Effective computations of module inverses with the Approximating k-ary GDD Algorithm by Ishmukhametov

Al Khalidi Arkan M.
Kazan Federal University, Russia
Corresponding author: arkan3538@gmail.com

Abstract. Finite field calculations are used in modern cryptographic protocols for generating keys, encrypting and decrypting data, and building an electronic digital signature. The module inversing is necessary part of these calculations based on the extended Euclidean algorithm. Ishmukhametov developed a new algorithm for calculating the greatest common divisor of natural numbers called the approximating algorithm which is a variant of the k-ary GCD Algorithm by J. Sorenson. In this paper we develop an extension version of this algorithm.

1. Introduction

The approximating algorithm was developed in 2016 by S. Ishmukhametov [1]. This algorithm is a modification of the k-ary GCD algorithm invented in the early 90s of the last century by Jonathan Sorenson [2] and independently by D. Jebelean [3]. Further development of the Algorithm was performed by I. Amer, A. Arkan M., R. Enikeev and other researchers [4-10]).

One of the drawbacks of the k-ary GCD Algorithm is that when it runs it accumulates so called extraneous factors, that is, new factors that are not part of the original GCD, so the final value of the Algorithm is a multiple of the required value. This implies a need in additional operations to calculate the common GCD of the initial numbers A and B together with output d of the k-ary Algorithm:

\[ d_0 = \text{GCD}(A, B, d) \]

In articles [11- 12], the authors use various methods to solve this problem of the appearance of extraneous factors, accelerating the general procedure for calculating GCD. In [13], a new algorithm was proposed for computing inverse modulo elements based on an approximating algorithm. This article provides theoretical data on the convergence of the extended approximating algorithm and provides examples of the inverse operations modulo based on this algorithm. In our article we will consider in detail the construction of an effective algorithm for solving the Bezout equation, the calculation of inverse elements modulo a given number, and the programming of these algorithms. We start with a theoretical description of the algorithm for solving the equation \( Au + Bv = 1 \) for given coprime numbers A and B. Solving the Bezout equation using the extended Euclidean algorithm

As mentioned earlier, one of the important tasks solved using the extended version of the Euclidean algorithm is to solve the Bezout equation:

\[ Au + Bv = d \quad (1) \]

where A, B are given natural numbers, d is their greatest common divisor, and u, v are unknown coefficients called Bezout coefficients.

When Bezout’ coefficients are known, one can find elements inverse in modulus. Indeed, let the numbers A and B be coprime, i.e. their GCD d is equal to 1, then taking the rest of both parts of (1) by modulo A, we obtain \( B^{-1} \equiv y \mod A \).

The solution of the Bezout equation according to the scheme of the extended Euclidean algorithm can be shown using a table of 7 columns. Consider this scheme as an example of a pair of numbers \( A = 185, B = 39 \) (table 1).

Table 1. An example of solving the Bezout equation
The first column contains iteration numbers. We put original numbers $A$ and $B$ in the first line and start an iteration procedure. At each iteration of the algorithm, it calculates the integer quotient of division $A$ by $B$ and the remainder $A \mod B$. Then we move values of $B$ and $A \mod B$ down-left, obtaining the source data for the next iteration.

The procedure is performing several times until zero appears in column $A \mod B$. At position $B$ in this line we get the GCD of the numbers $A$ and $B$. Since $A$ and $B$ are coprime, in our case the GCD is equal 1. This completes the direct run of the Euclidean Algorithm.

Then we implement the second part of the Algorithm filling in last two columns. It begins by writing 0 and 1 in the last line and continues by computing other values $u$ and $v$ by formulas

$$
\begin{align*}
  u_i &= u_{i+1} \\
v_i &= u_{i+1} - v_{i+1} \cdot \left\lfloor \frac{A}{B} \right\rfloor_i
\end{align*}
$$

The value $v_0$ gives us the value of the inverse element $B^{-1} \mod A$. The asymptotic complexity of this algorithm is the same as the usual Euclidean algorithm, that is, $O(\log B)$. If we compare the execution time of the forward and backward parts of the algorithm, then the second part runs faster because the multiplication operation has linear complexity with respect to the length of the input numbers, while the division operation with the remainder has the complexity $O(\log L)$ with respect to the length of the input numbers.

