^2 2^{-t}.$$

So we have

$$\begin{aligned}\mathsf{E}[L] &= \sum_{t\geq 1} \mathsf{P}[L \geq t] \\ &= \sum_{t\leq 2\log_2 r} \mathsf{P}[L \geq t] + \sum_{t>2\log_2 r} \mathsf{P}[L \geq t] \\ &\leq 2\log_2 r + \sum_{t>2\log_2 r} r^2 2^{-t} \\ &\leq 2\log_2 r + \sum_{t\geq 0} 2^{-t} \\ &= 2\log_2 r + 2.\end{aligned}$$

That proves the claim.

As discussed in the paragraph above this theorem, the cost of each iteration of the main loop is $O(k\ell^2 \operatorname{len}(q))$ operations in F. Combining this with the fact that $E[L] = O(\operatorname{len}(r))$, it follows that the expected number of operations in F for the entire algorithm is $O(\operatorname{len}(r)k\ell^2 \operatorname{len}(q))$. This is significantly better than the above quick-and-dirty estimate, but is not quite the result we are after—we have to get rid of the factor $\operatorname{len}(r)$. There are a number of ways to do this. We sketch one such way, which is a bit ad hoc, but sufficient for our purposes.

Let us define

$$S := \sum_{i=1}^{r} \sum_{j=i+1}^{r} L_{ij}.$$

We claim that the total work performed by the algorithm in attempting to split non-irreducible factors of g is

$$O(Sk^3 \operatorname{len}(q)).$$

To see why this is so, consider one iteration of the inner loop of the algorithm, where we are trying to split a factor h of g, where h is the product of two or more irreducible factors of g. Let us write $h = g_{i_1} \cdots g_{i_n}$, where $2 \leq n \leq r$. On the one hand, the number of operations in F performed in this step is at most $ck \deg(h)^2 \operatorname{len}(q)$ for some constant c, which we may write as $cn^2 \cdot k^3 \operatorname{len}(q)$. On the other hand, each pair of indices $(i_j, i_{j'})$, with $1 \leq j < j' \leq n$, contributes 1 to the sum defining S, for a total contribution from pairs at this step of $n(n-1)/2 \geq n^2/4$. The claim now follows.

Algorithm EDF is a little silly in that it wastes time trying to split irreducible factors (and although it would be trivial to modify the algorithm to avoid this, the asymptotic running time would not be affected significantly).

It is easy to see that attempting to split a single irreducible factor takes $O(k^3 \operatorname{len}(q))$ operations in F, and hence the total amount of work wasted in this way is $O(Lrk^3 \operatorname{len}(q))$.

We next claim that $\mathsf{E}[L_{ij}] = O(1)$, for all i, j. Indeed,

$$\mathsf{E}[L_{ij}] = \sum_{t \geq 1} \mathsf{P}[L_{ij} \geq t] \leq \sum_{t \geq 1} 2^{-(t-1)} = 2.$$

It follows that

$$\mathsf{E}[S] = \sum_{ij} \mathsf{E}[L_{ij}] = O(r^2).$$

Therefore, the expected number of operations in F performed by the algorithm is at most a constant times

$$\mathsf{E}[S]k^3 \operatorname{len}(q) + \mathsf{E}[L]rk^3 \operatorname{len}(q) = O(r^2 k^3 \operatorname{len}(q) + r \operatorname{len}(r) k^3 \operatorname{len}(q)),$$

which is $O(k\ell^2 \operatorname{len}(q))$. □

That completes the discussion of Algorithm EDF in the case $p = 2$.

The case $p > 2$

Now assume that $p > 2$, so that p, and hence also q, is odd. Algorithm EDF in this case is exactly the same as above, except that in this case, we define the polynomial M_k as

$$M_k := \mathtt{X}^{(q^k-1)/2} - 1 \in F[\mathtt{X}]. \tag{21.3}$$

Just as before, for $\alpha \in E$ with $\alpha = \theta(\alpha_1, \ldots, \alpha_r)$, we have

$$M_k(\alpha) = \theta(M_k(\alpha_1), \ldots, M_k(\alpha_r)).$$

Note that each group E_i^* is a cyclic group of order $q^k - 1$, and therefore, the image of the $(q^k - 1)/2$-power map on E_i^* is $\{\pm 1\}$.

Now, suppose we choose $\alpha \in E$ at random. Then if $\alpha = \theta(\alpha_1, \ldots, \alpha_r)$, the α_i will be independently distributed, with each α_i uniformly distributed over E_i. It follows that the values $M_k(\alpha_i)$ will be independently distributed. If $\alpha_i = 0$, which happens with probability $1/q^k$, then $M_k(\alpha_i) = -1$; otherwise, $\alpha_i^{(q^k-1)/2}$ is uniformly distributed over $\{\pm 1\}$, and so $M_k(\alpha_i)$ is uniformly distributed over $\{0, -2\}$. That is to say,

$$M_k(\alpha_i) = \begin{cases} 0 & \text{with probability } (q^k - 1)/2q^k, \\ -1 & \text{with probability } 1/q^k, \\ -2 & \text{with probability } (q^k - 1)/2q^k. \end{cases}$$

Thus, if $a := \operatorname{rep}(M_k(\alpha))$, then $\gcd(a, g)$ will be the product of those factors

g_i of g such that $M_k(\alpha_i) = 0$. We will fail to get a non-trivial factorization only if the $M_k(\alpha_i)$ are either all zero or all non-zero. Assume $r \geq 2$. Consider the worst case, namely, when $r = 2$. In this case, a simple calculation shows that the probability that we fail to split these two factors is

$$\left(\frac{q^k - 1}{2q^k}\right)^2 + \left(\frac{q^k + 1}{2q^k}\right)^2 = \frac{1}{2}(1 + 1/q^{2k}).$$

The (very) worst case is when $q^k = 3$, in which case the probability of failure is at most $5/9$.

