5 Division of Polynomials and Newton Iteration

Let F be a field. Given two polynomials $a, b \in \mathsf{F}[x]$ there exist $q, r \in \mathsf{F}[x]$ such that $a = qb + r$ where $r = 0$ or $\deg r < \deg b$. We have seen that if $\deg a = n$ and $\deg b = m$, we can compute q, r with $O(nm)$ operations in F. Using the simplifying assumption that $m < n$, we can then say that we can do division with remainder with $O(n^2)$ operations (which is accurate when m and n are about the same size).

However, we have also seen that we can multiply polynomials of degree at most n with $O(n \log n)$ operations in F, assuming some special properties of F, or more generally with $O(n(\log n)(\log\log n))$ operations in F (which we didn't cover in class, but uses a generalization of the FFT-based method).

Is division with remainder really "harder" than multiplication? Or can we compute division with remainder with $O(n \log n)$ operations in F? Can we use the fast polynomial multiplication algorithm? The answer to these last two questions is "yes". Around 1972, Borodin and Moenck, Strassen, Sieveking, and Kung derived a division algorithm that cost $O(n(\log n)^2(\log\log n))$ field operations.

5.1 Division using Newton Iteration

Let

$$a = a_n x^n + a_{n-1}x^{n-1} + \cdots + a_1 x + a_0,$$
$$b = b_m x^m + b_{m-1}x^{m-1} + \cdots + b_1 x + b_0,$$

with $a_n, b_m \neq 0$, and assuming $m \leq n$. We wish to find $q \in \mathsf{F}[x]$ and $r \in \mathsf{F}[x]$ satisfying $a = qb + r$ with $r = 0$ or $\deg r < \deg b$. For convenience we will assume that $b_m = 1$ (why is this not a bad restriction?).

A useful property of polynomials is the ability to "reverse" them in an algebraically meaningful way. Substitute $\frac{1}{y}$ for x in $a(x)$ above, to obtain

$$a\left(\frac{1}{y}\right) = a_n \frac{1}{y^n} + a_{n-1}\frac{1}{y^{n-1}} + \cdots + a_1 \frac{1}{y} + a_0$$

Multiplying both sides by y^n, we obtain

$$y^n a\left(\frac{1}{y}\right) = a_n + a_{n-1}y + a_{n-2}y^2 + \cdots + a_1 y^{n-1} + a_0 y^n,$$

the *reversal* of a. For convenience, we define $\mathrm{rev}_k(a) = y^k \cdot a(1/y)$. When a is a degree k polynomial, this is the reversal of a (when a has degree less than k, we introduce extra multiples of y; when a has degree greater than k, we end up with denominators which are powers of y).

Now substitute $\frac{1}{y}$ for the variable x in the expression $a(x) = q(x)b(x) + r(x)$. We obtain

$$a\left(\frac{1}{y}\right) = q\left(\frac{1}{y}\right) \cdot b\left(\frac{1}{y}\right) + r\left(\frac{1}{y}\right).$$

Multiplying both sides by y^n we get

$$y^n a\left(\frac{1}{y}\right) = \left(y^{n-m} q\left(\frac{1}{y}\right)\right)\left(y^m b\left(\frac{1}{y}\right)\right) + y^{n-m+1}\left(y^{m-1} r\left(\frac{1}{y}\right)\right), \text{ or equivalently}$$
$$\mathrm{rev}_n(a) = \mathrm{rev}_{n-m}(q)\cdot \mathrm{rev}_m(b) + y^{n-m+1}\mathrm{rev}_{m-1}(r)$$

It is useful to consider the ring of Taylor series in y. Recall that this is the ring of all infinite sums

$$a_0 + a_1 y + a_2 y^2 + \cdots \quad \text{where } a_i \in \mathsf{F} \text{ for } i \geq 0,$$

which we write as $\mathsf{F}[[y]]$. All polynomials in $\mathsf{F}[y]$ can be thought of as Taylor series (with all the higher degree coefficients zero), but there are Taylor series which are not polynomials. We add and multiply Taylor series much like polynomials, though of course there are an infinite number of coefficients. One of the many useful facts about Taylor series is that if $u \in \mathsf{F}[[y]]$ has a non-zero constant coefficient, then there exists another Taylor series $\tilde{u} \in \mathsf{F}[[y]]$ such that $u\tilde{u} = 1$. That is, $\tilde{u} = u^{-1}$.

