Factorization of polynomials over finite fields

From Wikipedia, the free encyclopedia

In mathematics and computer algebra the factorization of a polynomial consists of decomposing it into a product of irreducible factors. This decomposition is theoretically possible and is unique for polynomials with coefficients in any field, but rather strong restrictions on the field of the coefficients are needed to allow the computation of the factorization by means of an algorithm. In practice, algorithms have been designed only for polynomials with coefficients in a finite field, in the field of rationals or in a finitely generated field extension of one of them.

The case of the **factorization of univariate polynomials over a finite field**, which is the subject of this article, is especially important, because all the algorithms (including the case of multivariate polynomials over the rational numbers), which are sufficiently efficient to be implemented, reduce the problem to this case (see Polynomial factorization). It is also interesting for various applications of finite fields, such as coding theory (cyclic redundancy codes and BCH codes), cryptography (public key cryptography by the means of elliptic curves), and computational number theory.

As the reduction of the factorization of multivariate polynomials to that of univariate polynomials does not have any specificity in the case of coefficients in a finite field, only polynomials with one variable are considered in this article.

Contents

Background

Finite field

Main article: Finite field

The theory of finite fields, whose origins can be traced back to the works of Gauss and Galois, has played a part in various branches of mathematics. Due to the applicability of the concept in other topics of mathematics and sciences like computer science there has been a resurgence of interest in finite fields and this is partly due to important applications in coding theory and cryptography. Applications of finite fields introduce some of these developments in cryptography, computer algebra and coding theory.

A finite field or Galois field is a field with a finite order (number of elements). The order of a finite field is always a prime or a power of prime. For each prime power $q = p^r$, there exists exactly one finite field with q elements, up to isomorphism. This field is denoted $GF(q)$ or $\mathbf{F}_q$. If p is prime, $GF(p)$ is the prime field of order p; it is the field of residue classes modulo p, and its p elements are denoted 0, 1, ..., p–1. Thus $a = b$ in $GF(p)$ means the same as $a \equiv b \pmod{p}$.

Irreducible polynomials

Let F be a finite field. As for general fields, a non-constant polynomial f in $F[x]$ is said to be irreducible over F if it is not the product of two polynomials of positive degree. A polynomial of positive degree that is not irreducible over F is called *reducible over F*.

Irreducible polynomials allow to construct the finite fields of non prime order. In fact, for a prime power q, let $\mathbf{F}_q$ be the finite field with q elements, unique up to an isomorphism. A polynomial f of degree n greater than one, which is irreducible over $\mathbf{F}_q$, defines a field extension of degree n which is isomorphic to the field with q^n elements: the elements of this extension are the polynomials of degree lower than n; addition, subtraction and multiplication by an element of $\mathbf{F}_q$ are those of the polynomials; the product of two elements it the remainder of the division by f of their product as polynomials; the inverse of an element may be computed by the extended GCD algorithm (see Arithmetic of algebraic extensions).

It follows that, to compute in a finite field of non prime order, one needs to generate an irreducible polynomial. For this, the common method is to take a polynomial at random and test it for irreducibility. For sake of efficiency of the multiplication in the field, it is usual to search for polynomials of the shape $x^n + ax + b$.[*citation needed*]

Irreducible polynomials over finite fields are also useful for Pseudorandom number generators using feedback shift registers and discrete logarithm over $\mathbf{F}_{2^n}$.

Example

The polynomial $P = x^4 + 1$ is irreducible over $\mathbf{Q}$ but not over any finite field.

- On any field extension of $\mathbf{F}_2$, $P = (x+1)^4$.
- On every other finite field, at least one of −1, 2 and −2 is a square, because the product of two non squares is a square and so we have

1. If $-1 = a^2$, then $P = (x^2 + a)(x^2 - a)$.
2. If $2 = b^2$, then $P = (x^2 + bx + 1)(x^2 - bx + 1)$.
3. If $-2 = c^2$, then $P = (x^2 + cx - 1)(x^2 - cx - 1)$.