The same quick-and-dirty analysis given just above Theorem 21.4 applies here as well, but just as before, we can do better:

Theorem 21.5. *In the case $p > 2$, Algorithm EDF uses an expected number of $O(k\ell^2 \operatorname{len}(q))$ operations in F.*

Proof. The analysis is essentially the same as in the case $p = 2$, except that now the probability that we fail to split a given pair of irreducible factors is at most $5/9$, rather than equal to $1/2$. The details are left as an exercise for the reader. □

21.3.3 Analysis of the whole algorithm

Given an arbitrary polynomial $f \in F[\mathtt{X}]$ of degree $\ell > 0$, the distinct degree factorization step takes $O(\ell^3 \operatorname{len}(q))$ operations in F. This step produces a number of polynomials that must be further subjected to equal degree factorization. If there are s such polynomials, where the ith polynomial has degree ℓ_i, for $i = 1, \ldots, s$, then $\sum_{i=1}^{s} \ell_i = \ell$. Now, the equal degree factorization step for the ith polynomial takes an expected number of $O(\ell_i^3 \operatorname{len}(q))$ operations in F (actually, our initial, "quick and dirty" estimate is good enough here), and so it follows that the total expected cost of all the equal degree factorization steps is $O(\sum_i \ell_i^3 \operatorname{len}(q))$, which is $O(\ell^3 \operatorname{len}(q))$, operations in F. Putting this all together, we conclude:

Theorem 21.6. *The Cantor–Zassenhaus factoring algorithm uses an expected number of $O(\ell^3 \operatorname{len}(q))$ operations in F.*

This bound is tight, since in the worst case, when the input is irreducible, the algorithm really does do this much work.

Exercise 21.6. Show how to modify Algorithm DDF so that the main loop halts as soon as $2k > \deg(f)$.

Exercise 21.7. This exercise extends the techniques developed in Exercise 21.1. Let $f \in F[\mathtt{X}]$ be a monic polynomial of degree $\ell > 0$, and let $\eta := [\mathtt{X}]_f \in E$, where $E := F[\mathtt{X}]/(f)$. For integer $m > 0$, define polynomials

$$T_m := \mathtt{X} + \mathtt{X}^q + \cdots + \mathtt{X}^{q^{m-1}} \in F[\mathtt{X}] \quad \text{and} \quad N_m := \mathtt{X} \cdot \mathtt{X}^q \cdot \cdots \cdot \mathtt{X}^{q^{m-1}} \in F[\mathtt{X}].$$

(a) Show how to compute—given as input $\eta^{q^m} \in E$ and $\eta^{q^{m'}}$, where m and m' are positive integers, along with $T_m(\alpha)$ and $T_{m'}(\alpha)$, for some $\alpha \in E$—the values $\eta^{q^{m+m'}}$ and $T_{m+m'}(\alpha)$, using $O(\ell^{2.5})$ operations in F, and space for $O(\ell^{1.5})$ elements of F.

(b) Using part (a), show how to compute—given as input $\eta^q \in E$, $\alpha \in E$, and a positive integer m—the value $T_m(\alpha)$, using $O(\ell^{2.5} \operatorname{len}(m))$ operations in F, and space for $O(\ell^{1.5})$ elements of F.

(c) Repeat parts (a) and (b), except with "N" in place of "T."

Exercise 21.8. Using the result of the previous exercise, show how to implement Algorithm EDF so that it uses an expected number of

$$O(\operatorname{len}(k)\ell^{2.5} + \ell^2 \operatorname{len}(q))$$

operations in F, and space for $O(\ell^{1.5})$ elements of F.

Exercise 21.9. This exercise depends on the concepts and results in §19.6. Let E be an extension field of degree ℓ over F, specified by an irreducible polynomial of degree ℓ over F. Design and analyze an efficient probabilistic algorithm that finds a normal basis for E over F (see Exercise 20.14). Hint: there are a number of approaches to solving this problem; one way is to start by factoring $\mathtt{X}^\ell - 1$ over F, and then turn the construction in Theorem 19.7 into an efficient probabilistic procedure; if you mimic Exercise 11.2, your entire algorithm should use $O(\ell^3 \operatorname{len}(\ell) \operatorname{len}(q))$ operations in F (or $O(\operatorname{len}(r)\ell^3 \operatorname{len}(q))$ operations, where r is the number of distinct irreducible factors of $\mathtt{X}^\ell - 1$ over F).

21.4 Factoring polynomials: Berlekamp's algorithm

We now develop an alternative algorithm, due to Berlekamp, for factoring a polynomial over the finite field F.

This algorithm usually starts with a pre-processing phase to reduce the problem to that of factoring square-free polynomials. There are a number of ways to carry out this step. We present a simple-minded method here that is sufficient for our purposes.

