Computing Multiplicative Inverses in GF(p)

By George E. Collins

Abstract. Two familiar algorithms, the extended Euclidean algorithm and the Fermat algorithm (based on Fermat's theorem $a^p \equiv a \pmod{p}$), are analyzed and compared as methods for computing multiplicative inverses in GF(p). Using Knuth's results on the average number of divisions in the Euclidean algorithm, it is shown that the average number of arithmetic operations required by the Fermat algorithm is nearly twice as large as the average number for the extended Euclidean algorithm. For each of the two algorithms, forward and backward versions are distinguished. It is shown that all numbers computed in the forward extended Euclidean algorithm are bounded by the larger of the two inputs, a property which was previously established by Kelisky for the backward version. ■

1. Introduction. The importance of congruence arithmetic (i.e., arithmetic in a Galois field GF(p) with a prime number, p, of elements) in calculations requiring extensive operations on large integers is receiving increasing recognition. Takahasi and Ishibashi in an early paper [5] discussed several applications and noted the resulting economies in computing time, memory requirements and programming effort. More recently, Borosh and Fraenkel in [1] have treated its use in the exact solution of systems of linear equations. The author is currently using it in developing faster methods for polynomial g.c.d. calculation than those considered in [2].

All arithmetic operations in GF(p) are simple and rapid with the exception of division or, equivalently, the calculation of the multiplicative inverse of a nonzero element b, $0 < b < p$. It is well known that this calculation can be conveniently and efficiently performed either by using an algorithm based on Fermat's theorem or by using an extension of the Euclidean algorithm for the g.c.d. of two integers. Both methods are briefly considered in [5], for example.

In the following we make a careful comparison of the two algorithms. For each algorithm we distinguish between a forward and a backward version. A theorem on the forward Euclidean algorithm, analogous to a theorem of Kelisky, [3], on the backward Euclidean algorithm is proved to the effect that all integers which arise in the calculation are nicely bounded. It is shown, using a conjecture of Knuth, [4], that the Euclidean algorithm is about twice as fast as the Fermat algorithm on the average.

2. Euclidean Algorithm. In the Euclidean algorithm we are given integers a and b and we compute $d = \gcd(a, b)$ as follows. For convenience we may assume $a > b > 0$. Set $a_1 = a$, $a_2 = b$. Given a_i and a_{i+1} with $a_i > a_{i+1} > 0$ define q_i and a_{i+2} by the relations $a_i = a_{i+1} q_i + a_{i+2}$, $0 \leqq a_{i+2} < a_{i+1}$. We thus compute a finite sequence $a_1 > a_2 > \cdots > a_{n+1} > a_{n+2} = 0$ and we have $d = a_{n+1}$.

We will consider two ways of extending this algorithm in order to also obtain integers x and y such that $ax + by = d$. One of these ways is described by Knuth

Received June 11, 1968.

in [4]. It amounts to the following. Set $x_1 = 1$, $y_1 = 0$, $x_2 = 0$ and $y_2 = 1$. Given x_i, x_{i+1}, y_i, y_{i+1} and q_i (as defined above), set $x_{i+2} = x_i - q_i x_{i+1}$ and $y_{i+2} = y_i - q_i y_{i+1}$. Then $x = x_{n+1}$ and $y = y_{n+1}$. Clearly $ax_i + by_i = a_i$ holds for $i = 1$ and $i = 2$ and by a simple induction using the recurrence relations for the x_i, y_i and a_i, it holds for all i, in particular for $i = n + 1$.

The other method has been considered by Kelisky in [3]. It can be described as follows. Having computed the sequences $\{a_i\}$ and $\{q_i\}$ and having saved the q_i, set $u_1 = 0$, $u_2 = 1$. Given u_i and u_{i+1}, set $u_{i+2} = u_i - q_{n-i}u_{i+1}$. Then $x = u_n$ and $y = u_{n+1}$. Clearly $d = u_i a_{n-i+1} + u_{i+1}a_{n-i+2}$ holds for $i = 1$ and by a simple induction using the recurrence relations for the u_i and a_i, it holds for all i, in particular for $i = n$.

