Math 75 notes, Lecture 24

P. Pollack and C. Pomerance

A terrible algorithm

Suppose that $f(x) \in \mathbb{F}_p[x]$ has degree d and we wish to determine if $f(x)$ is irreducible or not. Here is an algorithm that works: Trial divide $f(x)$ with every non-constant, monic polynomial of degree at most $d/2$. This works, because if $f(x)$ is reducible, an irreducible factor of least degree must have degree $\leq d/2$. The algorithm has a bonus since if $f(x)$ is reducible, the algorithm will find a nontrivial factorization of $f(x)$.

So, why is it a terrible algorithm? The answer lies in the number of steps in the worst case. If f is irreducible, the number of trial divisions attempted is

$$p + p^2 + \cdots + p^{\lfloor d/2 \rfloor},$$

since there are exactly p^j monic polynomials of degree j. As a function of d, this expression grows exponentially. Worse, the dependence on p is also nasty, even for fixed d. Indeed, members of $\mathbb{F}_p$ are integers in the range $[0, p-1]$ and a number in this range can be described by its digits, so a description of a number in this range has length proportional to $\log p$, or smaller. A "good" algorithm would have a bound for the number of steps that only involves constant powers of $\log p$.

So, we take this as our ideal of a good algorithm. The number of steps should be no more than a fixed power of $\log p$ times a fixed power of d, the degree of the polynomial. So, the perfectly legitimate trial division algorithm is indeed terrible, since it fails on both counts.

Some good algorithms

First note that ordinary arithmetic in $\mathbb{F}_p$ is fine. The naive algorithms that we know are indeed good. For example, take addition. Two members of $\mathbb{F}_p$ have at most a constant times $\log p$ digits, so adding them using elementary-school arithmetic takes at most a constant time $\log p$ steps. If the answer is bigger than p, there is one final step of subtracting p, but the elementary-school algorithm is just fine here too, so we can see that in all, no more than a constant times $\log p$ steps are involved.

Not only is addition and subtraction good, but so too is multiplication. Here the usual "parallelogram" algorithm (named for the shape that the partial products form in the elementary-school algorithm) takes at most a constant times $(\log p)^2$ steps. In $\mathbb{F}_p$ one has the additional step of dividing by p and taking a remainder, but this too throws in no more than an additional constant times $(\log p)^2$ steps.

In addition to addition, subtraction, and multiplication, we can ask about division in $\mathbb{F}_p$. This involves finding the multiplicative inverse, and then multiplying by that inverse. For example, if you wish to divide by 5 in $\mathbb{F}_{17}$, this is the same as multiplying by 7, since $5 \cdot 7 = 1$ in this field. And we have learned that finding the inverse involves the extended Euclid algorithm for gcd. In particular, in following the extended gcd method for 5 and 17, one finds that

$7 \cdot 5 - 2 \cdot 17 = 1$, so that we can read off the inverse of 5. Now Euclid's algorithm involves a number of divisions with remainder, each of which is no problem, but we might be unhappy if the number of them needed is huge. In fact, it is possible to show that the number of divisions is bounded by a constant times the log of the smaller number. In particular, this tells us that we have a good algorithm for computing inverses in $\mathbb{F}_p$ and so for doing dvision in this field.

We also have good algorithms for doing arithmetic in $\mathbb{F}_p[x]$. Addition of two polynomials of degree at most d takes at most a constant times $d \log p$ elementary steps, which fits into our definition of good. Mutliplication, by the school method of multiplying polynomials, is also good, the time being bounded by a constant times $d^2(\log p)^2$. Moreover, gcd and extended gcd is also a good algorithm, since the number of divisions is at most the smaller of the degrees of the two polynomials involved.

Finally note that our latest trick of dealing with the formal derivative is no problem from an algorithmic standpoint. Computing $D(f)$ from f is also a good algorithm.

A good application

We can now use the above tools to write down a very simple algorithm that detects whether a given polynomial is squarefree and gives a nontrivial factorization of it if it is not squarefree. Here's the algorithm: Given a polynomial $f(x)$ in $\mathbb{F}_p[x]$ of degree at least 2, form $G(x) = \gcd(f(x), D(f(x)))$. If $G(x) = 1$, then $f(x)$ is squarefree. If the gcd is not 1, then $f(x)$ is not squarefree. Moreover, if $0 < \deg G(x) < \deg f(x)$, then $G(x)$ is a nontrivial factor of $f(x)$. And if $\deg G(x) = \deg f(x)$, then $f(x) = u(x^p)$ for some polynomial $u(x) \in \mathbb{F}_p[x]$, and so $f(x) = u(x)^p$.

We have already seen that these assertions are perfectly correct. The new thought now is that it is easy to compute $G(x)$, which is the key for detecting whether $f(x)$ is squarefree. We indeed have a good algorithm.