21.4.1 A simple square-free decomposition algorithm

Let $f \in F[\mathtt{X}]$ be a monic polynomial of degree $\ell > 0$. Suppose that f is not square-free. According to Theorem 20.4, $d := \gcd(f, \mathbf{D}(f)) \neq 1$, and so we might hope to get a non-trivial factorization of f by computing d; however, we have to consider the possibility that $d = f$. Can this happen? The answer is "yes," but if it does happen that $d = f$, we can still get a non-trivial factorization of f by other means:

Theorem 21.7. *Suppose that $f \in F[\mathtt{X}]$ is a polynomial of degree $\ell > 0$, and that $\gcd(f, \mathbf{D}(f)) = f$. Then $f = g(\mathtt{X}^p)$ for some $g \in F[\mathtt{X}]$. Moreover, if $g = \sum_i b_i \mathtt{X}^i$, then $f = h^p$, where $h = \sum_i b_i^{p^{(w-1)}} \mathtt{X}^i$.*

Proof. Since $\deg(\mathbf{D}(f)) < \deg(f)$, if $\gcd(f, \mathbf{D}(f)) = f$, then we must have $\mathbf{D}(f) = 0$. If $f = \sum_{i=0}^{\ell} a_i \mathtt{X}^i$, then $\mathbf{D}(f) = \sum_{i=1}^{\ell} i a_i \mathtt{X}^{i-1}$. Since this derivative must be zero, it follows that all the coefficients a_i with $i \not\equiv 0 \pmod{p}$ must be zero to begin with. That proves that $f = g(\mathtt{X}^p)$ for some $g \in F[\mathtt{X}]$. Furthermore, if h is defined as above, then

$$h^p = \left(\sum_i b_i^{p^{(w-1)}} \mathtt{X}^i \right)^p = \sum_i b_i^{p^w} \mathtt{X}^{ip} = \sum_i b_i (\mathtt{X}^p)^i = g(\mathtt{X}^p) = f. \quad \square$$

This suggests the following recursive algorithm. The input is the polynomial f as above, and a parameter s, which is set to 1 on the initial invocation. The output is a list of pairs (g_i, s_i) such that each g_i is a square-free, non-constant polynomial over F and $f = \prod_i g_i^{s_i}$.

Algorithm SFD:

$d \leftarrow \gcd(f, \mathbf{D}(f))$
if $d = 1$ then
    output (f, s)
else if $d \neq f$ then
    recursively process (d, s) and $(f/d, s)$
else
    let $f = \mathtt{X}^\ell + \sum_{i=0}^{\ell-1} a_i \mathtt{X}^i$ // *note that $a_i = 0$ except when $p \mid i$*
    set $h \leftarrow \mathtt{X}^{\ell/p} + \sum_{i=0}^{\ell/p-1} (a_{pi})^{p^{w-1}} \mathtt{X}^i$ // *note that $h = f^{1/p}$*
    recursively process (h, ps)

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

Theorem 21.8. *Algorithm SFD uses $O(\ell^3 + \ell(w-1)\operatorname{len}(p)/p)$ operations in F.*

Proof. For input polynomial f with $\deg(f) > 0$, let $R(f)$ denote the number of recursive invocations of the algorithm, and let $P(f)$ denote the number of p^{w-1}th powers in F computed by the algorithm. It is easy to see that the number of operations in F performed by the algorithm is

$$O(R(f)\deg(f)^2 + P(f)(w-1)\operatorname{len}(p)).$$

The theorem will therefore follow from the following two inequalities:

$$R(f) \leq 2\deg(f) - 1 \tag{21.4}$$

and

$$P(f) \leq 2\deg(f)/p. \tag{21.5}$$

We prove (21.4) by induction of $\deg(f)$. We assume (21.4) holds for all input polynomials of degree less than that of f, and prove that it holds for f. Let $d := \gcd(f, \mathbf{D}(f))$. If $d = 1$, then $R(f) = 1 \leq 2\deg(f) - 1$. If $d \neq 1$ and $d \neq f$, then applying the induction hypothesis, we have

$$\begin{aligned} R(f) &= 1 + R(d) + R(f/d) \leq 1 + (2\deg(d) - 1) + (2\deg(f/d) - 1) \\ &= 2\deg(f) - 1. \end{aligned}$$

Finally, if $d = f$, then again applying the induction hypothesis, we have

$$R(f) = 1 + R(f^{1/p}) \leq 1 + (2\deg(f)/p - 1) \leq \deg(f) \leq 2\deg(f) - 1.$$

The inequality (21.5) is proved similarly by induction. We assume (21.5) holds for all input polynomials of degree less than that of f, and prove that it holds for f. Let $d := \gcd(f, \mathbf{D}(f))$. If $d = 1$, then $P(f) = 0 \leq 2\deg(f)/p$. If $d \neq 1$ and $d \neq f$, then applying the induction hypothesis, we have

$$P(f) = P(d) + P(f/d) \leq 2\deg(d)/p + 2\deg(f/d)/p = 2\deg(f)/p.$$

Finally, if $d = f$, then again applying the induction hypothesis, we have

$$P(f) = \deg(f)/p + P(f^{1/p}) \leq \deg(f)/p + 2\deg(f)/p^2 \leq 2\deg(f)/p. \quad \square$$

The running-time bound in Theorem 21.8 is essentially tight (see Exercise 21.10 below). Although it suffices for our immediate purpose as a preprocessing step in Berlekamp's factoring algorithm, Algorithm SFD is by no means the most efficient algorithm possible for square-free decomposition of polynomials. We return to this issue below, in §21.6.

21.4.2 The main factoring algorithm

Let us now assume we have a monic square-free polynomial f of degree $\ell > 0$ that we want to factor into irreducibles, such as is output by the square-free decomposition algorithm above. We first present the mathematical ideas underpinning the algorithm.

Let E be the F-algebra $F[\mathtt{X}]/(f)$. We define a subset B of E as follows:

$$B := \{\alpha \in E : \alpha^q = \alpha\}.$$

It is easy to see that B is a subalgebra of E. Indeed, for $\alpha, \beta \in B$, we have $(\alpha+\beta)^q = \alpha^q + \beta^q = \alpha + \beta$, and similarly, $(\alpha\beta)^q = \alpha^q\beta^q = \alpha\beta$. Furthermore, one sees that $c^q = c$ for all $c \in F$, and hence B is a subalgebra.

