Chinese Remainder Theorem

Theorem *Let R be a Euclidean domain with m_1, m_2, ..., $m_k \in R$.*
If $\gcd(m_i, m_j) = 1$ *for* $1 \leq i < j \leq k$ *then* $m = m_1 \cdot m_2 \cdots m_k = \mathrm{lcm}(m_1, m_2, \ldots, m_k)$ *and*

$$R/m \cong R/m_1 \times R/m_2 \times \cdots \times R/m_k;$$

given $(r_1, \ldots, r_k)$ modulo each m_i we can construct a representative $r \in R/m$.

Proof. Let Φ map elements from R to their images $(r_1, \ldots, r_k)$ modulo each m_i.

An element $r \in R$ maps to 0 under Φ if and only if it is divisible by every m_i and hence by the product m, so $\ker \Phi = m \cdot R$.

To show surjectivity we construct an element $r \in R$ with given image $s_1, \ldots, s_k$. Use the extended Euclidean algorithm to construct k elements $e_i \in R$ with the property that $e_i \equiv 1 \bmod m_i$ and $e_i \equiv 0 \bmod m_j$ for $j \neq i$. This is possible since $\gcd(m_i, m/m_i) = 1$. Now choose $t_i \in R$ such that $t_i \equiv s_i \bmod m_i$, then

$$\Phi(t_1 \cdot e_1 + \cdots + t_k \cdot e_k) = (s_1, \ldots, s_k)$$

as required.

Note that this is constructive, and is precisely the *Lagrange interpolation* generalized to arbitrary Euclidean domains.

Mixed radix representation

A slightly different algorithm, at least in the case that $R = \mathbb{Z}$, arises from the so-called mixed radix representation of integers. With notation as before, suppose that R_i forms a set of m_i representatives for $\mathbb{Z}/m_i$. Then, if r_i ranges over R_i,

$$r_1 + r_2 \cdot m_1 + r_3 \cdot m_1 \cdot m_2 + \cdots + r_k \cdot m_1 \cdots m_{k-1}$$

ranges over a set R of representatives for $\mathbb{Z}/m\mathbb{Z}$, where m is the product $m_1 \cdots m_k$ of the co-prime moduli.

Example Let $m_1 = 4$ and $m_2 = 3$, and take $R_1 = \{-1, 0, 1, 2\}$ and $R_2 = \{0, 1, 2\}$. Then we get $R = \{-1, 3, 7, 0, 4, 8, 1, 5, 9, 2, 6, 10\}$.

In practice: use either *positive* set of representatives $0, 1, \ldots, m-1$ for every modulus, or (if moduli are odd) the *symmetric* representatives $-\frac{m-1}{2}, -\frac{m-3}{2}, \ldots, -1, 0, 1, \ldots, \frac{m-3}{2}, \frac{m-1}{2}$.

Garner's algorithm

For Garners algorithm one first computes representatives for

$$\gamma_j = (m_1 \cdots m_{j-1})^{-1} \bmod m_j,$$

for $j = 2, 3, \ldots k$, using the extended Euclidean algorithm again.

Note that this step needs to be performed only once if several applications of the Chinese Remainder Theorem are required (for the same modulus).

Given images $(s_1, s_2, \ldots, s_k)$ modulo the m_i, we now compute the mixed radix digits $r_1, r_2, \ldots, r_k$.

Note: there also exists a polynomial equivalent (univariate polynomials over a field) of Garner's algorithm as well: Newton interpolation.

Application: modular determinants

See example pp 151/182.

Polynomial greatest common divisors

In the following slides we are interested in computing greatest common divisors of polynomials, in particular over factorization domains that need not be fields. Important examples are $\mathbb{Z}[x]$ and $R[x_1, x_2, \ldots, x_k][x]$.

In principle, it is very well possible to pass to the quotient field of the coefficient ring, and remove possible denominators from the result, that can now be obtained using Euclid. However, for this we need to apply the gcd-algorithm recursively and repeatedly, and the coefficients tend to blow up.

Example

We will compare various ways of computing the gcd of $f = x^8 + x^6 - 3x^4 - 3x^3 + 8x^2 + 2x - 5$ and $g = 3x^6 + 5x^4 - 4x^2 - 9x + 21$ in $\mathbb{Z}[x]$.

The classical method works over a field, here $\mathbb{Q}$. The remainders for the division algorithm become:

$$\begin{aligned} r_2 &= -\tfrac{5}{9}x^4 + \tfrac{1}{9}x^2 - \tfrac{1}{3}, \\ r_3 &= -\tfrac{117}{25}x^2 - 9x + \tfrac{441}{25}, \\ r_4 &= \tfrac{233150}{19773}x - \tfrac{102500}{6591}, \\ r_5 &= -\tfrac{1288744821}{543589225}. \end{aligned}$$

As the final remainder is a non-zero constant, the greatest common divisor of f and g is 1. Already in this tiny example the rationals grow considerably: the denominators almost double in size in every step.

