The Berlekamp algorithm

John Kerl
University of Arizona Department of Mathematics
2009 Integration Workshop

August 6, 2009

Abstract

Integer factorization is a Hard Problem. Some cryptosystems, such as RSA, are in fact designed around the difficulty of integer factorization. For polynomials with coefficients in the finite field $\mathbb{F}_q$, on the other hand, we can use the Berlekamp algorithm to factor polynomials of high degree in reasonable amounts of time.

In this project, you will see how the algorithm works, prove its correctness, and analyze its computational complexity. This project is aimed toward those with interests in computational algebra, finite fields, and/or linear algebra.

Contents

1 Theory

1.1 Definitions

Definition 1.1. Let $f \in \mathbb{F}_q[x]$ have degree n. Suppose

$$f = e_1 \cdots e_k$$

be the *prime factorization* of f, for $e_i \in \mathbb{F}_q[x]$ with degrees $d_i = \deg(e_i) \geq 1$. Suppose f is *monic* and that all the e_i's are as well. Assume that f is *squarefree*, i.e. that all the e_i's are distinct. (See section 1.4 to see why this restriction is not a difficulty.) Of course, we don't yet know what the e_i's are, nor what k is; that is the purpose of a factorization algorithm.

Definition 1.2. Let

$$A_f = \mathbb{F}_q[x]/\langle f \rangle.$$

This is a vector space over $\mathbb{F}_q$, of degree n. In fact it is an *algebra*: a vector space wherein you can also multiply the vectors.

Definition 1.3. Let

$$B_f = \{h \in A_f : h^q \equiv h \pmod{f}\}.$$

This is the *Berlekamp subalgebra* of A_f. (You will prove below that this is actually a subalgebra.)

Definition 1.4. Define

$$Q_f : A_f \to A_f$$

by $a(x) \mapsto a(x)^q$ for each $a \in A_f$.

Proposition 1.5. *This is an $\mathbb{F}_q$-linear map from A_f to itself.*

Proof. Prove this. (Hint: use the *freshman's dream*.) □

Proposition 1.6. $B_f = \ker(Q_f - I)$.

Proof. Prove this. (Hint: you will need the *Frobenius automorphism* from finite-field theory.) □

Proposition 1.7. *B_f is a subalgebra of A_f.*

Proof. Prove this. (Hint: You can check subspace axioms, or you can use the fact that B_f is the kernel of a linear map.) □

Remark 1.8. By the Chinese Remainder Theorem,

$$A_f \cong \mathbb{F}_q[x]/\langle e_1 \rangle \times \cdots \times \mathbb{F}_q[x]/\langle e_k \rangle.$$

(Since each of the e_i's is irreducible, each factor is a finite field.)

1.2 Linear algebra

We can find a basis for the vector subspace B_f of A_f using linear algebra. (See section 2.2 for a worked example.) The set $\{x^{n-1}, x^{n-2}, \ldots, x, 1\}$ (where $n = \deg f$) is a basis for A_f. The map Q_f, defined above, is defined by its action on this standard basis. Let the M be the matrix of Q_f, with respect to the standard basis for A_f.

Question: Using both the polynomial point of view and the vector-space point of view, figure out how to write down the matrix for Q. Write $h(x)$ as a vector with respect to the standard basis; do the same for $Qh = h^q$, where the latter is reduced mod f. (Hint: reduction mod f is a ring homomorphism, so it moves through terms.)

Using row reduction, we can compute a basis for $B_f = \ker(Q_f - I)$ using row reduction on the matrix $M - I$. This is an important part of the factorization algorithm, for the following reason.

Proposition 1.9. *The nullity of $Q_f - I$ is the number of irreducible factors of $f(x)$.*

Proof. Please flesh out the details in the following sketch.

We have, by hypothesis,

$$f = e_1 \cdots e_k.$$

By the Chinese Remainder Theorem (mapping how, precisely?),

$$A_f = \mathbb{F}_q[x]/\langle e_1 \rangle \times \cdots \times \mathbb{F}_q[x]/\langle e_k \rangle.$$

Now let $h \in B_f$. The CRT map sends h to $(r_1, \ldots, r_k)$. Since $h \in B_f$,

$$0 = h^q - h = (r_1^q - r_1, \ldots, r_k^q - r_k).$$

This means $r_i^q = r_i$ for each i, so therefore each r_i is in $\mathbb{F}_q$.

We may conclude that $B_f \cong (\mathbb{F}_q)^k$. (How does this relate to the rank and nullity of $Q_f - I$?) □

This means that we can already test for irreducibility, just by checking the nullity of $Q_f - I$. Going ahead and finding a non-trivial factor of $f(x)$ takes a little more work.