The subalgebra B is called the **Berlekamp subalgebra of** E. Let us take a closer look at it. Suppose that f factors into irreducibles as

$$f = f_1 \cdots f_r,$$

and let

$$\theta : E_1 \times \cdots \times E_r \to E$$

be the F-algebra isomorphism from the Chinese remainder theorem, where $E_i := F[\mathtt{X}]/(f_i)$ is an extension field of F of finite degree for $i = 1, \ldots, r$. Now, for $\alpha = \theta(\alpha_1, \ldots, \alpha_r) \in E$, we have $\alpha^q = \alpha$ if and only if $\alpha_i^q = \alpha_i$ for $i = 1, \ldots, r$; moreover, by Theorem 20.8, we know that for any $\alpha_i \in E_i$, we have $\alpha_i^q = \alpha_i$ if and only if $\alpha_i \in F$. Thus, we may characterize B as follows:

$$B = \{\theta(c_1, \ldots, c_r) : c_1, \ldots, c_r \in F\}.$$

Since B is a subalgebra of E, then as F-vector spaces, B is a subspace of E. Of course, E has dimension ℓ over F, with the natural basis $1, \eta, \ldots, \eta^{\ell-1}$, where $\eta := [\mathtt{X}]_f$. As for the Berlekamp subalgebra, from the above characterization of B, it is evident that

$$\theta(1, 0, \ldots, 0),\ \theta(0, 1, 0, \ldots, 0),\ \ldots,\ \theta(0, \ldots, 0, 1)$$

is a basis for B over F, and hence, B has dimension r over F.

Now we come to the actual factoring algorithm.

Stage 1: Construct a basis for B

The first stage of Berlekamp's factoring algorithm constructs a basis for B over F. We can easily do this using Gaussian elimination, as follows. Let $\rho : E \to E$ be the map that sends $\alpha \in E$ to $\alpha^q - \alpha$. Since the qth power map on E is an F-algebra homomorphism (see Theorem 20.7)—and in particular, an F-linear map—the map ρ is also F-linear. Moreover, the kernel of ρ is

none other than the Berlekamp subalgebra B. So to find a basis for B, we simply need to find a basis for the kernel of ρ using Gaussian elimination over F, as in §15.4.

To perform the Gaussian elimination, we need to choose an ordered basis for E over F, and construct a matrix $Q \in F^{\ell \times \ell}$ that represents ρ with respect to that ordered basis as in §15.2, so that evaluation of ρ corresponds to multiplying a row vector on the right by Q. We are free to choose an ordered basis in any convenient way, and the most convenient ordered basis, of course, is $(1, \eta, \ldots, \eta^{\ell-1})$, as this directly corresponds to the way we represent elements of E for computational purposes. Let us define the F-vector space isomorphism

$$\begin{aligned} \epsilon : \quad & F^{1\times\ell} \to E \\ & (a_0, \ldots, a_{\ell-1}) \mapsto a_0 + a_1\eta + \cdots + a_{\ell-1}\eta^{\ell-1}. \end{aligned} \tag{21.6}$$

The maps ϵ and ϵ^{-1} are best thought of as "type conversion operators" that require no actual computation to evaluate. The matrix Q, then, is the $\ell \times \ell$ matrix whose ith row, for $i = 1, \ldots, \ell$, is $\epsilon^{-1}(\rho(\eta^{i-1}))$. Note that if $\alpha := \eta^q$, then $\rho(\eta^{i-1}) = (\eta^{i-1})^q - \eta^{i-1} = (\eta^q)^{i-1} - \eta^{i-1} = \alpha^{i-1} - \eta^{i-1}$. This observation allows us to construct the rows of Q by first computing α as η^q via repeated squaring, and then just computing successive powers of α.

After we construct the matrix Q, we apply Gaussian elimination to get row vectors $v_1, \ldots, v_r$ that form a basis for the row null space of Q. It is at this point that our algorithm actually discovers the number r of irreducible factors of f. We can then set $\beta_i := \epsilon(v_i)$ for $i = 1, \ldots, r$ to get our basis for B.

Putting this altogether, we have the following algorithm to compute a basis for the Berlekamp subalgebra. It takes as input a monic square-free polynomial f of degree $\ell > 0$. With $E := F[\mathtt{X}]/(f)$, $\eta := [\mathtt{X}]_f \in E$, and ϵ as defined in (21.6), the algorithm runs as follows:

Algorithm B1:

let Q be an $\ell \times \ell$ matrix over F (initially with undefined entries)
compute $\alpha \leftarrow \eta^q$ using repeated squaring
$\beta \leftarrow 1_E$
for $i \leftarrow 1$ to ℓ do // *invariant:* $\beta = \alpha^{i-1} = (\eta^{i-1})^q$
    $Q(i) \leftarrow \epsilon^{-1}(\beta)$, $Q(i,i) \leftarrow Q(i,i) - 1$, $\beta \leftarrow \beta\alpha$
compute a basis $v_1, \ldots, v_r$ of the row null space of Q using
    Gaussian elimination
for $i = 1, \ldots, r$ do $\beta_i \leftarrow \epsilon(v_i)$
output $\beta_1, \ldots, \beta_r$

The correctness of Algorithm B1 is clear from the above discussion. As for the running time:

Theorem 21.9. *Algorithm B1 uses $O(\ell^3 + \ell^2 \operatorname{len}(q))$ operations in F.*

Proof. This is just a matter of counting. The computation of α takes $O(\operatorname{len}(q))$ operations in E using repeated squaring, and hence $O(\ell^2 \operatorname{len}(q))$ operations in F. To build the matrix Q, we have to perform an additional $O(\ell)$ operations in E to compute the successive powers of α, which translates into $O(\ell^3)$ operations in F. Finally, the cost of Gaussian elimination is an additional $O(\ell^3)$ operations in F. □

Stage 2: Splitting with B

The second stage of Berlekamp's factoring algorithm is a probabilistic procedure that factors f using a basis $\beta_1, \ldots, \beta_r$ for B. As we did with Algorithm EDF in §21.3.2, we begin by discussing how to efficiently split f into two non-trivial factors, and then we present a somewhat more elaborate algorithm that completely factors f.

