

Euclidean Algorithm and Division in Finite Fields

The *greatest common divisor* $\gcd(r, s)$ of two integers r and s is the largest integer that divides both r and s . Euclid's algorithm is an efficient method for finding $\gcd(r, s)$; it is much faster than factoring r and s and selecting common factors. As a byproduct of the Euclidean algorithm, we can find integers a and b such that $d = \gcd(r, s) = ar + bs$. If $\gcd(r, s) = 1$ then the algorithm produces the reciprocal of r in the ring of integers modulo s .

The idea of the Euclidean algorithm is to reduce the computation of $\gcd(r, s)$ for $r, s > 0$ to the computation of $\gcd(r', s')$ where the pair (r', s') is "smaller" than (r, s) . One definition of *smaller* is that both $r' \leq r$ and $s' \leq s$ and either $r' < r$ or $s' < s$. If $r < s$ then $(s \bmod r) < r$ so $(r', s') = (r \bmod s, r)$ is a smaller problem according to this definition.

The Euclidean algorithm generates a decreasing sequence $r_1 > r_2 > \dots > r_n > 0$ where each r_i is obtained from the previous two numbers by $r_i = r_{i-2} \bmod r_{i-1}$ and the final value is $r_n = \gcd(r, s)$. The remainders are defined iteratively as follows:

$$\begin{aligned} s &= Q_1 r + r_1 & 0 < r_1 < |r| \\ r &= Q_2 r_1 + r_2 & 0 < r_2 < r_1 \\ r_1 &= Q_3 r_2 + r_3 & 0 < r_3 < r_2 \\ &\vdots \\ r_{n-2} &= Q_n r_{n-1} + r_n & 0 < r_n < r_{n-1} \\ r_{n-1} &= Q_{n+1} r_n \end{aligned}$$

This procedure halts after a finite number of steps because each remainder is a positive number smaller than the preceding remainder.

The successive remainders satisfy the linear recurrence $r_i = r_{i-2} - Q_i r_{i-1}$ with initial conditions $r_{-1} = s$ and $r_0 = r$. It follows that every common divisor of r_{i-1} and r_{i-2} also divides r_i , and every common divisor of r_i and r_{i-1} also divides r_{i-2} . Therefore $\gcd(r_i, r_{i-1}) = \gcd(r_{i-1}, r_{i-2})$ for every i . In particular, $r_n = \gcd(r_n, r_{n-1}) = \gcd(r_0, r_{-1}) = \gcd(r, s)$.

Each remainder r_i is an integer linear combination of the previous two remainders, so by induction the final remainder is a linear combination of the first two remainders $r_0 = r$ and $r_{-1} = s$; that is, $r_n = \gcd(r, s) = ar + bs$ for some integers a and b . (Obviously, a and b have different signs.) The coefficients a and b can be computed iteratively by generating the sequences $\{a_i\}$ and $\{b_i\}$ such that

$$r_i = a_i r + b_i s \quad i = -1, 0, 1, \dots, n$$

The initial values for $\{a_i\}$ and $\{b_i\}$ are obviously $a_{-1} = 0$, $b_{-1} = 1$ and $a_0 = 1$, $b_0 = 0$. We can determine a_i and b_i from a_{i-1} , a_{i-2} , b_{i-1} , b_{i-2} as follows:

$$\begin{aligned} r_i &= -Q_i r_{i-1} + r_{i-2} \\ &= -Q_i (a_{i-1} r + b_{i-1} s) + (a_{i-2} r + b_{i-2} s) \\ &= (-Q_i a_{i-1} + a_{i-2}) r + (-Q_i b_{i-1} + b_{i-2}) s = a_i r + b_i s \end{aligned}$$

We see that the sequences $\{a_i\}$ and $\{b_i\}$ satisfy the same time-varying linear recurrence that defines the remainder sequence $\{r_i\}$:

$$a_i = -Q_i a_{i-1} + a_{i-2}$$

$$b_i = -Q_i b_{i-1} + b_{i-2}$$

Example: Find a and b such that $\gcd(17, 37) = 17a + 37b$.

r_i	Q_i	a_i	b_i
37	—	0	1
17	—	1	0
3	2	-2	1
2	5	11	-5
1	1	-13	6
0	2	—	—

To check this result: $17 \cdot (-13) + 37 \cdot 6 = -221 + 222 = 1$.