2. Calculation of inverse modulo elements using the k-ary GCD Algorithm

We turn further to the k-ary Algorithm and consider formulas similar to formulas (2). Like as Euclidean’s algorithm it runs by iterations.

Let a small positive integer $k$ be chosen. Usually, $k$ is taken equal to a power of 2, $k = 2^s$, this accelerates the whole time of GCD computation.

At each iteration the procedure receives two numbers $A$ and $B$ as input, finds two small numbers $u$ and $v$ such that the following equation holds

$$A_n x_n + B_n y_n = 0,$$

then computes integer $C_n = (Ax + By)/k$, checks if $C$ is even, and divides it by 2 in the opposite case until it becomes odd. Then it forms a new pair $(B, C)$ as the input of the next iteration, if $B > C$, or $(C, B)$, otherwise. It stops when $C$ becomes equal to 0. Consider the last iteration of this algorithm.

Suppose that at step $n$ the equality $C_n = 0$ is reached. From this equality we directly obtain the equality $A_n x_n + B_n y_n = 0$, which gives us $B_n$ as the output value of the k-ary algorithm. Denote this value by $D_n$. We define the values of the parameters $u_n$ and $v_n$ equal to 0 and 1, similarly to the formulas of the extended Euclidean algorithm. This gives equality $A_n u_n + B_n v_n = D_n$ and using induction we can assume that this formula holds for some $0 < i \leq n$, i.e.

$$A_i u_i + B_i v_i = D_i.
$$

We derive the formula for $u_{i-1}, v_{i-1}$. Remind that the transition from step $t - 1$ to step $t$ was performed as a result of the following transform $A_t = B_{t-1}$ and $B_t = C_{t-1}$ where

$$C_{t-1} = (A_{t-1} x_{t-1} + B_{t-1} y_{t-1})/k 2^{s_{t-1}}.
$$

Substituting formulas (4) into (3) we get

$$B_{t-1} u_t + (A_{t-1} x_{t-1} + B_{t-1} y_{t-1})/k 2^{s_{t-1}} = D_t.
$$

The value $v_0$ gives us the value of the inverse element $B^{-1} \mod A$. The asymptotic complexity of this algorithm is the same as the usual Euclidean algorithm, that is, $O(\log B)$. If we compare the execution time of the forward and backward parts of the algorithm, then the second part runs faster because the multiplication operation has linear complexity with respect to the length of the input numbers, while the division operation with the remainder has the complexity $O(\log L)$ with respect to the length of the input numbers.
This gives the system of equations using the Chinese remainder theorem:

\[ A_{i-1}x_{i-1}v_i + B_{i-1}(k2^{s_i-1}u_i + y_{i-1}v_i) = D_i k2^{s_i-1}. \]

Denoting coefficients for \( A_{i-1} \), \( B_{i-1} \) via \( u_{i-1} \) and \( v_{i-1} \), the right part via \( D_{i-1} \) we obtain

\[
\begin{cases}
    u_{i-1} = x_{i-1}v_i \\
    v_{i-1} = k2^{s_i-1}u_i + y_{i-1}v_i \\
    D_{i-1} = D_i k2^{s_i-1}
\end{cases}
\]  

Applying (5) we find values for all \( u_i \) and \( v_i, i \geq 0 \). The last expression will look like

\[ A_0u_0 + B_0v_0 = D_0. \]

The expression in the right part is a power of two

\[ D_0 = D_n k^{n2^{s_0+\ldots+s_{n-1}}} = D_n 2^s, \quad s = nl + s', \]

where \( k = 2^l \), \( n \) is the number of iterations, \( s' = s_0 + s_1 + \ldots + s_{n-1} \) is sum of the powers of 2 by which the parameters \( C_i \) are reduced at iterations \( i, i = 0, 1, \ldots, n-1 \). Dropping index 0 we obtain equations

\[ Au + Bv = D \cdot 2^s \]  

and

\[ B^{-1} \equiv vD^{-1}2^{-s} \pmod{A} \]  