With a slight modification that is necessary because the leading coefficient of the divisor may not be invertible in R, we can define division with remainder over arbitrary domains.

Proposition *Let $f, g \in R[x]$, with $\deg f = m \geq n = \deg g$ and $g \neq 0$, leading coefficient of g being b_n. There exist $q, r \in R[x]$ (unique up to units if R is a domain) such that $b_n^{m-n+1} \cdot f = q \cdot g + r$ and $\deg r < n = \deg g$.*

Proof. Uniqueness: if $q_1 g + r_1 = q_2 g + r_2$ then $(q_1 - q_2)g = (r_2 - r_1)$; and if R then $\deg(r_2 - r_1) \leq \deg r_2 < \deg g \leq \deg g + \deg(q_1 - q_2)$, a contradiction unless $r_2 - r_1 = q_1 - q_2 = 0$.

The existence is proven by way of an Algorithm.

Pseudo-division

Algorithm: Pseudo-division

Input: polynomials $f = a_m x^m + \cdots + a_0$, and $g = b_n x^n + \cdots + b_0$ in $R[x]$, with $m \geq n$ and $a_m \neq 0 \neq b_n$.
Output: $q, r \in R[x]$ such that $b_n^{m-n+1} \cdot f = q \cdot g + r$ with $\deg r < n$.

$q := 0$;
$d := m - n$;
$r := b_n^{d+1} \cdot f$;
while $d \geq 0$:
    $c := \mathrm{lc}(r)/b_n$; // leading coefficient
    $q := q + c \cdot x^d$;
    $r := r - c \cdot x^d \cdot g$;
    $d := \deg(r) - n$;

For termination it suffices to observe that $\deg r$ decreases in every passage of the loop, since the leading terms of r and of $c \cdot x^d \cdot g$ are equal; hence d decreases. The loop is executed at most $m - n + 1$ times.

For correctness, note that at the beginning of the i-th passage through the loop the following two identities hold:

(i) $b_n^{m-n+1} \cdot f = q \cdot g + r$;
(ii) $b_n^{m-n+2-i}$ divides r.

The first identity holds for $i = 1$ by definition of q and r, while at the end of the loop $q \cdot g + r$ has been replaced by $(q + c \cdot x^d) \cdot g + (r - c \cdot x^d \cdot g) = q \cdot g + r$, which hence is invariant.

The second identity holds by definition for $i = 1$. If it holds for i, then $b_n^{m-n+1-i}$ divides c and therefore also $r - c \cdot x^d \cdot g$, which is the r at the beginning of the $i+1$-th passage. Since the loop is executed at most $m - n + 1$ times this shows that q and r polynomials in $R[x]$.

Note that the actual number of times the loop is executed may be less than $m-n+1$ because the degree of r may drop by more than 1. In that case the exponent of b_n in $b_n^{m-n+1} \cdot f = q \cdot g + r$ could be lowered accordingly. The algorithm should be modified to prevent q and r from picking up spurious factors b_n.

In principle (pseudo-)division with remainder can be used to find common divisors, as in the ordinary Euclidean algorithm for integers.

Definition Let $f, g \in R[x]$ and suppose $\deg f \geq \deg g$. A *polynomial remainder sequence for* f, g is a sequence $u_0 = f, u_1 = g, u_2, \ldots, u_k$ for which there exist $q_1, q_2, \ldots, q_k$ as well as $\alpha_1, \alpha_2, \ldots, \alpha_k, \beta_1, \beta_2, \ldots \beta_k \in R$

- $\alpha_i u_{i-1} = q_i u_i + \beta_i u_{i+1}$, for $i = 1, 2, \ldots, k-1$;
- $\deg u_{i+1} < \deg u_i$, for $i = 1, 2, \ldots, k-1$;
- $\alpha_k u_{k-1} = q_k u_k$.

The non-zero polynomials $f, g \in R[x]$ are equivalent over R, denoted $f \sim g$ if and only if there exist non-zero $r, s \in R$ such that $r \cdot f = s \cdot g$. Over the field of fractions of R this means precisely that f is a non-zero multiple of g.