Knuth calls the former method the extended Euclidean algorithm. In order to distinguish the two we shall, for obvious reasons, call that method the forward extended Euclidean algorithm and the one considered by Kelisky the backward extended Euclidean algorithm; for brevity in the following we shall just refer to the forward and backward methods.

Clearly, the backward method has the disadvantage that all the q_i must be saved temporarily, although Lamé's theorem [6] gives us a tight bound, $5[\log_{10} a + 1]$, for the number of these quotients. The backward method would appear somewhat faster (by a ratio of about 3 to 2) since a single sequence $\{u_i\}$ replaces the two sequences $\{x_i\}$ and $\{y_i\}$. However, in the forward method we can dispense with the $\{x_i\}$ sequence since x can be computed as $(d - by)/a$ once d and y are known. Then the forward and backward methods become equal in computing time. Further, it is only y that is wanted when a is p and we are computing b^{-1} in GF(p).

Kelisky showed in [3] that $u_i < a$ for all i. In fact, he showed that $|u_i| \leqq \frac{1}{2}a_{n-i+2}/d$ for all i. It follows that if a is a single-precision integer, so are all intermediate quantities in the backward algorithm. We will now establish a similar result for the forward algorithm.

THEOREM. *$x_{n+2} = (-1)^{n+1}b/d$ and $y_{n+2} = (-1)^n a/d$.*

Proof. Let D_i be the determinant

$$\begin{vmatrix} x_i & y_i \\ x_{i+1} & y_{i+1} \end{vmatrix}.$$

Then

$$D_{i+1} = \begin{vmatrix} x_{i+1} & y_{i+1} \\ x_{i+2} & y_{i+2} \end{vmatrix} = \begin{vmatrix} x_{i+1} & y_{i+1} \\ x_i - q_i x_{i+1} & y_i - q_i y_{i+1} \end{vmatrix}$$

$$= \begin{vmatrix} x_{i+1} & y_{i+1} \\ x_i & y_i \end{vmatrix} = -D_i.$$

But $D_1 = 1$, so $D_i = (-1)^{i+1}$ for all i. It follows that $\gcd(x_i, y_i) = 1$ for all i, in particular $\gcd(x_{n+2}, y_{n+2}) = 1$. But $ax_{n+2} + by_{n+2} = a_{n+2} = 0$ and hence $(a/d)x_{n+2} = -(b/d)y_{n+2}$. Since $\gcd(a/d, b/d) = 1$ also, $x_{n+2} = \pm b/d$. The appropriate signs are determined by the observation that the sequences $\{x_i\}$ and $\{y_i\}$ alternate in sign.

From the recurrence relations we easily establish, moreover, that $0 < x_3 < -x_4 < x_5 < -x_6 < \cdots$ and $0 < y_2 < -y_3 < y_4 < -y_5 < \cdots$. Hence in the forward algorithm, also, all intermediate numbers are single-precision provided a is.

We have, in fact, $x_{n+2} = x_n - q_n x_{n+1}$, $|q_n x_{n+1}| = |x_n - x_{n+2}| \leqq |x_{n+2}| = b/d$, hence $|x_{n+1}| \leqq \frac{1}{2} b/d$ since $q_n \geqq 2$. Similarly $|y_{n+1}| \leqq \frac{1}{2} a/d$.

3. Comparison with the Fermat Algorithm. By Fermat's theorem [6], $b^{p-1} \equiv 1 \pmod{p}$ if b is a nonzero element of GF(p) and hence $b^{-1} = b^{p-2}$ in GF(p). b^{p-2} can be conveniently and efficiently evaluated using the binary representation $p - 2 = \sum_{i=0}^{n} a_i 2^i$. Here again there are two possible algorithms, which could be called forward and backward, respectively.

In the forward algorithm we set $r_0 = b$, $s_0 = 1$ and use the recurrence relations $r_{i+1} = r_i^2$,

$$\begin{aligned} s_{i+1} &= s_i, \quad \text{if } a_{i+1} = 0 \\ &= s_i r_i, \quad \text{if } a_{i+1} = 1. \end{aligned}$$

Then $s_{n+1} = b^{p-2}$. Here the low-order bits of $p - 2$ are used first.