1.3 GCDs

Proposition 1.10. *For each $h \in B_f$,*

$$f(x) = \prod_{c \in \mathbb{F}_q} \gcd(f, h - c).$$

Proof. Prove this. (Hint: Show that the left-hand side divides the right-hand side, and vice versa.) □

Corollary 1.11. *For each $h \in B_f$, if $0 < \deg h$, there exists some $c \in \mathbb{F}_q$ such that $g = \gcd(f, h - c) \neq 1$. Such a g is a non-trivial factor of f.*

Proof. Prove this. □

1.4 Squarefree preparation

This section is optional.

Berlekamp's algorithm requires squarefree input — yet, given an input polynomial f, how do you know if it's squarefree? Isn't this the kind of thing you want a factorization algorithm to do for you?

It turns out there is a trick. Given f, let f' be the formal derivative of f with respect to x. Then let

$$g = \gcd(f, f').$$

Case 1: If g has degree 0, then the input f is squarefree, and is ready for Berlekamp.

Question: Prove why this is true.

Case 2: If $f' = 0$: the input is a perfect pth power, where p is the characteristic of $\mathbb{F}_q$. You can find out what this is just by looking at f.

Question: Use the freshman's dream to find out why. (Hint: make up a few polynomials over $\mathbb{F}_2$ and square them; make up a few polynomials over $\mathbb{F}_3$ and cube them. What do they look like? Looking at only the pth powers, is it easy to guess what they are the pth powers of?)

(Now, if f is a pth power, its pth root might also be a pth power: e.g. if $f(x) = x^4 + 1$ over $\mathbb{F}_2$. So, you need to recursively run the pth root of f through the squarefree-preparation routine.)

Case 3: Else — i.e. if g has degree ≥ 1, and if $f' \neq 0$, then recursively apply the squarefree-preparation algorithm to g and f/g.

2 Practicalities

2.1 Pseudocode

Question: Use the pieces of theory from the above sections to write out a pseudocode description of Berlekamp's algorithm. (You may assume the input is squarefree, or you may incorporate the information in section 1.4 and describe how non-squarefree input is handled.) The algorithm should take an input polynomial, and describe how to factor it into distinct irreducible factors. Your description should include what can happen when there are three or more factors. You need not describe row reduction in detail: you may treat this as a black-box step of the algorithm. However, you should be explicit about how each of the basis vectors are handled.

Extra question: What do you choose to do if the input polynomial is non-monic?

2.2 Worked example

Given squarefree $f(x)$, we want to find polynomials $h(x)$ such that $h^q \equiv h \pmod{f}$. Take $f = x^5 + x^4 + 1 = 110001 = 111 \cdot 1011$. The degree of f is 5. By explicit search, we can find the following polynomials h of degree < 5 such that $h^2 \equiv h \pmod{f}$: 0, 1, 11100, 11101.

To avoid having to do a search, we use linear algebra instead.

$$\begin{aligned} f &= x^5 + x^4 + 1 = 110001 \\ x^0 &\equiv 00001 \pmod f \\ x^2 &\equiv 00100 \pmod f \\ x^4 &\equiv 10000 \pmod f \\ x^6 &\equiv 10011 \pmod f \\ x^8 &\equiv 11111 \pmod f. \end{aligned}$$

$$\begin{aligned} h &= a_4x^4 + a_3x^3 + a_2x^2 + a_1x + a_0 \\ h^2 &= a_4x^8 + a_3x^6 + a_2x^4 + a_1x^2 + a_0 \\ &= a_4(x^4 + x^3 + x^2 + x + 1) + a_3(x^4 + x + 1) + a_2(x^4) + a_1(x^2) + a_0(1) \end{aligned}$$

$$\begin{bmatrix} a_4 \\ a_3 \\ a_2 \\ a_1 \\ a_0 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} a_4 \\ a_3 \\ a_2 \\ a_1 \\ a_0 \end{bmatrix}$$

$$\begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} a_4 \\ a_3 \\ a_2 \\ a_1 \\ a_0 \end{bmatrix}.$$

Call the matrix M. Its $(n-1-j)$th column is $x^{jq} \pmod f$. Put $M - I$ in row-echelon form to obtain

$$\begin{bmatrix} 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

with kernel basis

$$(1, 1, 1, 0, 0), \qquad (0, 0, 0, 0, 1).$$

These are $h_1 = 11100$ and $h_2 = 1$, respectively. Compute $\gcd(f, h_1) = 111$ and $\gcd(f, h_1 + 1) = 1011$ to obtain non-trivial factors of f.