Since b is monic of degree m, we know that the constant coefficient of $\mathrm{rev}_m(b)$ is 1. Thus, $\mathrm{rev}_m(b)$ has an inverse in $\mathsf{F}[[y]]$. Suppose we could compute $\mathrm{rev}_m(b)^{-1} \in \mathsf{F}[[y]]$ exactly. Then we could compute q and r as follows. We have

$$\mathrm{rev}_n(a) = \mathrm{rev}_{n-m}(g)\cdot \mathrm{rev}_m(b) + y^{n-m+1}\mathrm{rev}_{m-1}(r).$$

Therefore,

$$\mathrm{rev}_n(a) \equiv \mathrm{rev}_{n-m}(q)\cdot \mathrm{rev}_m(b) \mod y^{n-m+1},$$

and

$$\mathrm{rev}_n(a)\cdot \mathrm{rev}_m(b)^{-1} \equiv \mathrm{rev}_{n-m}(q) \mod y^{n-m+1}.$$

We then have $q = \mathrm{rev}_{n-m}(\mathrm{rev}_{n-m}(q))$ and $r = a - q\cdot b$.

Example 5.1. *Let $a = x^3 + 2x^2 + x + 2$ and $b = x^2 + x + 2$ be polynomials in $\mathbb{Z}_3[x]$. Then*

$$\mathrm{rev}_3(a) = 2y^3 + y^2 + 2y + 1,$$
$$\mathrm{rev}_2(b) = 2y^2 + y + 1.$$

We claim $\mathrm{rev}_2(b)^{-1} \equiv 2y + 1 \bmod y^2$. *Check:*

$$\begin{aligned}(2y+1)(2y^2+y+1) &= 4y^3 + 4y^2 + 3y + 1\\ &\equiv 1 \bmod y^2.\end{aligned}$$

Now

$$\begin{aligned}\mathrm{rev}_1(q) &= (2y^3 + y^2 + 2y + 1)(2y+1) \bmod y^2\\ &= y + 1,\end{aligned}$$

so $q = x + 1$ and $r = a - qb = x$.

We now need a fast method to compute the inverse of $\mathrm{rev}_m(b)$ modulo y^{n-m+1}.

Problem 5.2. *Given $g \in \mathsf{F}[x]$ and $k \in \mathbb{N}$, find $h \in \mathsf{F}[x]$ of degree less than k satisfying $hg \equiv 1 \bmod x^k$.*

From mathematics and scientific computing courses recall that Newton iteration involves computing successive approximations to solutions of $f(t) = 0$, generally for some function $f : \mathbb{R} \to \mathbb{R}$. From a suitable initial approximation, t_0, subsequent approximations are computed using

$$t_{i+1} = t_i - \frac{f(t_i)}{f'(t_i)}.$$

For our problem, we need to find a zero of a function $\Phi : \mathsf{F}[[y]] \to \mathsf{F}[[y]]$, namely the function

$$\Phi(X) = \frac{1}{X} - g,$$

since $\Phi(\tilde{g}) = 0$ where $\tilde{g} \in \mathsf{F}[[y]]$ is such that $\tilde{g} \cdot g = 1$ (i.e., $\tilde{g}$ is the Taylor series inverse of g). Clearly

$$\Phi'(X) = -\frac{1}{X^2}$$

and our Newton iteration step is

$$h_{i+1} = h_i - \frac{\dfrac{1}{h_i} - g}{-1/h_i^2} = 2h_i - gh_i^2.$$

The following theorem tells us a good initial approximation and shows us that this method "converges" quickly to a solution.

