CS681	Computational Number Theory

Lecture 10: Distinct Degree Factoring

Instructor: Piyush P Kurur *Scribe: Ramprasad Saptharishi*

Overview

Last class we left of with a glimpse into distant degree factorization. This class, we shall look at the details of it and also how it can be used as an irreducibility test.

1 DDF: The Problem

We are given a polynomial f over a finite field $\mathbb{F}_p$ of degree say n. We want to factor them in to degree 1 factors, degree 2 factors etc. To make this more explicit, let us assume that f factorizes as

$$f = g_{11}g_{12}\cdots g_{1m_1}g_{21}\cdots g_{dm_d}$$

where each g_{ij} is an irreducible factor of f of degree i. Hence given f, we want to return $\prod_j g_{ij}$ for each i. That is, return the factor of f that is the product of all degree i irreducible factors of f. And we want to do this for all i. This is called *distinct degree factorization*. We want an efficient algorithm for this.

But before we start thinking of algorithms, what do we mean by efficient? It is the usual 'polynomial time in input length' but what is the input length? We are looking at $f(X) = \mathbb{F}_p[X]$ and each coeffiecient of the polynomial is from $\mathbb{F}_p$. And since $\mathbb{F}_p$ contains just p elements, we can encode them in binary using $\log p$ bits. Hence the input size is about $n \log p$. Thus we are interested in algorithms that have running time of $(n \log p)^c$ for some constant c.

2 Extracting Square-free Parts

When we said that f factorizes as $g_{11}\cdots g_{dm_d}$, it is very much possible that there are some $gij = g_{ik}$ or in other words the square of g_{ij} divides f. The first step of any factoring algorithm is to remove such multiple factors and

make f *square free* (make sure that f is not divisible by any square).

Suppose we were looking at polynomials over $\mathbb{Z}[X]$ or something. Then we have a very nice way of checking for such multiple factors. We know that if f has a repeated root α , then the derivative f' has α as a root as well. Infact, if $(x-\alpha)^m$ divided f then $(x-\alpha)^{m-1}$ will divide f'. And hence, the gcd of f and f' will have $(x-\alpha)^{m-1}$ as a factor. Thus, we can divide f by this gcd and get rid of higher multiplicity terms.

But in the case of $\mathbb{Z}[X]$, we had an interpretation of real numbers, limits and hence we could define what a derivative is. But when it comes to finite fields, what does differentiation mean? How can we use this technique to get extract the square-free part?

Note that we don't need the notion of a derivative as a tangent or something. That is where the limits come in. What we need is a condition where if some g^m divides f then g^{m-1} should divide f' for the f' that we are going to define. Thus, since we are just in the realms of polynomials, we shall use the rules of differentiation as just a formula.

Thus let D be a map that sends X^m to mX^{m-1}. Now extend this linearly to all polynomials to get

$$D\left(a_0 + a_1X + a_2X^2 + \cdots a_mX^m\right) = a_1 + 2a_2X + \cdots ma_mX^{m-1}$$

Now, we leave the following things to be proven as an exercise.

Exercise

1. $D(f+g) = D(f) + D(g)$
2. $D(fg) = fD(g) + gD(f)$
3. $D(f^m) = mf^{m-1}D(f)$
4. **Theorem:** If h is a factor of f such that $h^m \mid f$ then $h^{m-1} \mid g$. And further, if h^m is the highest power of h that divides f then h^{m-1} is the highest power of h that divides f'.

Once we have these properties, we can extract the square-free part of f easily. Given an f construct the formal derivative f'. Let $g = gcd(f, f')$. The polynomial f/g consists is the square free extraction of f.

There is, however, a small catch here. But we shall get back to it in the end of the class and discuss it in the next class in detail. For the moment, let us proceed with distinct degree factorization.

3 Distinct Degree Factorization

Given $f \in \mathbb{F}_p[X]$ of degree n, we want to get the distant degree factorization of f. That is, we want to get $g_1, g_2, \cdots, g_n$ such that $f = g_1 g_2 \cdots g_n$ where each g is the product of all irreducible factors of f of degree i. And we want to do this efficiently.

The key idea is the formula we proved last class:

$$X^{p^m} - X = \prod_{\substack{f \in \text{Irr}(\mathbb{F}_p, d) \\ d|m}} f(X)$$

Firstly, let us look at the case when $m = 1$. Then $X^p - 1$ is the product of all irreducible polynomials over $\mathbb{F}_p$ of degree 1. And thus, in particular, it contains the degree 1 irreducible factors of f within it.

Thus, $g_1 = gcd(f, X^p - X)$ will be the product of all degree 1 irreducible factors of f as $X^p - X$ has just degree 1 irreducible factors and all those that divide f will be a part of the gcd. Thus, we have obtained the required g_1. Now, call $f_2 = f/g_1$ and now let $g_2 = gcd(f_2, X^{p^2} - X)$ and this will have precisely all degree 2 irreducible factors of f (degree 1 factors won't appear since we have removed all degree 1 factors by dividing by g_1). Thus, we can repeat this procedure.