Proposition *Let $u_0, u_1, \ldots, u_k$ be a polynomial remainder sequence for f, g. Then for $i = 2, \ldots, k$:*

- *$u_i \sim r_i$, where r_i is the pseudo-remainder of u_{i-2} and u_{i-1};*
- *there exist $\gamma_i, \delta_i \in R[x]$ such that*

$$\beta_1 \cdot \beta_2 \cdots \beta_{i-1} \cdot u_i = \gamma_i u_0 + \delta_i u_1;$$

more precisely, this holds with

$$\begin{aligned} \gamma_i &= -q_{i-1}\gamma_{i-1} + \alpha_{i-1}\beta_{i-2}\gamma_{i-2} \\ \delta_i &= -q_{i-1}\delta_{i-1} + \alpha_{i-1}\beta_{i-2}\delta_{i-2} \end{aligned}$$

$$\beta_0 = 1, \gamma_0 = 1, \gamma_1 = 0, \delta_0 = 0, \delta_1 = 1.$$

Corollary *Up to $\sim$ polynomial remainder sequences are uniquely determined by f, g.*
Corollary *Let R be a unique factorization domain; if $u_0, u_1, \ldots, u_k$ is a polynomial remainder sequence for f, g then $u_k \sim \gcd(f, g)$.*

All this implies that there is very little choice left in using polynomial remainder sequences for determining greatest common divisors. Nevertheless, different strategies lead to significantly different results, as we will illustrate in the case of $\mathbb{Z}[x]$. The main complication over $\mathbb{Z}$ lies in the *intermediate coefficient growth*: even if both input and output are small, it may be that intermediate results grow enormously, if no special care is taken. In the following example we show some ways of choosing polynomial remainder sequences and their effects.

Example We compare various ways of constructing polynomial remainder sequences for the pair of polynomials $f = x^8+x^6-3x^4-3x^3+8x^2+2x-5$ and $g = 3x^6+5x^4-4x^2-9x+21$ in $\mathbb{Z}[x]$.

(a) classical:

The classical method works over a field and takes $\alpha_i = \beta_i = 1$ for all i. The polynomial remainder sequences becomes, as we saw before:

$$\begin{aligned} u_2 &= -\tfrac{5}{9}y^4 + \tfrac{1}{9}y^2 - \tfrac{1}{3}, \\ u_3 &= -\tfrac{117}{25}y^2 - 9y + \tfrac{441}{25}, \\ u_4 &= \tfrac{233150}{19773}y - \tfrac{102500}{6591}, \\ u_5 &= -\tfrac{1288744821}{543589225}. \end{aligned}$$

The other choices perform operations entirely within $\mathbb{Z}$, and use $\alpha_i = c_i^{d_{i-1}-d_i+1}$, where c_i is the leading coefficient of u_i and $d_i = \deg u_i$. The methods differ in the choice of β_i.

(b) Euclidean:

Choose $\beta_i = 1$. This is the choice taking the ordinary usual pseudo-division in each step. In our example it determines the following remainder sequence:

$$\begin{array}{lcr} u_2 & = & -15x^4 + 3x^2 - 9, \\ u_3 & = & 15795x^2 + 30375x - 59535, \\ u_4 & = & 1254542875143750x - 1654608338437500, \\ u_5 & = & 1259333879550074310093114199218750 0. \end{array}$$

Of course the result $\gcd(f, g) = 1$ is as before (and since f is monic), but the coefficient growth is spectacular: u_5 has 35 decimal digits!

(c) primitive:

For this variant one takes for β_i the contents of the pseudo-remainder of u_{i-1} by u_i; then u_{i+1} becomes the primitive part of that pseudo-remainder. In our example:

$$\begin{aligned} u_2 &= -5x^4 + x^2 - 3, \\ u_3 &= 13x^2 + 25x - 49, \\ u_4 &= 4663x - 6150, \\ u_5 &= 1. \end{aligned}$$

This straightforward choice leads to the smallest sequence of polynomial remainders. But the price to be paid is that in every step the contents has to be computed, which is a greatest common divisor computation on the sequence of coefficients.

(d) reduced:

Take $\beta_i = \alpha_{i-1}$. Here and in the next variant the idea is to avoid gcd computations by taking out a common factor of the pseudo-remainder that is easily computed; it is not obvious that α_{i-1} works, but it does. The result in our case:

$$\begin{aligned} u_2 &= -15x^4 + 3x^2 - 9, \\ u_3 &= 585x^2 + 1125x - 2205, \\ u_4 &= -18885150x + 24907500, \\ u_5 &= 527933700, \end{aligned}$$

which is not bad given that no gcds are computed.

(e) subresultant:

The choice for β_i here is not obvious, and follows from subresultant consideration. Explicitly β_i is defined recursively by

$$\begin{aligned}
\psi_i &= (-c_{i-1})^{d_{i-2}-d_{i-1}}\psi_{i-1}^{1-d_{i-2}+d_{i-1}}, \\
\beta_i &= -c_{i-1}\psi_i^{d_{i-1}-d_i} \quad \text{with} \\
\psi_1 &= -1, \quad \text{and} \\
\beta_1 &= (-1)^{d_0-d_1+1}
\end{aligned}$$

using the same notation as before. It results in:

$$\begin{aligned}
u_2 &= 15x^4 - 3x^2 + 9, \\
u_3 &= -65x^2 - 125x + 245, \\
u_4 &= 9326x - 12300, \\
u_5 &= -260708,
\end{aligned}$$

for our standard example. That is better than the reduced case, and also avoids gcd computations. This choice seems to guarantee coefficient grow that is linear at worst.