Consider various case that arise when solving equation (6).  
1. \( D = 1 \). Denote by \( A \) element \( 2^{-1} \pmod{A} = (A+1)/2 \). Then,

\[ B^{-1} \pmod{A} = v2^{-s} \pmod{A} = vA^2 \pmod{A}. \]  

Since \( 2^{-1} \pmod{A} = (A+1)/2 \), then \( 2^{-s} \pmod{A} \) can be easily found by the algorithm of fast raising to a power modulo a given number (see [16]).

2. \( D > 1 \), \( GCD(A,D) = 1 \). Denote by \( D_1 \) the inverse \( D_1 = D^{-1} \pmod{A} \). It can be found by few iterations of the extended Euclidean algorithm, since \( D \) takes small values. Then,

\[ B^{-1} \pmod{A} = vD^{-1}2^{-s} \pmod{A} = vD_1A^2 \pmod{A} \]

3. \( D > 1 \), \( d = GCD(A,D) > 1 \), \( d' = GCD(A/d, D) = 1 \).  

In this case, we cannot find \( D^{-1} \pmod{A} \) since integers \( A \) and \( D \) are not co-prime. Divide all parts of equation (6) by integer \( d \) which is a factor of \( A \) and \( D \), but not of \( B \) since \( GCD(A,B) = 1 \) by assumption. We get the equation

\[ A'v + B' = D' \cdot 2^s \]

In this equation, \( GCD(A',D') = 1 \) and \( B'_1 = B^{-1} \pmod{A} \) can be calculated as in the previous case. Here \( A' = A/d \).

Further, we need to find the inverse \( B^{-1} \pmod{D} = D_2 \), again using the extended Euclidean algorithm. This gives the system of equations

\[
\begin{cases}
    B^{-1}_0 \equiv B_1 \pmod{A/d} \\
    B^{-1}_0 \equiv B_2 \pmod{d}
\end{cases}
\]

Using the Chinese Remainder Theorem, one can easily solve the system and find \( B^{-1} \pmod{A} \), since modules \( A/d \) \( u d \) are mutually co-prime.

4. \( D > 1 \), \( d = GCD(A,D) \geq 1 \), \( d' = GCD(A/d, D) > 1 \)

Now modules \( d \) and \( A' = A/d \) are not co-prime, therefore, it is impossible to apply the Chinese remainder theorem. In this case, we will search first \( A^{-1} \pmod{B} \).

Calculation of \( A^{-1} \pmod{B} \) causes cases similar to cases 2 and 3 considered above. Case 2 arises if

\[ GCD(B,D) = 1 \], then the solution is found by the formula

\[ A^{-1} \pmod{B} = uD^{-1}2^{-s} \pmod{A} = uD_12^{-s} \pmod{A} \]  

In case \( GCD(B,D) = d > 1 \), we need to form a system of two equations like (9) which is solved using the Chinese remainder theorem:

\[
\begin{cases}
    A_0^{-1} \equiv A_1 \pmod{B/d} \\
    A_0^{-1} \equiv A_2 \pmod{d}
\end{cases}
\]

Case 4 for \( A^{-1} \pmod{B} \) fails, since otherwise it would mean that both numbers \( A \) and \( B \) are divisible by \( (d')^2 \) but then \( D \) would be divided into \( (d')^2 \), and \( GCD(A,D) \) would be divided into \( (d')^2 \), which violates the assumption \( GCD(A,D) = d' \).
Let the value \( A^{-1} \mod B \) found. Now find \( B^{-1} \mod A \) according to the formula

\[
B^{-1} \mod A \equiv \frac{1 - A_1 A \mod B}{B} \mod A
\]

(12)

Note that the right side of (12) is calculated by direct division.

3. Examples of inverse calculations

In this section, we consider examples of calculating the inverse elements for different cases of the ratio of coefficients in the Bezout equation.