Let $M_1 \in F[\mathtt{X}]$ be the polynomial defined by (21.2) and (21.3); that is,

$$M_1 := \begin{cases} \sum_{j=0}^{w-1} \mathtt{X}^{2^j} & \text{if } p = 2, \\ \mathtt{X}^{(q-1)/2} - 1 & \text{if } p > 2. \end{cases}$$

Using our basis for B, we can easily generate a random element β of B by simply choosing $c_1, \ldots, c_r$ at random, and computing $\beta := \sum_i c_i \beta_i$. If $\beta = \theta(b_1, \ldots, b_r)$, then the b_i will be uniformly and independently distributed over F. Just as in Algorithm EDF, $\gcd(\operatorname{rep}(M_1(\beta)), f)$ will be a non-trivial factor of f with probability at least $1/2$, if $p = 2$, and probability at least $4/9$, if $p > 2$.

That is the basic splitting strategy. We turn this into an algorithm to completely factor f using the same technique of iterative refinement that was used in Algorithm EDF. That is, at any stage of the algorithm, we have a partial factorization $f = \prod_{h \in H} h$, which we try to refine by attempting to split each $h \in H$ using the strategy outlined above. One technical difficulty is that to split such a polynomial h, we need to efficiently generate a random element of the Berlekamp subalgebra of $F[\mathtt{X}]/(h)$. A particularly efficient way to do this is to use our basis for the Berlekamp subalgebra of $F[\mathtt{X}]/(f)$ to generate a random element of the Berlekamp subalgebra of $F[\mathtt{X}]/(h)$ for all $h \in H$ simultaneously. Let $g_i := \operatorname{rep}(\beta_i)$ for $i = 1, \ldots, r$. If we choose $c_1, \ldots, c_r \in F$ at random, and set $g := c_1 g_1 + \cdots + c_r g_r$, then $[g]_f$ is a random element of the Berlekamp subalgebra of $F[\mathtt{X}]/(f)$, and by

the Chinese remainder theorem, it follows that the values $[g]_h$ for $h \in H$ are independently distributed, with each $[g]_h$ uniformly distributed over the Berlekamp subalgebra of $F[\mathtt{X}]/(h)$.

Here is the algorithm for completely factoring a polynomial, given a basis for the corresponding Berlekamp subalgebra. It takes as input a monic, square-free polynomial f of degree $\ell > 0$, together with a basis $\beta_1, \ldots, \beta_r$ for the Berlekamp subalgebra of $F[\mathtt{X}]/(f)$. With $g_i := \mathrm{rep}(\beta_i)$ for $i = 1, \ldots, r$, the algorithm runs as follows:

Algorithm B2:

$H \leftarrow \{f\}$
while $|H| < r$ do
    choose $c_1, \ldots, c_r \in F$ at random
    $g \leftarrow c_1 g_1 + \cdots + c_r g_r \in F[\mathtt{X}]$
    $H' \leftarrow \emptyset$
    for each $h \in H$ do
        $\beta \leftarrow [g]_h \in F[\mathtt{X}]/(h)$
        $d \leftarrow \gcd(\mathrm{rep}(M_1(\beta)), h)$
        if $d = 1$ or $d = h$
            then $H' \leftarrow H' \cup \{h\}$
            else $H' \leftarrow H' \cup \{d, h/d\}$
    $H \leftarrow H'$
output H

The correctness of the algorithm is clear. As for its expected running time, we can get a quick-and-dirty upper bound as follows:

- The cost of generating g in each loop iteration is $O(r\ell)$ operations in F. For a given h, the cost of computing $\beta := [g]_h \in F[\mathtt{X}]/(h)$ is $O(\ell \deg(h))$ operations in F, and the cost of computing $M_1(\beta)$ is $O(\deg(h)^2 \operatorname{len}(q))$ operations in F. Therefore, the number of operations in F performed in each iteration of the main loop is at most a constant times

$$r\ell + \ell \sum_{h \in H} \deg(h) + \operatorname{len}(q) \sum_{h \in H} \deg(h)^2$$
$$\leq 2\ell^2 + \operatorname{len}(q) \left(\sum_{h \in H} \deg(h) \right)^2 = O(\ell^2 \operatorname{len}(q)).$$

- The expected number of iterations of the main loop until we get some non-trivial split is $O(1)$.

- The algorithm finishes after getting $r-1$ non-trivial splits.
- Therefore, the total expected cost is $O(r\ell^2 \operatorname{len}(q))$ operations in F.

A more careful analysis reveals:

Theorem 21.10. *Algorithm B2 uses an expected number of*

$$O(\operatorname{len}(r)\ell^2 \operatorname{len}(q))$$

operations in F.