Modular gcd

There is an alternative method for computing greatest common divisors in $\mathbb{Z}[x]$; it uses reduction modulo prime numbers. Both this and an algorithm for factorization of polynomials over the integers will use reduction modulo primes in the hope that over $\mathbb{F}_p$ the problem will be easier, and yet informative about the integer case. Suppose that $f \in \mathbb{Z}[x]$ factors: $f = c \cdot d \in \mathbb{Z}[x]$; then clearly for the reduction modulo p (which we will denote by $\bar{}$, we find accordingly: $\bar{f} = \bar{c} \cdot \bar{d}$. However, it may well be that both sides reduce to 0, or at least that the degree of c or d drops.

This first (easy) complication can be overcome by avoiding some primes.

Lemma *For all but finitely many primes p it holds that* $\deg \gcd(\bar{f}, \bar{g}) = \deg \gcd(f, g)$.

If we would know a bound on the integer coefficients of the gcd of two polynomials in $\mathbb{Z}[x]$, knowledge of the greatest common divisor modulo a sufficiently large determines the coefficients uniquely. Such a bound is provided by the Landau-Mignotte estimate.

Put $\|f\| = \sqrt{\sum_{i=0}^{m} |a_i|^2}$ for $f \in \mathbb{C}[x]$.

Theorem (Landau-Mignotte) *If g divides f in $\mathbb{C}[x]$ then* $|b_i| \leq \frac{|b_n|}{|a_m|}\binom{n}{i}\|f\|$.

Corollary *If $c = \gcd(f, g) \in \mathbb{Z}[x]$, then*

$$c_i \leq M(f, g) = 2^{\min(m,n)} \gcd(a_m, b_n) \min\left(\frac{\|f\|}{|a_m|}, \frac{\|g\|}{|b_n|}\right),$$

for $0 \leq i \leq \deg c$, where $c = \sum_i c_i x^i$.

The idea behind the modular gcd algorithm is now to combine information modulo various primes, rather than to do the computation modulo one very large prime. The information is combined by application of the Chinese remainder theorem.

One additional complication is that after reduction modulo p the leading coefficient can be adapted freely, while the integer leading coefficient is uniquely determined (up to sign). This problem is overcome by looking at primitive polynomials only, and taking the primitive part of the result in the end.

Algorithm: Modular gcd

$l := \gcd(a_m, b_n)$; $B := 2 \cdot l \cdot M(f, g)$;
repeat
    $p :=$ nextprime(p, l);
    $m := \gcd(\bar{f}, \bar{g})$;
    $m := l \bmod p \cdot (m/m_{\deg m})$;
    if $\deg m = 0$ then
        return 1;
    $P := p$;
    $n := m$;
    while $P \leq M$ do
        $p :=$ nextprime(p, l);
        $m := \gcd(\bar{f}, \bar{g})$;
        if $\deg m < \deg n$ then
            break; *from the while loop:*
              go to begin of repeat loop
        if $\deg m = \deg n$ then
            $n :=$ Chinese(n, m, P, p);
            $P := p \cdot P$;
    $n :=$ pp(n);
until $n|f$ and $n|g$;
return n;

Remarks

The algorithm calls subroutines M, `Chinese` and `nextprime`.

The value of $M(f, g)$ should be the Landau-Mignotte bound on f and g.

The function `Chinese`(a, b, P, p) returns a polynomial $c \in \mathbb{Z}/(P \cdot p)\mathbb{Z}[x]$ such that $c \equiv a \bmod P$ and $c \equiv b \bmod p$, which exists and is essentially unique by the Chinese remainder theorem.

The function `nextprime`(p, l) returns the first prime not dividing l following p in some ordering of the primes. This ordering need not be the usual ordering by size; in fact in practice it is better to use primes that are fairly big, but not too big so that the work modulo p contributes significantly to the product P but computations modulo p are not too expensive. One often starts at the largest prime not exceeding the square root of a word in computer memory, and then works downwards in size.

The Landau-Mignotte estimate is often much too pessimistic. Therefore, in practice one often replaces the criterion that $P > M$ by a test to see if the current n considered as integral polynomial divides f and g already; if so that polynomial is immediately returned. It follows from the exercises that this so-called *early-abort* strategy will not lead to wrong answers.