**Example 1.** Let \( A = 193681 \), \( B = 121045 \) and \( k = 16 \) be given. Coefficients \( x \) and \( y \) satisfying

\[
\frac{A}{g_1} + \frac{B}{g_2} = 0
\]

are calculated according to the Approximating Algorithm [1]. This ensures for each step \( i \) inequality \( C_i \leq B_i \) and fast convergence of the algorithm. Parameters of computation are listed in table 2

| Table 2. Calculation of GCD with the Approximating Algorithm. |
|-------------------|-----------------|-----------------|-----------------|-----------------|
| \( i \) | \( A \) | \( B \) | \( x \) | \( y \) | \( C = (A \times B) / k \) | \( z \) | \( r = 2^z \) |
| 0 | 193681 | 121045 | 7 | -11 | 1517 | 1 | 1 |
| 1 | 121045 | 1517 | 7 | -43 | -1 | 1 | - |
| 2 | 1517 | 43 | 13 | -1 | 459 | 1 | |
| 3 | 43 | 1 | | | | | |

The calculation finished at step \( n = 3 \). Extra factors did not appear. Define the values \( u_3 = 0 \), \( v_3 = 1 \). Then calculate the values \( u_i, v_i \), \( i \leq 3 \) by formulas (5).

| Table 3. Reverse run of the algorithm |
|-------------------|-----------------|
| \( i \) | \( u_i \) | \( v_i \) |
| 3 | 0 | 1 |
| 2 | -13 | 459 |
| 3 | -3213 | 256373 |
| 0 | 1794611 | -2871511 |

\( 2^{-1} \mod A = (A + 1) / 2 = 96641, s = nl + s_0 = 3 \times 4 + 0 = 12, 2^{-s} \mod 43077. \)

We substitute \( n \) into system (7) and find \( u \) and \( v \):

\[
\begin{align*}
    u &= u_0 D \mod A = 34432 \\
    v &= (1 - vB) / A = -21519
\end{align*}
\]

Then \( B^{-1} \mod A = u = 34432. \)

**Example 2.** Consider case when the calculation ended by reaching equality

\[
A_0 x_n + B_0 y_n = 0.
\]

В этом случае \( A_0 x_n = -B_0 y_n \), and GCD\((A_0, B_0) = D_n = B_0 / x_n = -A_n y_n > 1. \) The algorithm completes the calculation, returning a response \( D_n \), being an extraneous factor. In such situation defining \( u_n = 0 \), \( v_n = 1 \), we get the equality

\[
A_n u_n + B_n v_n = D_n
\]

Substituting values \( A_n = B_{n-1} \) and \( B_n = C_{n-1} = (A_{n-1} x_{n-1} + B_{n-1} y_{n-1}) / k r_{n-1} \) we get

\[
A_{n-1} x_{n-1} + B_{n-1} y_{n-1} = D_n x_{n-1} k r_{n-1} = D_{n-1},
\]

and

\[
\begin{align*}
    u_{n-1} &= x_{n-1} \\
    v_{n-1} &= y_{n-1}
\end{align*}
\]

(13)

Other values \( u_i, v_i \), \( i < n - 1 \), are calculated by formulas (5).

Consider the example. Let be given \( A = 1936704039 \), \( B = 1210259645 \). We construct the table of the direct run of the approximating algorithm (table 4):

| Table 4. Direct progress of the approximating algorithm |
|-------------------|-----------------|-----------------|-----------------|-----------------|
| \( i \) | \( A \) | \( B \) | \( X \) | \( y \) | \( C = (A \times B) / k \) | \( Z \) | \( r = 2^z \) |
| 0 | 1936704039 | 1210259645 | 11 | - | 45583154 | 1 | 2 |
Number of iterations = 5, \( D_n = 121 \), and the case holds \( A_n x_n + B_n y_n = 0 \). Define \( u_5 = 0 \), \( v_5 = 1 \), and set \( u_3 = 3 \), \( v_3 = -215 \) by the formulas (9). Calculating \( u_i \), \( v_i \), \( i < 4 \), by formulas (4) we finally get \( u = u_0 = 561122539, v = v_0 = -8979299593, \ s = 4 + 4 \cdot 5 = 24 \). This gives the equation

\[
D^{-1} \mod A = 121^{-1} \mod 1936704039 = 1904692402, \quad \text{and} \quad Z^{-s} \mod A = (A+1)^s \mod A = 336252028.
\]