An important application of the extended Euclidean algorithm is in computing reciprocals in finite fields $\text{GF}(p)$, where p is a prime number. If r is a nonzero element of $\text{GF}(p)$ then $0 < r < p$ and so r is relatively prime to p ; that is, $\gcd(r, p) = 1$. The extended Euclidean algorithm yields integers a and b such that $1 = ar + bp$. Therefore the reciprocal of r is $r^{-1} = a \bmod p$. The example above shows that the reciprocal of 17 in $\text{GF}(37)$ is $-13 = 24$.

When using Euclid's algorithm to find reciprocals in $\text{GF}(p)$, the b_i column is needed only as a check. It can be shown that the value a_n produced by the Euclidean algorithm is always less than p in magnitude, so $a \bmod p$ is either a or $p + a$.

Simplified divisions

The Euclidean algorithm as described above appears to involve repeated divisions. To guarantee that each remainder is small, namely, $0 \leq r_i < r_{i-1}$, the quotients must be defined by $Q_i = \lfloor r_{i-2}/r_{i-1} \rfloor$. In fact, very rough divisions are sufficient, and other choices for Q_i may be computationally more efficient. Choosing Q_i to be the largest power of 2 $\leq r_{i-2}/r_{i-1}$ leads to an implementation of the Euclidean algorithm whose complexity is about the same as that of the usual binary division algorithm.

Example: Find the reciprocal of 32 in $\text{GF}(109)$ using incomplete division.

r_i	Q_i	a_i	b_i
109	—	0	1
32	—	1	0
45	2	-2	1
32	0	1	0
13	1	-3	1
6	2	7	-2
1	2	-17	5
0	6	—	—

Therefore in $\text{GF}(109)$ the reciprocal of 32 is $-17 = 92$.

Another method for computing the gcd that avoids division is Stein’s binary GCD algorithm, Algorithm B in section 4.5.2 of D. E. Knuth’s *The Art of Computer Programming*, volume 2.

Euclidean algorithm for polynomials

The Euclidean algorithm can be used to compute greatest common divisors of polynomials whose coefficients belong to a field. The extended Euclidean algorithm for polynomials produces polynomials $a(x)$ and $b(x)$ such that

$$\text{gcd}(r(x), s(x)) = a(x)r(x) + b(x)s(x).$$

If $r(x)$ and $s(x)$ are relatively prime, then $a(x)$ is the reciprocal of $r(x)$ in the ring of polynomials mod $s(x)$. If $s(x)$ is irreducible, then every nonzero polynomial $r(x)$ of degree less than $\text{deg } s(x)$ is relatively prime to $s(x)$, so every nonzero element in the ring of polynomials mod $s(x)$ has a reciprocal. It follows that the polynomials with arithmetic modulo an irreducible polynomial form a field.

Example: Find the reciprocal of $x^3 + x^2$ modulo $x^4 + x + 1$ over GF(2).

$r_i(x)$	$Q_i(x)$	$a_i(x)$	$r_i(x)$	$Q_i(x)$	$a_i(x)$
$x^4 + x + 1$	–	0	10011	–	0
$x^3 + x^2$	–	1	1100	–	1
$x^2 + x + 1$	$x + 1$	$x + 1$	111	11	11
x	x	$x^2 + x + 1$	10	10	111
1	$x + 1$	$x^3 + x$	1	11	1010
0	x	–	0	10	–

Thus $(x^3 + x^2)^{-1} \text{ mod } (x^4 + x + 1) = x^3 + x$. In the second tableau, we use *synthetic* representation of polynomials as bit vectors (msb first). This representation is natural for both computer software and hardware implementations of polynomial arithmetic. Addition and subtraction are simply bitwise exclusive-or, while multiplication and division require shift-and-xor steps.