Complexity

Polynomial factoring algorithms use basic polynomial operations such as products, divisions, gcd, powers of one polynomial modulo another, etc. A multiplication of two polynomials of degree at most n can be done in $O(n^2)$ operations in $\mathbf{F}_q$ using "classical" arithmetic, or in $O(n\log(n))$ operations in $\mathbf{F}_q$ using "fast" arithmetic. A Euclidean division (division with remainder) can be performed within the same time bounds. The cost of a polynomial greatest common divisor between two polynomials of degree at most n can be taken as $O(n^2)$ operations in $\mathbf{F}_q$ using classical methods, or as $O(n\log^2(n))$ operations in $\mathbf{F}_q$ using fast methods. For polynomials h, g of degree at most n, the exponentiation h^q mod g can be done with $O(\log(q))$ polynomial products, using exponentiation by squaring method, that is $O(n^2\log(q))$ operations in $\mathbf{F}_q$ using classical methods, or $O(n\log(q)\log(n))$ operations in $\mathbf{F}_q$ using fast methods.

In the algorithms that follow, the complexities are expressed in terms of number of arithmetic operations in $\mathbf{F}_q$, using classical algorithms for the arithmetic of polynomials.

Factoring algorithms

Many algorithms for factoring polynomials over finite fields include the following three stages:

Square-free factorization

The algorithm determines a square-free factorization for polynomials whose coefficients come from the finite field $\mathbf{F}_q$ of order $q = p^m$ with p a prime. This algorithm firstly determines the derivative and then computes the gcd of the polynomial and its derivative. If it is not one then the gcd is again divided into the original polynomial, provided that the derivative is not zero (a case that exists for non-constant polynomials defined over finite fields).

This algorithm uses the fact that, if the derivative of a polynomial is zero, then it is a polynomial in x^p, which is the pth power of the polynomial obtained by substituting x by $x^{1/p}$.

This algorithm works also over a field of characteristic zero, with the only difference that it never enters in the blocks of instructions where pth roots are computed. However, in this case, Yun's algorithm is much more efficient because it computes the greatest common divisors of polynomials of lower degrees. A consequence is that, when factoring a polynomial over the integers, the algorithm which follows is not used: one compute first the square-free factorization over the integers, and to factor the resulting polynomials, one chooses a p such that they remain square-free modulo p.

```
Algorithm: SFF (Square-Free Factorization)
Input: A monic polynomial f in F_q[x]
Output: Square-free factorization of f
```

```
i←1; R ← 1; g ← f';
if g ≠ 0 then {
   c ← gcd(f, g);
   w ← f/c;
   while w ≠ 1 do {
         y ← gcd(w, c); z ← w/y;
         R ← R·z^i; i ← i+1;
         w ← y; c ← c/y }
   if c ≠ 1 then {
         c ← c^(1/p);
         Output(R·SFF(c)^p) }
   else  Output(R)
```

```
else {
        f ← f^(1/p);
        Output(SFF(f)^p) }
end.
```

Example of a square-free factorization

Let

$$f = x^{11} + 2x^9 + 2x^8 + x^6 + x^5 + 2x^3 + 2x^2 + 1 \in \mathbf{F}_3[x],$$

to be factored over the field with three elements.

The algorithm computes first

$$c = \gcd(f, f') = x^9 + 2x^6 + x^3 + 2.$$

Since the derivative is non-zero we have $w = f/c = x^2 + 2$ and we enter the while loop. After one loop we have $y = x + 2, z = x + 1$ and $R = x + 1$ with updates $i = 2, w = x + 2$ and $c = x^8 + x^7 + x^6 + x^2+x+1$. The second time through the loop gives $y = x + 2, z = 1, R = x + 1$, wi updates $i = 3, w = x + 2$ and $c = x^7 + 2x^6 + x + 2$. The third time through the loop also does not change R. For the fourth time through the loop we get $y = 1, z = x + 2, R = (x + 1)(x + 2)^4$, with updates $i = 5, w = 1$ and $c = x^6 + 1$. Since $w = 1$, we exit the while loop. Since $c \neq 1$, it must be a perfect cube. The cube root of c, obtained by replacing x^3 by x is $x^2 + 1$, and calling the square-free procedure recursively determines that it is square-free. Therefore, cubing it and combining it with the value of R to that point gives the square-free decomposition

$$f = (x + 1)(x^2 + 1)^3(x + 2)^4.$$

Distinct-degree factorization

This algorithm splits a square-free polynomial into a product of polynomials whose irreducible factors all have the same degree. Let $f \in \mathbf{F}_q[x]$ of degree n be the polynomial to be factored.

```
Algorithm Distinct-degree factorization(DDF)
Input: A monic square-free polynomial  f ∈ F_q[x]
Output: The set of all pairs (g, d), such that
          f has an irreducible factor of degree d and
          g is the product of all monic irreducible factors of f of degree d.
Begin
    i := 1;      S := ∅,      f* := f;
    while deg f* ≥ 2i do
        g = gcd(f*, x^(q^i) − x)
        if g ≠ 1, then
          S := S ∪ (g, i);
          f* := f*/g;
        end if
        i := i+1;
    end while;
    if f* ≠ 1, then S := S ∪ (f*, deg f*);
    if S = ∅
        then return {(f, 1)}
```

```
        else return S
End
```