Theorem 5.3. *Let $g, h_0, h_1, \ldots \in \mathsf{F}[x]$, with $h_0 = 1$ and*

$$h_{i+1} \equiv 2h_i - gh_i^2 \bmod x^{2^{i+1}},$$

for all i. Assume also that the constant coefficient of g is 1 (i.e., $g_0 = 1$). Then, for all i,

$$gh_i \equiv 1 \bmod x^{2^i}.$$

Proof. The proof is by induction on i. For $i = 0$ we have

$$gh_0 \equiv g_0 h_0 \equiv 1 \cdot 1 \equiv 1 \bmod x^{2^0}.$$

Assume $gh_i \equiv 1 \bmod x^{2^i}$, for some $i \geq 0$. Now

$$\begin{aligned} 1 - gh_{i+1} &\equiv 1 - g(2h_i - gh_i^2) \\ &\equiv 1 - 2gh_i + g^2h_i^2 \\ &\equiv (1 - gh_i)^2 \\ &\equiv 0 \bmod x^{2^{i+1}} \end{aligned}$$

We conclude that for all $i \in \mathbb{N}$ we have $gh_i \equiv 1 \bmod x^{2^i}$. □

We can now derive a complete algorithm to compute the inverse of $g \bmod x^k$.

```
Algorithm: InversePolyMod
Input:  ▸ $g = g_0 + g_1 x + \cdots + g_n x^n$ and $k \in \mathbb{N}$;
Output: ▸ $u \in \mathsf{F}[x]$ satisfying $1 - gu \equiv 0 \bmod x^k$.
(1) $h_0 = 1$, $r = \lceil \log_2 k \rceil$.
(2) for $i = 0, \ldots, r-1$ do calculate $h_{i+1} = (2h_i - gh_i^2) \operatorname{rem} x^{2^i}$
(3) return $h_r$
```

In our analysis below we will assume that $\mathsf{M}(n)$ is the number of field operations required to multiply together two polynomials of degree at most n. We will assume that $\mathsf{M}(n) \geq n$ and that $\mathsf{M}(2n) \geq 2\mathsf{M}(n)$, which are pretty reasonable assumptions.

Theorem 5.4. *The algorithm* InversePolyMod *uses $O(\mathsf{M}(n))$ field operations to correctly compute the reciprocal.*

Proof. The number of field operations used for computing h_{i+1} from h_i in Step 2 is at most $2\mathsf{M}(2^i) + 2 \cdot 2^i$, since the arithmetic is performed mod x^{2^i}. Since $n < k$, the time to compute the reciprocal is at most

$$\sum_{0<i<r} (2\mathsf{M}(2^i) + 2^{i+1}).$$

Now since $\mathsf{M}(2n) \geq 2\mathsf{M}(n)$, clearly $\mathsf{M}(2^j n) \geq 2^j \mathsf{M}(n)$ for any $j \geq 0$. This means that $\mathsf{M}(n) \leq 2^{-j}\mathsf{M}(2^j n)$, which is often useful in these sorts of summations. Here, we can see that $\mathsf{M}(2^i) \leq 2^{i-r+1}\mathsf{M}(2^{r-1})$. So the time to compute is bounded by

$$\begin{aligned}
\sum_{0<i<r} (2\mathsf{M}(2^i) + 2^{i+1}) &= 2\sum_{i=1}^{r-1} \mathsf{M}(2^i) + \sum_{i=2}^{r} 2^i \\
&\leq 2\mathsf{M}(2^{r-1}) \sum_{i=1}^{r-1} 2^{i-r+1} + \sum_{i=0}^{r} 2^i \\
&\leq \mathsf{M}(2^r) \sum_{i=0}^{r-2} 2^{-i} + 2^{r+1} \\
&\leq \mathsf{M}(2^r) \sum_{i=0}^{\infty} 2^{-i} + 2\mathsf{M}(2^r) \\
&\leq 4\mathsf{M}(2^r)
\end{aligned}$$

Since 2^r is less than $2n$, the cost of the algorithm is therefore $O(\mathsf{M}(n))$. □

We immediately get the desired result on division with remainder.

Corollary 5.5. *For polynomials of degree n in $\mathsf{F}[x]$, division with remainder requires $O(\mathsf{M}(n))$ field operations.*

It may seem circular to use an algorithm that uses the rem operation to compute division with remainder (after all, rem is closely related to division). However, we are using the rem operation to compute modulo a power of x, and hence to truncate the high powers of x. Thus we are using only a very simple form of division. It is similar to finding the quotient and remainder of a large number written in base 10 when divided by 10000. Division in this special case requires no operations.