The implementation of the Euclidean algorithm for polynomials can be simplified by choosing the quotients $Q_i(x)$ to be monomials x^j rather than the result of the complete division $\lfloor r_{i-2}(x)/r_{i-1}(x) \rfloor$. This version of the algorithm will usually require more steps than the version that uses complete polynomial division to determine $Q_i(x)$, but the computation of the quotient, remainder, and $a_i(x)$ is easier at each step.

Example: Find the reciprocal of $x^3 + x + 1$ modulo $x^5 + x^2 + 1$ over GF(2) using incomplete synthetic divisions.

r_i	Q_i	a_i
100101	–	0
1011	–	1
1001	100	100
10	1	101
1	100	10000
0	10	–

Therefore $(x^3 + x + 1)^{-1} \text{ mod } (x^5 + x^2 + 1) = x^4$.

Computational complexity of Euclidean algorithm

One way to measure the running time of Euclid's algorithm for integers is to count the number of division steps. A division step reduces the problem by only a small amount when the quotient is small, such as $q_i = 1$. But on average, the remainders in the Euclidean algorithm decrease exponentially at a rate no smaller than $\phi = (1 + \sqrt{5})/2 \approx 1.61803$ (the ubiquitous Golden Ratio).

The worst case for the Euclidean algorithm is the computation of the greatest common divisor of two successive Fibonacci numbers. The *Fibonacci numbers* are the numbers in the sequence $\{F_i\}$ defined by the linear recurrence $F_n = F_{n-1} + F_{n-2}$, with the initial values $F_0 = 0, F_1 = 1$. The first few Fibonacci numbers are 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55. It can be shown that F_{n+2} and F_{n+1} are the smallest pair of numbers for which the gcd computation requires n division steps. (See Knuth's *Art of Computer Programming*, volume 2, section 4.5.3.)

Example: It is easy to show that $\gcd(F_{n+1}, F_{n+2}) = 1$. (An amazing fact: $\gcd(F_m, F_n) = F_{\gcd(m,n)}$.) The Euclidean algorithm takes quite a few steps to determine this. Consider $F_8 = 21, F_9 = 34$.

r_i	Q_i	a_i	b_i	r_i	Q_i	a_i	b_i	r_i	Q_i	a_i	b_i
34	—	0	1	35	—	0	1	34	—	0	1
21	—	1	0	21	—	1	0	22	—	1	0
13	1	-1	1	14	1	-1	1	12	1	-1	1
8	1	2	-1	7	1	2	-1	10	1	2	-1
5	1	-3	2	0	2	—	—	2	1	-3	2
3	1	5	-3					0	4	—	—
2	1	-8	5								
1	1	13	-8								
0	1	—	—								

Euclid's algorithm requires many more steps to find $\gcd(34, 21)$ than to find the gcd of nearby larger pairs of numbers, $(35, 21)$ and $(34, 22)$.

Computation of reciprocals in $\text{GF}(2^m)$ using the extended Euclidean algorithm for polynomials can be accomplished in about m clock cycles using m -bit wide shifts and exclusive-ors. The inputs to the Euclidean algorithm are $p(x)$, the prime polynomial over $\text{GF}(2)$ of degree m that defines arithmetic in $\text{GF}(2^m)$, and $r(x)$, a polynomial of degree less than m whose reciprocal in $\text{GF}(2^m)$ is being computed. Each polynomial division step produces a remainder whose degree is less than that of the previous remainder. At most m division steps are needed to obtain the gcd (which is known in advance to be 1) and the polynomial $a(x)$ that satisfies $a(x)r(x) = 1 \pmod{p(x)}$.

Some division steps require more cycles because dividing by a polynomial whose degree is much less than that of the dividend takes additional cycles. But in this case, the resulting remainder is of small degree. The overall computation consists of a mixture of easy and hard division steps, but the total number of clocks needed is $O(m)$ in the worst case. (The multiplicative constant hidden by the $O(m)$ notation is small—close to 1 in a careful implementation.)

Other methods for finding reciprocals in $\text{GF}(2^m)$ are covered in a future handout.