In the backward algorithm, the high-order bits are used first, as follows. Set $t_0 = 1$ and use

$$\begin{aligned} t_{i+1} &= t_i^2, \quad \text{if } a_{n-i} = 0 \\ &= b t_i^2, \quad \text{if } a_{n-i} = 1. \end{aligned}$$

Then $t_{n+1} = b^{p-2}$.

The amount of computation is the same for the two Fermat algorithms. In either algorithm one must perform approximately $n + m$ multiplications, where m is the number of one bits in the binary representation of $p - 2$. If one may assume that on the average $m = n/2$, then the average number of multiplications for either Fermat algorithm is approximately $\frac{3}{2} \log_2 p$. Since these are multiplications in GF(p), there will be associated with each integer multiplication an integer division to reduce the result to the range $0 \leqq x < p$.

If either of the Euclidean algorithms is used, then for each of the n divisions required we must also do one multiplication and one subtraction (the sequence $\{x_i\}$ is not computed in the forward algorithm; we compute either $y_{i+2} = y_i - q_i y_{i+1}$ or $u_{i+2} = u_i - q_{n-i} u_{i+1}$). According to Knuth [4, p. 7], the average value of n is $(12 (\ln 2)/\pi^2) \ln p = .8427 \ln p = .584 \log_2 p$, approximately.

Hence, the Fermat algorithm requires an average of about $3 \log_2 p$ arithmetic operations whereas the Euclidean algorithm requires an average of about $(7/4) \times \log_2 p$ arithmetic operations. In the Fermat algorithm, half of the operations are multiplications and half are divisions. In the Euclidean algorithm, there are equal numbers of multiplications, divisions and subtractions. Since subtractions generally take less time than multiplications or divisions, and since the multipliers in the Euclidean algorithm are almost all very small integers, one would expect the Euclidean algorithm to be approximately twice as fast as the Fermat algorithm on most computers.

As an empirical test of this analysis, both algorithms were programmed in assembly language for the CDC 1604 computer. The Fermat program contains a loop which is executed once for each bit of $p - 2$. When half of these bits are ones, the average loop execution time is about 206 microseconds. The Euclid program contains a loop which is executed once for each division and whose execution time is about 174 microseconds.

It follows that on the 1604 the Euclidean algorithm should take about .584(174/206) = .49 of the time for the Fermat algorithm. A large number of trials using primes of the order 10^{13} were timed and the results agreed with this analysis within a relative error of about five per cent. The average time for an inversion using the Euclidean algorithm was about 4 milliseconds.

It is worth noting that it is often possible to use only primes p such that $p - 2$ contains only a few ones in its binary representation, in which case the Fermat algorithm is faster than otherwise. However, even in the ideal case the Euclidean algorithm time was found to be only about .68 of the Fermat algorithm time. The two programs are about equal in size and were equally easy to write.

4. Acknowledgement. This study has been aided by the interest and programming assistance of both W. J. Fabens and Ellis Horowitz. Support was also provided by the University of Wisconsin Computing Center.

Computer Sciences Department
University of Wisconsin
Madison, Wisconsin 53706

1. I. Borosh & A. S. Fraenkel, "Exact solutions of linear equations with rational coefficients by congruence techniques," *Math. Comp.*, v. 20, 1966, pp. 107–112. MR **32** #4831.
2. G. E. Collins, "Subresultants and reduced polynomial remainder sequences," *J. Assoc. Comput. Mach.*, v. 14, 1967, pp. 128–142. MR **35** #6352.
3. R. P. Kelisky, "Concerning the Euclidean algorithm," *Fibonacci Quart.*, v. 3, 1965, pp. 219–223. MR **32** #5579.
4. D. E. Knuth, *The Art of Computer Programming.* Vol. 1: *Fundamental Algorithms*, Addison-Wesley, Reading, Mass., 1968.
5. H. Takahasi & Y. Ishibashi, "A new method for 'exact calculation' by a digital computer," *Information Processing in Japan*, v. 1, 1961, pp. 28–42.
6. J. V. Uspenshy & M. A. Heaslet, *Elementary Number Theory*, McGraw-Hill, New York, 1939. MR **1**, 38.