2.3 Complexity analysis

Question: Describe the number of $\mathbb{F}_q$ arithmetic operations needed to perform one step of Berlekamp's algorithm, i.e. to find one non-trivial factor of an input polynomial $f(x)$. (For row reduction, you can count operations. For the GCD step, describe the complexity of a single GCD, since the number of GCDs that need to be computed may vary.)

A Background: integers

I assume you're familiar with integer arithmetic, but perhaps not so much with polynomials over $\mathbb{F}_q$. Moreover, even if you have had exposure to finite fields, my experience is that many instructors teach the subject very abstractly. So, you might have been taught existence and uniqueness of finite fields, but you might never have been taught how to actually do explicit computations with them.

Here we remind ourselves of some facts about the familiar integers, in order to facilitate comparison with perhaps-less-familiar polynomials over finite fields, below.

A.1 Integer factorization and complexity

How do you factor an integer n into primes? You don't need to find all prime factors at once. It suffices to find any non-trivial factor a. If there is no such, then n is prime. Otherwise you get $b = n/a$. Then, using a divide-and-conquer strategy, you can apply whatever algorithm you used for n to the integers a and b.

So, how do we find a non-trivial factor of a given n? One way is to trial-divide n by all $2 \leq a < n$. If the remainder is zero for any such a, then $a \mid n$. What is the computational complexity? This takes $n - 2$ divisions; we say that this algorithm is $O(n)$ in the number of divisions. (Note of course that division itself takes longer when n is bigger; you would get a different order of complexity if you were to count not number of divisions, but say, digit operations.)

This is clearly non-optimal, though. If $a \mid n$, then $b = n/a \mid n$ as well, and we can take $a \leq b$. That is, if n has one factor greater than or equal to $\sqrt{n}$, then it has another factor less than or equal to $\sqrt{n}$. Given $n = 101$, once you've gotten past $a = 10$ you can stop. Thus, a second algorithm for finding a non-trivial factor of n is to trial-divide by all $2 \leq a \leq \sqrt{n}$. This takes $O(\sqrt{n})$ trial divisions.

This is also non-optimal. Once you've found that $2 \nmid n$, it's silly to divide n by 4, 6, or any other even number. Likewise, once you know $3 \nmid n$, you needn't trial-divide n by any other multiple of 3. So, a third algorithm would be to trial-divide n by all $2 \leq p \leq \sqrt{n}$ only for primes p. (Of course, you can only do this if you have a big enough table of primes.) What's the complexity? By the prime number theorem, the number of primes less than or equal to $\sqrt{n}$ is approximately $\sqrt{n}/\ln(\sqrt{n})$, so the number of trial divisions is $O(\sqrt{n}/\ln(\sqrt{n}))$.

There are lots of integer-factorization algorithms, some easy and some hard, and also there are quite nice primality-testing algorithms which don't produce a factorization but which tell you if you should bother. I won't describe them further; the above examples suffice to demonstrate that (a) there are different ways of doing things, and (b) one can quantify the amount of time they take.

A.2 Modular arithmetic with integers

Given any $n \geq 2$, we can compute in the residue ring $\mathbb{Z}/n\mathbb{Z} = \mathbb{Z}/\langle n \rangle$. (I happily write this $\mathbb{Z}_n$, although number theorists balk at this notation since it means something else to them.) The method is to reduce mod n and take remainders. E.g. if $n = 10$ and $a = 27$, we reduce 27 mod 10 to obtain the canonical representative 7 for the equivalence class containing 7. Specifically, we divide 27 by 10 and take the remainder, 7.

Given equivalence-class representatives a and b, we can add, subtract, or multiply them in $\mathbb{Z}$, then reduce mod n. This is because reduction mod n is a ring homomorphism from $\mathbb{Z}$ to $\mathbb{Z}_n$.

If n is prime (call it p), then $\mathbb{Z}_p$ is a field. This is how we construct finite fields of prime order p. Otherwise,

$\mathbb{Z}_n$ is a ring with zero-divisors.

B Background: polynomial and modular polynomial arithmetic over $\mathbb{F}_2$

This project studies factorizations of polynomials in $\mathbb{F}_q[x]$. Since this may be less familiar than working with integers, we give some computational examples and some notation.

B.1 Polynomial arithmetic

Take $q = 2$. Then $x^4 + 1$ is a fourth-degree polynomial with coefficients 0 or 1, with coefficient arithmetic done mod 2. This factors as $(x^2 + 1)^2$, which in turn factors as $(x + 1)^4$. So, $x^4 + 1$ is *reducible*.