Proof. The proof follows the same line of reasoning as the analysis of Algorithm EDF. Indeed, using the same argument as was used there, the expected number of iterations of the main loop is $O(\operatorname{len}(r))$. As discussed in the paragraph above this theorem, the cost per loop iteration is $O(\ell^2 \operatorname{len}(q))$ operations in F. The theorem follows. □

The bound in the above theorem is tight (see Exercise 21.11 below): unlike Algorithm EDF, we cannot make the multiplicative factor of $\operatorname{len}(r)$ go away.

21.4.3 Analysis of the whole algorithm

Putting together Algorithm SFD with algorithms B1 and B2, we get Berlekamp's complete factoring algorithm. The running time bound is easily estimated from the results already proved:

Theorem 21.11. *Berlekamp's factoring algorithm uses an expected number of $O(\ell^3 + \ell^2 \operatorname{len}(\ell) \operatorname{len}(q))$ operations in F.*

So we see that Berlekamp's algorithm is in fact faster than the Cantor–Zassenhaus algorithm, whose expected operation count is $O(\ell^3 \operatorname{len}(q))$. The speed advantage of Berlekamp's algorithm grows as q gets large. The one disadvantage of Berlekamp's algorithm is space: it requires space for $\Theta(\ell^2)$ elements of F, while the Cantor–Zassenhaus algorithm requires space for only $O(\ell)$ elements of F. One can in fact implement the Cantor–Zassenhaus algorithm so that it uses $O(\ell^3 + \ell^2 \operatorname{len}(q))$ operations in F, while using space for only $O(\ell^{1.5})$ elements of F—see Exercise 21.13 below.

Exercise 21.10. Give an example of a family of input polynomials f that cause Algorithm SFD to use at least $\Omega(\ell^3)$ operations in F, where $\ell := \deg(f)$.

Exercise 21.11. Give an example of a family of input polynomials f that cause Algorithm B2 to use an expected number of at least $\Omega(\ell^2 \operatorname{len}(\ell) \operatorname{len}(q))$ operations in F, where $\ell := \deg(f)$.

Exercise 21.12. Using the ideas behind Berlekamp's factoring algorithm, devise a deterministic irreducibility test that, given a monic polynomial of degree ℓ over F, uses $O(\ell^3 + \ell^2 \operatorname{len}(q))$ operations in F.

Exercise 21.13. This exercise develops a variant of the Cantor–Zassenhaus algorithm that uses $O(\ell^3 + \ell^2 \operatorname{len}(q))$ operations in F, while using space for only $O(\ell^{1.5})$ elements of F. By making use of Algorithm SFD (which with a bit of care can be implemented so as to use space for $O(\ell)$ elements of F) and the variant of Algorithm EDF discussed in Exercise 21.8, our problem is reduced to that of implementing Algorithm DDF within the stated time and space bounds, assuming that the input polynomial is square-free.

(a) For non-negative integers i, j, with $i \neq j$, show that the irreducible polynomials in $F[\mathtt{X}]$ that divide $\mathtt{X}^{q^i} - \mathtt{X}^{q^j}$ are precisely those whose degree divides $i - j$.

(b) Let $f \in F[\mathtt{X}]$ be a monic polynomial of degree $\ell > 0$, and let $m \approx \ell^{1/2}$. Let $\eta := [\mathtt{X}]_f \in E$, where $E := F[\mathtt{X}]/(f)$. Show how to compute

$$\eta^q, \eta^{q^2}, \ldots, \eta^{q^{m-1}} \in E \quad \text{and} \quad \eta^{q^m}, \eta^{q^{2m}}, \ldots, \eta^{q^{(m-1)m}} \in E$$

using $O(\ell^3 + \ell^2 \operatorname{len}(q))$ operations in F, and space for $O(\ell^{1.5})$ elements of F.

(c) Combine the results of parts (a) and (b) to implement Algorithm DDF on square-free inputs of degree ℓ, so that it uses $O(\ell^3 + \ell^2 \operatorname{len}(q))$ operations in F, and space for $O(\ell^{1.5})$ elements of F.

21.5 Deterministic factorization algorithms (∗)

The algorithms of Cantor and Zassenhaus and of Berlekamp are probabilistic. The exercises below develop a deterministic variant of the Cantor–Zassenhaus algorithm. (One can also develop deterministic variants of Berlekamp's algorithm, with similar complexity.)

This algorithm is only practical for finite fields of small characteristic, and is anyway mainly of theoretical interest, since from a practical perspective, there is nothing wrong with the above probabilistic method. In all of these exercises, we assume that we have access to a basis $\epsilon_1, \ldots, \epsilon_w$ for F as a vector space over $\mathbb{Z}_p$.

To make the Cantor–Zassenhaus algorithm deterministic, we only need to develop a deterministic variant of Algorithm EDF, as Algorithm DDF is already deterministic.

EXERCISE 21.14. Let $g = g_1 \cdots g_r$, where the g_i are distinct monic irreducible polynomials in $F[\mathtt{X}]$. Assume that $r > 1$, and let $\ell := \deg(g)$. For this exercise, the degrees of the g_i need not be the same. For an intermediate field F', with $\mathbb{Z}_p \subseteq F' \subseteq F$, let us call a set $S = \{\lambda_1, \ldots, \lambda_s\}$ of polynomials in $F[\mathtt{X}]_{<\ell}$ a **separating set for g over F'** if the following conditions hold:

- for $i = 1, \ldots, r$ and $u = 1, \ldots, s$, there exists $c_{ui} \in F'$ such that $\lambda_u \equiv c_{ui} \pmod{g_i}$, and
- for any distinct pair of indices i, j, with $1 \leq i < j \leq r$, there exists $u = 1, \ldots, s$ such that $c_{ui} \neq c_{uj}$.