The correctness of the algorithm is based on the following:

Lemma. For $i \geq 1$ the polynomial

$$x^{q^i} - x \in \mathbf{F}_q[x]$$

is the product of all monic irreducible polynomials in $\mathbf{F}_q[x]$ whose degree divides i.

At first glance, this is not efficient since it involves computing the GCD of polynomials of a degree which is exponential in the degree of the input polynomial. However

$$g = \gcd\left(f^*, x^{q^i} - x\right)$$

may be replaced by

$$g = \gcd\left(f^*, \left(x^{q^i} - x \mod f^*\right)\right).$$

Therefore we have to compute:

$$x^{q^i} - x \mod f^*,$$

there are two methods:

Method I. Start from the value of

$$x^{q^{i-1}} \mod f^*$$

computed at the preceding step and to compute its q-th power modulo the new f^*, using exponentiation by squaring method. This needs

$$O\left(\log(q)\deg(f)^2\right)$$

arithmetic operations in $\mathbf{F}_q$ at each step, and thus

$$O\left(\log(q)\deg(f)^3\right)$$

arithmetic operations for the whole algorithm.

Method II. Using the fact that the q-th power is a linear map over $\mathbf{F}_q$ we may compute its matrix with

$$O\left(\deg(f)^2(\log(q) + \deg(f))\right)$$

operations. Then at each iteration of the loop, compute the product of a matrix by a vector (with $O(\deg(f)^2)$ operations). This induces a total number of operations in $\mathbf{F}_q$ which is

$$O\left(\deg(f)^2(\log(q)+\deg(f))\right).$$

Thus this second method is more efficient and is usually preferred. Moreover, the matrix that is computed in this method is used, by most algorithms, for equal-degree factorization (see below); thus using it for the distinct-degree factorization saves further computing time.

Equal-degree factorization

Main article: Cantor–Zassenhaus algorithm

In this section, we consider the factorization of a monic squarefree univariate polynomial f, of degree n, over a finite field $\mathbf{F}_q$, which has $r \geq 2$ pairwise distinct irreducible factors $f_1, \ldots, f_r$ each of degree d.

We first describe an algorithm by Cantor and Zassenhaus (1981) and then a variant that has a slightly better complexity. Both are probabilistic algorithms whose running time depends on random choices (Las Vegas algorithms), and have a good average running time. In next section we describe an algorithm by Shoup (1990), which is also an equal-degree factorization algorithm, but is deterministic. All these algorithms require an odd order q for the field of coefficients. For more factorization algorithms see e.g. Knuth's book The Art of Computer Programming volume 2.

```
Algorithm Cantor–Zassenhaus algorithm.
Input: A finite field F_q of odd order q.
       A monic square free polynomial f in F_q[x] of degree n = rd,
            which has r ≥ 2 irreducible factors each of degree d
Output: The set of monic irreducible factors of f.
```

```
Factors:={f};
while Size(Factors) < r do,
   Choose h in F_q[x] with deg(h) < n at random;
   g := h^((q^d − 1)/2) − 1  (mod f)
   for each u in Factors with deg(u) > d do
       if gcd(g, u) ≠ 1 and gcd(g, u) ≠ u, then
         Factors:= Factors \ {u} ∪ {(gcd(g, u), u/gcd(g, u))};
       endif;
endwhile
return Factors.
```

The correctness of this algorithm relies on the fact that the ring $\mathbf{F}_q[x]/f$ is a direct product of the fields $\mathbf{F}_q[x]/f_i$ where f_i runs on the irreducible factors of f. As all these fields have q^d elements, the component of g in any of these fields is zero with probability

$$\frac{q^d-1}{2q^d} \sim \frac{1}{2}.$$

This implies that the polynomial $\gcd(g, u)$ is the product of the factors of g for which the component of g is zero.