Making the terrible algorithm good

We saw above that individually trial dividing by every monic polynomial of degree j is a terrible way to see if $f(x)$ has an irreducible factor of degree j. Is there some way we can test all of the degree j's at the same time? Well yes, there is. Recall that the polynomial $x^{p^j} - x$ is the product of all of the irreducible polynomials in $\mathbb{F}_p[x]$ of degree dividing j. So, in particular, if $\gcd(x^{p^j} - x, f(x)) = 1$, then $f(x)$ has no irreducible factors of degree dividing j.

This is very promising, because we have seen that Euclid has given us a good algorithm for computing gcd. But there is a problem. To get the gcd started, it would seem we need to divide $f(x)$ into the polynomial $x^{p^j} - x$ to get the first remainder. And the time for this would seem to involve the degree of $x^{p^j} - x$, which is the exponentially large p^j.

But there is a faster way of getting up to this high power. Suppose $r_j(x)$ is the remainder when $f(x)$ is divided into x^{p^j}. Then we can build up to $r_j(x)$ one step at a time. We have $r_1(x) = x$ (assuming that the degree of $f(x)$ is at least 2). Then $r_2(x)$ is the remainder when $f(x)$ is divided into $r_1(x)^p$, and so one until we find that $r_j(x)$ is the remainder when $f(x)$ is divided into $r_{j-1}(x)^p$.

So, if $p = 2$, this is very easy. For example, to figure the remainder when $f(x)$ is divided into $x^{2^{100}}$, there are just 100 squarings of polynomials of degree $< \deg f(x)$, followed by dividing by $f(x)$ to get the remainder.

If $p = 3$, it would be cubings, which can be done in two steps. How many steps to raise to the 5th power? It can be done in 3 steps, by squaring twice and then multiplying by the original. This idea scales nicely. Say you want to raise to the 101st power. Compute

$$r_2(x) = r(x)^2 \bmod f(x), \quad r_4(x) = r_2(x)^2 \bmod f(x), \ \ldots \ , r_{64}(x) = r_{32}(x)^2 \bmod f(x),$$

taking just 6 steps. Now write 101 in binary, it is $64 + 32 + 4 + 1$, so we have

$$\begin{aligned} r_{96}(x) &= r_{64}(x) r_{32}(x) \bmod f(x), \\ r_{100}(x) &= r_{96}(x) r_4(x) \bmod f(x), \\ r_{101}(x) &= r_{100}(x) r(x) \bmod f(x). \end{aligned}$$

Thus, raising to the power 101, modulo a polynomial $f(x)$ of degree d, can be done in 9 arithmetic steps with polynomials of degree $< 2d$.

In general this repeated squaring algorithm can raise to the pth power modulo $f(x)$ in at most a constant times $\log p$ arithmetic steps with polynomials of degree $< 2d$, where $d = \deg f(x)$. So, it is a good algorithm. And so, the first step of the gcd: we can first find $x^{p^j} \bmod f(x)$, and then subtract x from this giving us the remainder when $f(x)$ is divided into $x^{p^j} - x$. Moreover, if we had previously dealt with $x^{p^{j-1}} - x$, then we would have had previously the remainder $r_{j-1}(x)$ when $f(x)$ is divided into $x^{p^{j-1}}$, and so we can arrive at $r_j(x)$ without starting from scratch, namely, $r_j(x) = r_{j-1}(x)^p \bmod f(x)$.

This is a good algorithm for irreducibility testing that has a bonus towards factoring. Say you are given $f(x) \in \mathbb{F}_p[x]$ of degree d, and that $f(x)$ is squarefree. Let $f_1(x) = \gcd(x^p - x, f(x))$. Let $f_2(x) = \gcd(x^{p^2} - x, f(x)/f_1(x))$, and continue finding $f_j(x)$ for $j \le d/2$. Here each $f_j(x)$ is the product of the distinct irreducible factors of $f(x)$ of degree j. If all of these polynomials are 1, then f is irreducible. If any of these polynomials are not 1, then f is reducible, and we know something about the degrees of the irreducible factors of $f(x)$.

A final splitting

So, we can easily detect nonsquarefree polynomials, and split those that are not. We can easily detect irreducible polynomials. And we can easily split squarefree polynomials that have irreducible factors of different degrees. What's left? The answer, polynomials that are squarefree where all of the irreducible factors have the same degree.

Here's an example. We know that 2 is a square in $\mathbb{F}_{103}$. Indeed, one way to know this is by using the fact from elementary number theory that 2 is a square modulo the odd prime p precisely when p is 1 or 7 mod 8. Another way to detect squares in a finite field of odd order comes from the existence of a primitive element, as discussed on the last test. If $\beta \in \mathbb{F}_{p^k}^\times$, then β is a square if and only if $\beta^{(p^k-1)/2} = 1$. So we have a method contained in this course of checking that 2 is a square in $\mathbb{F}_{103}$, namely checking that $2^{51} \equiv 1 \pmod{103}$.