Then we are able to find the Bezout coefficients by the formulas \( v = (v_0 \cdot D) \mod A, u = (1 - vB) / A \):

\[
\begin{align*}
&v = 1879680695 \\
u = (1 - vB) / A = -1174625366
\end{align*}
\]

**Example 3.** Let \( A = 55, B = 13 \), \( u = -1, v = 5, d = 5 \). Then,

\[
Au + Bv = 55 \cdot (-1) + 13 \cdot 5 = 10 = d \cdot 2^4.
\]

Here \( d = \gcd(A, D) = \gcd(55,5) = 5 \neq 1 \), \( d' = \gcd(A/d, d) = 1 \),

This case corresponds to case 3 in the options for completing the calculation. Find \( A' = A/d = 11 \).

\[
B^{-1} \mod A' = B^{-1} \mod 11 = 2^{-1} \mod 11 = 6, \quad B^{-1} \mod D = B^{-1} \mod 5 = 2
\]

Substitute these into system (9):

\[
\begin{align*}
&\frac{B^{-1} \mod 11}{B^{-1} \mod 5} = 2. \\
&\text{Value } B^{-1} \mod A \text{ find using the Garner algorithm:}
\end{align*}
\]

\[
\begin{align*}
&x \mod p = x_1 \\
&x \mod q = x_2
\end{align*}
\]

\[
\frac{x - x_1}{p} \mod q = \frac{p}{q}\mod p = \frac{2}{1} \mod p = 2,
\]

Then

\[
B^{-1} \mod A = 2 + \left( \frac{6 - 2}{5} \mod 11 \right) \cdot 5 = 2 + (4 \cdot 5^{-1} \mod 11) \cdot 5 = 17.
\]

**Example 4.** Consider the equation \( Au + Bv = D^2 \), \( v; e A = 45, B = 7, u = 1, v = -3, D = 3, s = 3, D \cdot 2^s = 24 \). Here \( d = \gcd(A, D) = 3 > 1 \), and \( \gcd(A_1, D) = \gcd(15, 3) = 3 > 1 \).

This example falls under case 4 of the parameter relationship. We will first search \( A^{-1} \mod B \):

\[
A^{-1} \mod B = u(D^{-1} 2^{-s} \mod B) = 1 \cdot (3^{-1} 2^{-3} \mod 7) = 5.
\]

Find further \( B^{-1} \mod A \) by the formula (12):

\[
B^{-1} \mod A = \frac{1 \cdot 5 \cdot 45}{7} \mod A = -32 \mod 45 = 13
\]

**4. Compilation of various algorithm execution options**

Let collect all the options for calculating \( B^{-1} \mod A \) in the table 5 below.

| Option | \( D_n \) | \( \gcd(A, D_n) \) | \( \gcd(A_1, D_n) \) | \( B^{-1} \mod A \) |
|--------|----------|-----------------|-----------------|------------------|
| 1      | 1        | 1               | 1               | \( v2^{-s} \mod A \) |
| 2      | >1       | >1              | >1              | \( v D_n^{-1} 2^{-s} \mod A \) |
| 3      | >1       | >1              | >1              | \( \text{CRT}(A_1, d), A_1 = A/d \) |
| 4      | >1       | >1              | >1              | \( A^{-1} \mod B \) (happening 2) |
| 5      | >1       | >1              | >1              | \( A^{-1} \mod B \) (happening 3) |
5. Speed estimation of the Extended k-ary Algorithm

Let us estimate the number of operations required to calculate the inverse element according to our scheme in comparison with the scheme of the extended Euclidean algorithm. Denote by \( N_d \) and \( N_E \) the number of iterations when calculating the GCD according to the scheme of the approximating algorithm and the Euclidean algorithm, respectively. According to [7] \( N_E \approx 5N_d \) with \( k = 4096 \) and the length of the input numbers up to 3000 bits.