It has been shown that the average number of iterations of the while loop of the algorithm is less than $2.5\log_2 r$, giving an average number of arithmetic operations in $\mathbf{F}_q$ which is $O(dn^2\log(r)\log(q))$.[1]

In the typical case where $d\log(q) > n$, this complexity may be reduced to

$$O(n^2(\log(r)\log(q)+n))$$

by choosing h in the kernel of the linear map

$$v \to v^q - v \pmod f$$

and replacing the instruction

$$g := h^{\frac{q^d-1}{2}} - 1 \pmod f$$

by

$$g := h^{\frac{q-1}{2}} - 1 \pmod f.$$

The proof of validity is the same as above, replacing the direct product of the fields $\mathbf{F}_q[x]/f_i$ by the direct product of their subfields with q elements. The complexity is decomposed in $O(n^2\log(r)\log(q))$ for the algorithm itself, $O(n^2(\log(q)+n))$ for the computation of the matrix of the linear map (which may be already computed in the square-free factorization) and $O(n^3)$ for computing its kernel. It may be noted that this algorithm works also if the factors have not the same degree (in this case the number r of factors, needed for stopping the while loop, is found as the dimension of the kernel). Nevertheless, the complexity is slightly better if square-free factorization is done before using this algorithm (as n may decrease with square-free factorization, this reduces the complexity of the critical steps).

Victor Shoup's algorithm

Like the algorithms of the preceding section, Victor Shoup's algorithm is an equal-degree factorization algorithm.[2] Unlike them, it is a deterministic algorithm. However, it is less efficient, in practice, that the algorithms of preceding section. For Shoup's algorithm, the input is restricted to polynomials over prime fields $\mathbf{F}_q$.

Let $g = g_1 \dots g_k$ be the desired factorization, where the g_i are distinct monic irreducible polynomials of degree d. Let $n = \deg(g) = kd$. We consider the ring $R = \mathbf{F}_q[x]/g$ and denote also by x the image of x in R. The ring R is the direct product of the fields $R_i = \mathbf{F}_q[x]/g_i$, and we denote by p_i the natural homomorphism from the R onto R_i. The Galois group of R_i over $\mathbf{F}_q$ is cyclic of order d, generated by the field automorphism $u \to u^p$. It follows that the roots of g_i in R_i are

$$p_i(x), p_i(x^q), p_i\left(x^{q^2}\right), p_i\left(x^{q^{d-1}}\right).$$

If $q > n$, the Newton's identities allow to compute the s_i with

Like in the preceding algorithm, this algorithm uses the same subalgebra B of R as the Berlekamp's algorithm, sometimes called the "Berlekamp subagebra" and defined as

$$\begin{aligned} B &= \{\alpha \in R \ : \ p_1(\alpha), \cdots, p_k(\alpha) \in \mathbf{F}_q\} \\ &= \{u \in R \ : \ u^q = u\} \end{aligned}$$

A subset S of B is said a separating set if, for every $1 \le i < j \le k$ there exists $s \in S$ such that $p_i(s) \neq p_j(s)$. In the preceding algorithm, a separating set is constructed by choosing at random the elements of S. In Shoup's algorithm, the separating set is constructed in the following way. Let s in $R[Y]$ be such that

$$\begin{aligned} s &= (Y-x)\left(Y-x^q\right)\cdots\left(Y-x^{q^{d-1}}\right) \\ &= s_0 + \cdots + s_{d-1}Y^{d-1} + Y^d \end{aligned}$$

Then $\{s_0, \ldots, s_{d-1}\}$ is a separating set because $p_i(s) = g_i$ for i =1, ..., k (the two monic polynomials have the same roots). As the g_i are pairwise distinct, for every pair of distinct indexes (i, j), at least one of the coefficients s_h will satisfy $p_i(s_h) \neq p_j(s_h)$.

Having a separating set, Shoup's algorithm proceeds as the last algorithm of the preceding section, simply by replacing the instruction "choose at random h in the kernel of the linear map $v \to v^q - v \pmod f$" by "choose $h + i$ with h in S and i in {1, ..., k–1}".

Rabin's test of irreducibility

Like distinct-degree factorization algorithm, Rabin's algorithm[3] is based on the Lemma stated above. Distinct-degree factorization algorithm tests every d not greater than half the degree of the input polynomial. Rabin's algorithm takes advantage that the factors are not needed for considering fewer d. Otherwise, it is similar to distinct-degree factorization algorithm. It is based on the following fact.