By contrast, $f(x) = x^4 + x + 1$ has no non-trivial factors; we say that it is *irreducible*. How can I be so sure of that statement? Well, since degree four is small, we can do an explicit search. If f has a non-trivial factor, i.e. a factor with degree greater than or equal to one, one factor must have degree 1, 2, or 3. There are two degree-one polynomials over $\mathbb{F}_2$: x and $x + 1$. There are four degree-two polynomials: x^2, $x^2 + 1$, $x^2 + x$, and $x^2 + x + 1$. There are eight degree-three polynomials, x^3 through $x^3 + x^2 + x + 1$. So, all we need to do is use long division, dividing $x^4 + x + 1$ by each of these fourteen possible divisors. If none of them goes into $x^4 + x + 1$ with a zero remainder, then $x^4 + x + 1$ is irreducible.

This is similar to the most naive trial division for integers. As with integer division, where we need to trial-divide up to $\sqrt{n}$, here too we can stop trial-dividing by polynomials with degree up to $n/2$ where $n = \deg f$. For larger degree, the number of trial divisors gets large.

Question: What is the number of trial divisors using the above method? You can write it as a function of n and q.

Elwyn Berlekamp's algorithm (1967) is a powerful way to factor polynomials over $\mathbb{F}_q$. Note that we don't have such a powerful algorithm for polynomials with coefficients in $\mathbb{Q}$, or for integers.

For shorthand, we'll often write polynomials with their coefficients only, in descending order of degrees of terms. For example, $x^4 + 1 = 10001 = 101^2 = 11^4$ and $x^4 + x + 1 = 10011$.

You can check that

$$110001 = 111 \cdot 1011$$

and

$$111100001110 = 10 \cdot 11 \cdot 111 \cdot 1011 \cdot 1101$$

Likewise you can check that each of these factors is in fact irreducible. (The ring $F[x]$, for any field F, is a unique factorization domain so irreducibility and primality are equivalent. Nonetheless, the convention is to use the words "prime" and "composite" for integers, and the words "irreducible" and "reducible" for polynomials.)

B.2 Modular polynomial arithmetic

Once we know how to add, subtract, and multiply polynomials in $\mathbb{F}_q[x]$, it's not much harder to do arithmetic in the residue ring $\mathbb{F}_q[x]/\langle f \rangle$. Just as with modular arithmetic of integers, given a polynomial $a(x)$, we divide

it by the modulus $f(x)$ and take the remainder. This means in particular that $\deg a < \deg f$, where a here refers to the canonical representative.

Question: Given $f(x) \in \mathbb{F}_q[x]$, with $\deg f = n$, how many elements are there in $\mathbb{F}_q[x]/\langle f \rangle$?

Given two polynomials $a(x)$ and $b(x)$, we can add, subtract, or multiply them, then reduce the result mod $f(x)$. For example, if $f(x) = 10011 \in \mathbb{F}_2[x]$, $a(x) = 1111$, and $b(x) = 101$, then $a(x)b(x)$ can be found using high-school multiplication (with coefficients mod 2, i.e. you can just forget to carry):

```
  1111
*  101
------
  1111
1111
------
110011
```

Now to reduce mod $f(x)$, you can do long division and keep the remainder. But, since *all* you need is the remainder and not the quotient, it's even easier. Just keep subtracting off multiples of $f(x)$ until the result has degree less than the degree of $f(x)$:

```
  110011
- 10011
  ------
   10101
 - 10011
  ------
     110
```

So, just as $27 \cdot 36 \equiv 2 \pmod{10}$ in the integers, we have $1111 \cdot 101 \equiv 110 \pmod{10011}$ in $\mathbb{F}_2[x]$.

If $f(x)$ is irreducible in $\mathbb{F}_q[x]$, then $\mathbb{F}_q[x]/f(x)$ is a field — a finite field of size q^n where $n = \deg(f)$. Otherwise, it's a ring with zero-divisors.

References

[**DF**] D.S. Dummit and R.M. Foote. *Abstract Algebra* (2nd ed.). John Wiley and Sons, 1999.

[**PARI**] H. Cohen, K. Belebas et al. PARI/GP. `pari.math.u-bordeaux.fr`.

[**Knu**] D.E. Knuth. *The Art of Computer Programming, Vol. 1: Fundamental Algorithms* (2nd ed.). Addison-Wesley, 1973.

[**Knu**] D.E. Knuth. *The Art of Computer Programming, Vol. 2: Seminumerical Algorithms* (2nd ed.). Addison-Wesley, 1973.

[**LN**] R. Lidl and H. Niederreiter. *Finite Fields*. Cambridge University Press, 1997.