At iteration of the back run of the Euclidean Algorithm two operations are performed:

\[
\begin{align*}
  u_i &= v_{i+1}, \\
  v_i &= u_{i+1} - v_{i+1} \cdot \text{int}(A_i/B_i).
\end{align*}
\]

The first operation is simple assignment, it has a linear complexity with respect to lengths \( A_i \) and \( B_i \). Integer division \( A_i/B_i \) was performed in the main loop of the Euclidean scheme, here it was simply extracted from their previously saved array, therefore, at one iteration of the Euclidean scheme, one operation of two long numbers is performed, which has linear complexity \( O(L) \) regarding length of \( A_i \). Indeed, the attitude \( \text{int}(A_i/B_i) \) takes small values and usually fits in the size of one machine word, and the parameters \( u_i \) and \( v_i \) are bounded by initial numbers \( A \) and \( B \).

We now consider the basic operations on the iteration of the reverse run of the Approximating Algorithm. At each iteration, the calculation of the parameters \( u_i \) and \( v_i \) is performed according to formulas (5):

\[
\begin{align*}
  u_i &= z_i x_i v_{i+1}, \\
  v_i &= z_i y_i v_{i+1} + k r_i u_{i+1}.
\end{align*}
\]

Here three multiplications are performed of arguments \( u_i \) and \( v_i \) by machine word size numbers \( x_i \) and \( y_i \). In other words, the number of operations per iteration is approximately three times larger opposite the number of operations per iteration of the Euclidean scheme. In addition, in the approximating algorithm, it is necessary to find the element inverse modulo \( A \) for the extraneous factor if it appears.

If we take into account that the number of iterations of the approximating algorithm is approximately 5 times less than the number of iterations of the Euclidean scheme, then, in general, the return run of the approximating algorithm is comparable in speed with the return run of the Euclidean Algorithm.

If we compare amount of work at the first (direct) stage of both algorithms then in both case the first stage continues longer than the second one. For example, according to the Euclidean scheme the main operation of one iteration is integer division \( q_i = A_i/B_i \) has complexity at best \( O(L \cdot \ln L) \) relative to the length of the input numbers. Therefore the calculation of inverse elements in finite fields according to the approximating algorithm scheme will be performed 3 to 5 times faster than according to the classical Euclidean scheme.

6. Conclusions

In the article we have derived the basic formulas for efficient programming of the Extended Approximating Algorithm. These formulas allows users to perform operations in finite fields several times faster than by the Extended Euclidean Algorithms.

References

[1] Ishmukhametov ST 2016 J. Math. 37(6): 723-729.
[2] Sorenson J, 1994, J. Alg. 16(1): 110-144.
[3] Jebelean TA 1993 Proc. Algebr. Comp. 93:111-116.
[4] Amer IF 2018 S. T. Manag: 12–15.
[5] Amer I, AlHalidi AM, Ishmukahmetov ST 2019 Nauka i prosvechenie: 10–14.
[6] Amer I, AlHalidi AM, Ishmukahmetov ST 2019 U. Z. K. U., 161: 110-119.
[7] Antonov NA, Ishmukhametov ST, AlHalidi AM, Maiorova ME 2017 4 (13):1108-1125.
[8] Dolgov DA Chebyshevskii Sb 2018, KFU, 19(2): 421–431.
[9] Еникеев RR 2019 Ф.матем. науки. Т. 250–262.
[10] Еникеев RR 2019 Ф.матем. науки.Т. 161(3): 393–404.
[12] Sidi MS 2007, *Info.* 102(6): 247-252.
[13] Ishmukhametov S, Mubarakov BG, Al-Anni KM 2017 *Russian Math. (Iz. VUZ)*, 11: 30-38.