This is a naive algorithm and has a pretty serious problem. The trouble is that the algorithm would have to compute $gcd(f_i, X^{p^i} - X)$ which is a HUGE degree polynomial. We want our running time as $(n \log p)^c$ but this naive algorithm takes about $O(p)$ time! And this is definitely not acceptable since even finding all linear irreducible factors might take $O(p)$ time! We definitely need to change this. But the polynomial $X^{p^i} - X$ is a nice polynomial and hence allows a nice simple trick to make the algorithm polynomial time.

Let us look at the gcd algorithm. The first step is to compute $X^{p^i} - X \bmod f_i$ and the problem with this is that the degree of the polynomial is too large to apply the naive algorithm. But we can do far better in the case of this special polynomial.

Note that $X^{p^i} - X \bmod f_i = X^{p^i} \bmod f_i - X \bmod f_i$ and the difficulty was in computing $X^{p^i} \bmod f_i$. This can be done quite efficiently using the technique of repeated squaring.

3.1 Repeated Squaring

In general, we have a polynomial f of degree n and we want to calculate $X^M \bmod f$ in time $(n \log M)^c$ for some constant c. The idea is pretty simple.

Firstly assume M to be a power of 2. Then we can do the following. Start with X, and square it. And square it again and so on. The moment the degree goes beyond n, take it modulo f to reduce it's degree back to less than n. And continue this squaring process, for $\log M$ steps. Thus, we would have computed $X^M \bmod f$ in time $(n \log M)^c$.

Now what do we do for M that is not a power of two? The answer is simple again. Just look at the binary representation of M. Say $m = b_0 + b_1 2 + b_2 2^2 + \cdots b_l 2^l$. Then

$$X^M = X^{b_0}(X^{b_1})^{2^1}(X^{b_2})^{2^2} \cdots (X^{b_l})^{2^l}$$

And since each b_i is either 1 or 0, X^{b_i} is either 1 or X. Thus, we can just compute $X^{2^i} \bmod f$ for $0 \leq i \leq \log M$ and multiply all those residues that have b_i as 1.

Algorithm 1 EVALUATE $X^M \bmod f$ USING REPEATED SQUARING

1: Let $M = b_0 + b_1 2 + \cdots b_l 2^l$, the binary representation of M, where $l = \lfloor \log M \rfloor$.
2: $g_0 = 1$ and $g = b_0 X$.
3: **for** $i = 1$ to l **do**
4: $\quad g_i = (g_{i-1})^2 \bmod f$
5: $\quad$ **if** $b_i = 1$ **then**
6: $\quad\quad g = g \cdot g_i \bmod f$
7: $\quad$ **end if**
8: **end for**
9: **return** g

Now, with this repeated squaring algorithm, we can do distinct degree factorization. The algorithm is given at the end of the lecture.

4 A Catch and a Hint

First let us get to the catch that we had mentioned earlier. Our method for extracting the square free part of f involved taking the formal derivative and then the gcd of f with f'. But strange things can happen in a finite field. For example, let us take the polynomial $f(X) = X^{2p} - 3X^p + 5$. The formal derivative is $2pX^{2p-1} - 3pX^{p-1}$ which is 0 in $\mathbb{F}_p$! What do we do now?

It is easy to see that the derivative can become 0 if the only surviving terms of the polynomial have degree that's a multiple of a prime. Then this polynomial $f(X)$ can be thought of as $g(X^p)$ and we can go on to do our algorithm for g instead of f. In our example, $g(X) = X^2 - 3X + 5$ which is well behaved. All we need to do is get the factors of g and replace every occurance of X by X^p. And more over, since we are in $\mathbb{F}_p$, it is easy to check that $f(X) = g(X^p) = (g(X))^p$.

So much for the small catch. Now here is a hint. The algorithm for distinct degree factorization immediately gives us an algorithm to test if a given polynomial over $\mathbb{F}_p[X]$ is irreducible. The reduction is very straightforward and the students are encouraged to think about it. We shall discuss this in the next class.

Algorithm 2 DISTINCT DEGREE FACTORIZATION

Input: $f(X) \in \mathbb{F}_p[X]$ of degree n.

1: $f_0 = f$.
2: **for** $i = 1$ to n **do**
3: Using repeated squaring, compute $s_i = X^{p^i} \bmod f_{i-1}$
4: Compute $g_i = gcd(f_{i-1}, s_i - X)$.
5: $f_i = f_{i-1}/g_i$
6: **end for**
7: **return** $\{g_1, g_2, \cdots, g_n\}$.