Show that if S is a separating set for g over $\mathbb{Z}_p$, then the following algorithm completely factors g using $O(p|S|\ell^2)$ operations in F.

```
H ← {g}
for each λ ∈ S do
    for each a ∈ Z_p do
        H' ← ∅
        for each h ∈ H do
            d ← gcd(λ − a, h)
            if d = 1 or d = h
                then H' ← H' ∪ {h}
                else H' ← H' ∪ {d, h/d}
        H ← H'
output H
```

EXERCISE 21.15. Let g be as in the previous exercise. Show that if S is a separating set for g over F, then the set

$$S' := \{\sum_{i=0}^{w-1} (\epsilon_j \lambda)^{p^i} \bmod g : 1 \leq j \leq w,\ \lambda \in S\}$$

is a separating set for g over $\mathbb{Z}_p$. Show how to compute this set using $O(|S|\ell^2 \operatorname{len}(p) w(w-1))$ operations in F.

EXERCISE 21.16. Let g be as in the previous two exercises, but further suppose that each irreducible factor of g is of the same degree, say k. Let $E := F[\mathtt{X}]/(g)$ and $\eta := [\mathtt{X}]_g \in E$. Define the polynomial $\phi \in E[\mathtt{Y}]$ as follows:

$$\phi := \prod_{i=0}^{k-1} (\mathtt{Y} - \eta^{q^i}).$$

If

$$\phi = \mathtt{Y}^k + \alpha_{k-1}\mathtt{Y}^{k-1} + \cdots + \alpha_0,$$

with $\alpha_0, \ldots, \alpha_{k-1} \in E$, show that the set

$$S := \{\mathrm{rep}(\alpha_i) : 0 \leq i \leq k-1\}$$

is a separating set for g over F, and can be computed deterministically using $O(k^2 + k\,\mathrm{len}(q))$ operations in E, and hence $O(k^2\ell^2 + k\ell^2\,\mathrm{len}(q))$ operations in F.

Exercise 21.17. Put together all of the above pieces, together with Algorithm DDF, so as to obtain a deterministic algorithm for factoring polynomials over F that runs in time at most p times a polynomial in the input length, and make a careful estimate of the running time of your algorithm.

Exercise 21.18. It is a fact that when our prime p is odd, then for all integers a, b, with $a \not\equiv b \pmod{p}$, there exists a non-negative integer $i \leq p^{1/2}\log_2 p$ such that $(a+i \mid p) \neq (b+i \mid p)$ (here, "$(\cdot \mid \cdot)$" is the Legendre symbol). Using this fact, design and analyze a deterministic algorithm for factoring polynomials over F that runs in time at most $p^{1/2}$ times a polynomial in the input length.

The following two exercises show that the problem of factoring polynomials over F reduces in deterministic polynomial time to the problem of finding roots of polynomials over $\mathbb{Z}_p$.

Exercise 21.19. Let g be as in Exercise 21.14. Suppose that $S = \{\lambda_1, \ldots, \lambda_s\}$ is a separating set for g over $\mathbb{Z}_p$, and $\phi_u \in F[\mathtt{X}]$ is the minimal polynomial over F of $[\lambda_u]_g \in F[\mathtt{X}]/(g)$ for $u = 1, \ldots, s$. Show that each ϕ_u is the product of linear factors over $\mathbb{Z}_p$, and that given S along with the roots of all the ϕ_u, we can deterministically factor g using $(|S| + \ell)^{O(1)}$ operations in F. Hint: see Exercise 17.9.

Exercise 21.20. Using the previous exercise, show that the problem of factoring a polynomial over a finite field F reduces in deterministic polynomial time to the problem of finding roots of polynomials over the prime field of F.

21.6 Faster square-free decomposition (∗)

The algorithm presented in §21.4.1 for square-free decomposition was simple and suitable for our immediate purposes, but is certainly not the most efficient algorithm possible. The following exercises develop a faster algorithm for this problem.

We begin with an exercise that more fully develops the connection be-

tween square-free polynomials and formal derivatives for polynomials over arbitrary fields:

EXERCISE 21.21. Let K be a field, and let $f \in K[\mathtt{X}]$ with $\deg(f) > 0$.

(a) Show that if $\mathbf{D}(f) = 0$, then the characteristic of K must be a prime p, and f must be of the form $f = g(\mathtt{X}^p)$ for some $g \in K[\mathtt{X}]$.

(b) Show that if K is a finite field or a field of characteristic zero, then f is square-free if and only if $d := \gcd(f, \mathbf{D}(f)) = 1$; moreover, if $d \neq 1$, then either $\deg(d) < \deg(f)$, or K has prime characteristic p and $f = h^p$ for some $h \in K[\mathtt{X}]$.

(c) Give an example of a field K of characteristic p and an irreducible polynomial $f \in K[\mathtt{X}]$ such that $f = g(\mathtt{X}^p)$ for some $g \in K[\mathtt{X}]$.

Next, we consider the problem of square-free decomposition of polynomials over fields of characteristic zero, which is simpler than the corresponding problem over finite fields.

EXERCISE 21.22. Let $f \in K[\mathtt{X}]$ be a monic polynomial over a field K of characteristic zero. Suppose that the factorization of f into irreducibles is

$$f = f_1^{e_1} \cdots f_r^{e_r}.$$

Show that

$$\frac{f}{\gcd(f, \mathbf{D}(f))} = f_1 \cdots f_r.$$

EXERCISE 21.23. Let K be a field of characteristic zero. Consider the following algorithm that takes as input a monic polynomial $f \in K[\mathtt{X}]$ of degree $\ell > 0$:

$j \leftarrow 1,\ \ g \leftarrow f/\gcd(f, \mathbf{D}(f))$
repeat
    $f \leftarrow f/g,\ \ h \leftarrow \gcd(f, g),\ \ m \leftarrow g/h$
    if $m \neq 1$ then output (m, j)
    $g \leftarrow h,\ \ j \leftarrow j + 1$
until $g = 1$

Using the result of the previous exercise, show that this algorithm outputs a list of pairs (g_i, s_i), such that each g_i is square-free, $f = \prod_i g_i^{s_i}$, and the g_i are pairwise relatively prime. Furthermore, show that this algorithm uses $O(\ell^2)$ operations in K.