Let p_1, ..., p_k, be all the prime divisors of n, and denote $n/p_i = n_i$, for $1 \le i \le k$ polynomial f in $\mathbf{F}_q[x]$ of degree n is irreducible in $\mathbf{F}_q[x]$ if and only if $\gcd\left(f, x^{q^{n_i}} - x\right) = 1$, for $1 \le i \le k$, and f divides $x^{q^n} - x$. In fact, if f has a factor of degree not dividing n, then f does not divide $x^{q^n} - x$; if f has a factor of degree dividing n, then this factor divides at least one of the $x^{q^{n_i}} - x$.

```
Algorithm Rabin Irreducibility Test
Input: A monic polynomial f in F_q[x] of degree n,
       p_1, ..., p_k all distinct prime divisors of n.
Output: Either "f is irreducible" or "f is reducible".
Begin
    for j = 1 to k do
       n_j = n/p_j;
    for i = 1 to k do
       h := x^(q^n_i) - x mod f;
       g := gcd(f, h);
       if g ≠ 1, then return 'f is reducible' and STOP;
    end for;
    g := x^(q^n) - x mod f;
    if g = 0, then return "f is irreducible",
        else return "f is reducible"
end.
```

The basic idea of this algorithm is to compute $x^{q^{n_i}} \bmod f$ starting from the smallest $n_1, \ldots, n_k$ by repeated squaring or using the Frobenius automorphism, and then to take the correspondent gcd. Using the elementary polynomial arithmetic, the computation of the matrix of the Frobenius automorphism needs $O(n^2(n + \log q))$ operations in $\mathbf{F}_q$, the computation of

$$x^{q^{n_i}} - x \pmod f$$

needs $O(n^3)$ further operations, and the algorithm itself needs $O(kn^2)$ operations, giving a total of $O(n^2(n+\log q))$ operations in $\mathbf{F}_q$. Using fast arithmetic (complexity $O(n \log n)$ for multiplication and division, and $O(n(\log n)^2)$ for GCD computation), the computation of the $x^{q^{n_i}} - x \bmod f$ by repeated squaring is $O(n^2 \log n \log q)$, and the algorithm itself is $O(kn(\log n)^2)$, giving a total of $O(n^2 \log n \log q)$ operations in $\mathbf{F}_q$.

See also

- Berlekamp's algorithm
- Cantor–Zassenhaus algorithm
- Polynomial factorization

References

- KEMPFERT,H (1969) On the *Factorization of Polynomials* Department of Mathematics, The Ohio State University,Columbus,Ohio 43210
- Shoup,Victor (1996) *Smoothness and Factoring Polynomials over Finite Fields* Computer Science Department University of Toronto
- Von Zur Gathen, J., Panario, D. (2001) Factoring Polynomials Over Finite Fields: A Survey . Fachbereich Mathematik-Informatik, Universitat Paderborn. Department of Computer Science, University of Toronto.
- Gao Shuhong, Panario Daniel,*Test and Construction of Irreducible Polynomials over Finite Fields* Department of mathematical Sciences, Clemson University, South Carolina, 29634-1907, USA. and Department of computer science University of Toronto, Canada M5S-1A4
- Shoup, Victor (1989) New Algorithms for Finding Irreducible Polynomials over Finite Fields Computer Science Department University of Wisconsin–Madison
- Geddes, K.O. (1990) Algorithms for Computer Algebra

External links

- Some irreducible polynomials http://www.math.umn.edu/~garrett/m/algebra/notes/07.pdf
- Field and Galois Theory :http://www.jmilne.org/math/CourseNotes/FT.pdf
- Galois Field:http://designtheory.org/library/encyc/topics/gf.pdf
- Factoring polynomials over finite fields: http://www.science.unitn.it/~degraaf/compalg/polfact.pdf

Notes

1. ^ Flajolet, Philippe; Steayaert, Jean-Marc (1982), "A branching process arising in dynamic hashing, trie searching and polynomial factorization", *Automata, languages and programming (Aarhus, 1982)*, Lecture Notes in Comput. Sci. **140**, Springer, pp. 239–251
2. ^ Victor Shoup, On the deterministic complexity of factoring polynomials over finite fields, Information Processing Letters 33:261-267, 1990
3. ^ Rabin, Michael (1980). "Probabilistic algorithms in finite fields". *SIAM Journal on Computing* **9** (2): 273–280. doi:10.1137/0209024 (http://dx.doi.org/10.1137%2F0209024).

Categories: Polynomials | Algebra | Computer algebra | Coding theory | Cryptography

Computational number theory