Good, 2 is a square in $\mathbb{F}_{103}$. This implies that $x^2 - 2 \in \mathbb{F}_{103}[x]$ is reducible. So, factor it! It would be the product of two degree 1's, so it is in exactly the case described above, namely a squarefree reducible polynomial where each irreducible factor has the same degree. That is, a method of polynomial factorization would have the application then of finding square roots of squares in $\mathbb{F}_p$.

We now describe a general algorithm for factoring such a polynomial. It is a good algorithm, but not in the sense of the algorithms discussed so far. What's unusual about the algorithm is that it involves random choices. In fact, we'll see that the with a random choice being made, the algorithm has a nice and small number of steps, but only a 50% shot at producing a nontrivial factorization. So, the idea is to repeat the process, and we expect to get a random choice that works sooner than later.

A random algorithm

We look first at the case that p is an odd prime. And to keep the description simple, we suppose that our polynomial $f(x)$ that we wish to factor is known to be the product of two different monic irreducible polynomials of degree j. Say these two are $h_1(x), h_2(x)$. Giving them names does not necessarily mean that we know what they are; it is our task to find them. Here is our algorithm: Randomly choose a polynomial $g(x)$ of degree $< 2j$. Then

(i) Check if $\gcd(g(x), f(x))$ is a nontrivial factor of f. If so, we're done.

(ii) Check if $\gcd(g(x)^{(p^j-1)/2} - 1, f(x))$ is a nontrivial factor of f. If so, we're done.

We claim that at least half of all possible choices of $g(x)$ in this algorithm will uncover the factorization of $f(x)$.

Let's see how often we succeed in step (i). As you'll show in homework, we fail to uncover a factorization in step (i) for exactly $(p^j - 1)^2$ choices of g, and so we succeed for

$$p^{2j} - 1 - (p^j - 1)^2 = 2p^j - 2$$

choices. (Notice that $\deg \gcd(g(x), f(x)) \le \deg g(x) < 2j = \deg f(x)$, so that either $g(x)$ and $f(x)$ are coprime, or $\gcd(g(x), f(x))$ is a nontrivial factor of $f(x)$.) Ok, what about step (ii)? This requires a more detailed analysis.

With an irreducible polynomial $h(x) \in \mathbb{F}_p[x]$ of degree j, we may construct the finite field $\mathbb{F}_{p^j}[x]$ in the usual way as $\mathbb{F}_p[x]/(h(x))$. In particular, nonzero elements of $\mathbb{F}_{p^j}$ are represented by polynomials $r(x)$ of degree $< j$, where $r(x)$ actually is just the representative of the coset $r(x) + (h(x))$. Now elements of $\mathbb{F}_{p^j}^\times$ are either squares or not, with half of them squares and half not (here's where we use that p is odd). The criterion for an element β to be a square is that $\beta^{(p^j-1)/2} = 1$, while the criterion for being a nonsquare is that $\beta^{(p^j-1)/2} = -1$.

It follows that with $h(x)$ denoting either of the polynomials $h_1(x)$ or $h_2(x)$, precisely half of all the polynomials $r(x)$ of degree $< j$ satisfy

$$r(x)^{\frac{p^j-1}{2}} \equiv 1 \pmod{h(x)}, \quad \text{and the other half satisfy} \quad r(x)^{\frac{p^j-1}{2}} \equiv -1 \pmod{h(x)},$$

according as $r(x)$ is or is not a square modulo $h(x)$.

We have two realizations of $\mathbb{F}_{p^j}$, one via $h_1(x)$, the other via $h_2(x)$. We choose polynomials $r_1(x), r_2(x)$ both of degree $< d$ so that $r_1(x)$ is a square modulo $h_1(x)$, but $r_2(x)$ is not a square modulo $h_2(x)$, or vice versa. The number of ways of choosing $r_1(x), r_2(x)$ like this is precisely

$$\frac{p^j-1}{2}\frac{p^j-1}{2}+\frac{p^j-1}{2}\frac{p^j-1}{2}=\frac{1}{2}(p^j-1)^2.$$

We now construct a polynomial $g(x)$ of degree $< 2j$ satisfying

$$g(x)\equiv r_1(x)\pmod{h_1(x)}\quad\text{and}\quad g(x)\equiv r_2(x)\pmod{h_2(x)}.$$

Since $h_1(x), h_2(x)$ are relatively prime in $\mathbb{F}_p[x]$, there are polynomials $v_1(x), v_2(x)$ with

$$v_1(x)h_1(x)+v_2(x)h_2(x)=1.$$

Consider the polynomial

$$w(x)=r_2(x)v_1(x)h_1(x)+r_1(x)v_2(x)h_2(x)$$

and notice that

$$w(x)\equiv r_1(x)\pmod{h_1(x)}\ \text{ and }\ w(x)\equiv r_2(x)\pmod{h_2(x)}.$$

If we reduce $w(x)$ modulo $h_1(x)h_2(x)$ it will have degree $< 2j$ and will still satisfy the same two congruences. This reduced polynomial is our $g(x)$.