We now turn our attention to square-free decomposition over finite fields.

EXERCISE 21.24. Let $f \in F[\mathtt{X}]$ be a monic polynomial over F (which, as usual, has characteristic p and cardinality $q = p^w$). Suppose that the factorization of f into irreducibles is

$$f = f_1^{e_1} \cdots f_r^{e_r}.$$

Show that

$$\frac{f}{\gcd(f, \mathbf{D}(f))} = \prod_{\substack{1 \le i \le r \\ e_i \not\equiv 0 \pmod{p}}} f_i.$$

EXERCISE 21.25. Consider the following algorithm that takes as input a monic polynomial $f \in F[\mathtt{X}]$ of degree $\ell > 0$:

```
s ← 1
repeat
    j ← 1,  g ← f/gcd(f, D(f))
    repeat
        f ← f/g,  h ← gcd(f, g),  m ← g/h
        if m ≠ 1 then output (m, js)
        g ← h,  j ← j + 1
    until g = 1
    if f ≠ 1 then // f is a pth power
        // we compute a pth root as in Algorithm SFD
        f ← f^{1/p},  s ← ps
until f = 1
```

Using the result of the previous exercise, show that this algorithm outputs a list of pairs (g_i, s_i), such that each g_i is square-free, $f = \prod_i g_i^{s_i}$, and the g_i are pairwise relatively prime. Furthermore, show that this algorithm uses $O(\ell^2 + \ell(w-1)\operatorname{len}(p)/p)$ operations in F.

21.7 Notes

The average-case analysis of Algorithm IPT, assuming its input is random, and the application to the analysis of Algorithm RIP, is essentially due to Ben-Or [14]. If one implements Algorithm RIP using fast polynomial arithmetic, one gets an expected cost of $O(\ell^{2+o(1)} \operatorname{len}(q))$ operations in F. Note that Ben-Or's analysis is a bit incomplete—see Exercise 32 in Chapter 7 of Bach and Shallit [12] for a complete analysis of Ben-Or's claims.

The asymptotically fastest probabilistic algorithm for constructing an irreducible polynomial over F of degree ℓ is due to Shoup [91]. That algorithm uses an expected number of $O(\ell^{2+o(1)} + \ell^{1+o(1)} \operatorname{len}(q))$ operations in F, and

in fact does not follow the "generate and test" paradigm of Algorithm RIP, but uses a completely different approach. Exercise 21.2 is based on [91].

As far as *deterministic* algorithms for constructing irreducible polynomials of given degree over F, the only known methods are efficient when the characteristic p of F is small (see Chistov [26], Semaev [83], and Shoup [89]), or under a generalization of the Riemann hypothesis (see Adleman and Lenstra [4]). Shoup [89] in fact shows that the problem of constructing an irreducible polynomial of given degree over F is deterministic, polynomial-time reducible to the problem of factoring polynomials over F.

The algorithm in §21.2 for computing minimal polynomials over finite fields is due to Gordon [41].

The Cantor–Zassenhaus algorithm was initially developed by Cantor and Zassenhaus [24], although many of the basic ideas can be traced back quite a ways. A straightforward implementation of this algorithm using fast polynomial arithmetic uses an expected number of $O(\ell^{2+o(1)}\operatorname{len}(q))$ operations in F.

Berlekamp's algorithm was initially developed by Berlekamp [15, 16], but again, the basic ideas go back a long way. A straightforward implementation using fast polynomial arithmetic uses an expected number of $O(\ell^3 + \ell^{1+o(1)}\operatorname{len}(q))$ operations in F; the term ℓ^3 may be replaced by ℓ^ω, where ω is the exponent of matrix multiplication (see §15.6).

There are no known efficient, deterministic algorithms for factoring polynomials over F when the characteristic p of F is large (even under a generalization of the Riemann hypothesis, except in certain special cases).

The square-free decomposition of a polynomial over a field K of characteristic zero can be computed using an algorithm of Yun [105] using $O(\ell^{1+o(1)})$ operations in K. Yun's algorithm can be adapted to work over finite fields as well (see Exercise 14.30 in von zur Gathen and Gerhard [37]).

The asymptotically fastest algorithms for factoring polynomials over a finite field F are due to von zur Gathen, Kaltofen, and Shoup: the algorithm of von zur Gathen and Shoup [38] uses an expected number of $O(\ell^{2+o(1)} + \ell^{1+o(1)}\operatorname{len}(q))$ operations in F; the algorithm of Kaltofen and Shoup [51] has a cost that is subquadratic in the degree—it uses an expected number of $O(\ell^{1.815}\operatorname{len}(q)^{0.407})$ operations in F. Exercises 21.1, 21.7, and 21.8 are based on [38]. Although the "fast" algorithms in [38] and [51] are mainly of theoretical interest, a variant in [51], which uses $O(\ell^{2.5} + \ell^{1+o(1)}\operatorname{len}(q))$ operations in F, and space for $O(\ell^{1.5})$ elements of F, has proven to be quite practical (Exercise 21.13 develops some of these ideas; see also Shoup [92]).