We claim that all of the $\frac{1}{2}(p^j-1)^2$ polynomials $g(x)$ constructed this way succeed in step (ii) of our algorithm in factoring f. For example, suppose that $r_1(x)$ is a square modulo $h_1(x)$ and $r_2(x)$ a nonsquare modulo $h_2(x)$. Then

$$g(x)^{\frac{p^j-1}{2}}\equiv r_1(x)^{\frac{p^j-1}{2}}\equiv 1\pmod{h_1(x)}\quad\text{while}\quad g(x)^{\frac{p^j-1}{2}}\equiv r_2(x)^{\frac{p^j-1}{2}}\equiv -1\pmod{h_1(x)}.$$

It follows that

$$\gcd(g(x)^{(p^j-1)/2}-1, f(x))=h_1(x).$$

If instead $r_1(x)$ is a nonsquare modulo $h_1(x)$ but $r_2(x)$ is a square modulo $h_2(x)$, then this gcd will pick out the factor $h_2(x)$.

So the number of choices of $g(x)$ of degree $< 2j$ for which out algorithm succeeds in splitting f is at least

$$2p^j-2+\frac{1}{2}(p^j-1)^2,$$

and it is a simple exercise in inequalities to show that this is at least half the total number of polynomials of degree $< 2j$ (i.e., at least $\frac{1}{2}(p^{2j}-1)$).

An example

Let us illustrate this algorithm for the polynomial $f(x) = x^2 + 2$ in $\mathbb{F}_{17}[x]$. Say we have already discovered that $x^2 + 2$ is not irreducible. This is not so hard and can be seen mentally as follows. First, it is clearly squarefree, since it is coprime to its derivative $2x$. Next, $(-2)^4 = 16 = -1$ in $\mathbb{F}_{17}$, so $(-2)^8 = 1$, which shows that -2 is a square. Thus, $x^2 + 2$ is reducible, it can be factored as a difference of two squares. To do this though, we would need to find a squareroot of -2. We're in the case of $j = 1$ in the above discussion, so let's choose a random monic polynomial $g(x)$ of degree $< 2j = 2$. OK, say $g(x) = x - 5$. We need to compute $(x - 5)^8 \bmod f(x)$. First, we have

$$(x-5)^2 = x^2 - 10x + 25 = -2 - 10x + 25 = 7x + 6.$$

Next, we have

$$(x-5)^4 = (7x+6)^2 = 49x^2 + 84x + 36 = -2x^2 - x + 2 = -x + 6.$$

And then we have

$$(x-5)^8 = (-x+6)^2 = x^2 - 12x + 36 = -12x = 5x.$$

So we need to check the gcd of $5x - 1$ and $f(x) = x^2 + 2$. It's easier to do this if we make $5x - 1$ monic, and this entails multiplying it by $5^{-1} = 7$. So we need to check $\gcd(x - 7, x^2 + 2)$. But $x - 7$ is a divisor, and we find that

$$x^2 + 2 = (x-7)(x+7).$$

We also find that the squareroots of -2 are ± 7.

What to do when $p = 2$

What made the above idea work is the factorization

$$x^{p^j} - x = x\left(x^{(p^j-1)/2} - 1\right)\left(x^{(p^j-1)/2} + 1\right),$$

which strongly uses that p is odd. However there is an alternate factorization when $p = 2$. Let

$$T(x) = x + x^2 + \cdots + x^{2^{j-1}}.$$

Note that

$$T(x)^2 = x^2 + x^4 + \cdots + x^{2^j},$$

so that

$$T(x)^2 + T(x) = x + x^{2^j}.$$

That is, we have the factorization

$$x^{2^j} - x = x^{2^j} + x = T(x)(T(x)+1).$$

This leads to a factorization algorithm as follows. Take $g(x)$ at random of degree $< 2j$. Then compute $\gcd(T(g(x)), f(x))$. If f is squarefree, reducible, and with all irreducible factors of degree j, then this gcd will be a nontrivial factor of $f(x)$ with probability at least $1/2$.

Much of the above material is taken from *The art of computer programming*, volume 2, Seminumerical algorithms, by Donald Knuth.