International Journal of Science and Research (IJSR)
ISSN (Online): 2319-7064
Index Copernicus Value (2013): 6.14 | Impact Factor (2013): 4.438

Development of New Algorithm for Finding Inverse of Modular Multiplication

Dr. J. Thirumaran[1], S. Raja[2]

[1]Dean, Rathinam College of Arts & Science, Coimbatore-642021, India

[2]Assistant Professor, Rathinam College of Arts & Science, Coimbatore-642021, India

Abstract: ***The output of division of two integers in most of the cases is not integer in traditional arithmetic. However, in modular arithmetic, (c/d) mod p is either integer if d and p are relatively prime. . Basic Arrays and their Properties are analyzed first, The available algorithms are analyzed and MMI algorithm is proposed. The comparative Analysis of NEA vs. XEA are mad and complexity Analysis of MMI Algorithm is also made.***

Keywords: Modular multiplication, NMI, NEA, XEA, Algorithms

1. Introduction: Division of Two Integers

The output of division of two integers in most of the cases is not integer in traditional arithmetic. However, in modular arithmetic, (c/d) mod p is either integer if d and p are relatively prime, or it does not exist if d and p are not co-prime, i.e., if gcd(d, p) > 1. Analogous case exists in traditional arithmetic: For instance, if dy = 1 and d = 0, then there is no unique y that satisfies
0y = 1.

In this paper, we provide an Enhanced-Euclid algorithm (NEA) that finds for two relatively prime integers p0 and p1 an integer number x, satisfying the equation p_1x mod p_0 = 1. (1.1) This integer x is called a multiplicative inverse of p1 modulo p0 or, for short, a Modular Multiplicative Inverse (MMI). However, if p_0 and p_1 are not relatively prime, then the NEA finds a gcd(p_0 , p_1). The Extended- Euclid algorithm (XEA) (Knuth, 1997) also finds a MMI of p0 and p_1 if gcd(p_0 , p_1) = 1. Otherwise, the XEA finds gcd(p_0 , p_1).

In this paper, we prove a validity of the NEA and provide its analysis. The analysis demonstrates that the NEA is faster than the XEA.

2. Basic Arrays and their Properties

Let us consider five finite integer arrays:
{pi}, {ci}, {tk }, {wk }, {zk }. (2.1)

Definition 2.1. Let {pi} and {ci} be integer arrays defined according to the following generating rules:
given two relatively prime integers p0 and p1 such that p0 > p1 ,
for i ≥ 1 while pi ≥ 2,
do pi+1 := pi−1 mod pi and ci := ×pi−1 /pi∗. (2.2)

Definition 2.2. Let for every k ≥ 1 {tk } be an arbitrary array; let {wk } and {zk } be defined by the following generating rules: if w0 , w1 , z0 and z1 are initially specified,
then for every k ≥ 2,
wk := wk−1 tk−1 + wk−2 and zk := zk−1 tk−1 + zk−2 . (2.3)

Proposition 2.3. Let us consider a sequence of determinants Dk := zk zk−1 , then for every k ≥ 1,
Dk = $(-1)^{k-1}$ D1 . (2.4) Consider Dk and substitute in the left column the values of wk and zk defined in (2.3).
After simplifications, it follows that Dk = −Dk−1 , then this relation, if applied telescopically, implies (2.4).

Proposition 2.4. Let all three arrays {tk }, {wk } and {zk } be integer, and
w0 := 1, z0 := 0, |z1 | := 1.
Proposition 2.3 implies that for every w1$(-1)^{k-1}$ z1 zk is a multiplicative inverse of wk−1 modulo wk . Indeed, since D1 = −z1 , then (2.4) implies that
$-w_k\ z_{kk}\ 1_{k-1} - z\ w = (1-1)^k z$,

Proposition 2.5. If for every 0 ≤ k ≤ r, tk := cr−k , then wk := pr−k , i.e.,
{wk } = {pi}R and {tk } = {ci}R , (2.6)
where the superscript R in (2.6) means that the arrays {ci} and {pi} are written in reverse.
Thus p0 and p1 are seeds that generate the arrays
{pi}, {ci}, {tk } := {cr−k } and {wk } := {pr−k }.

Theorem 2.6. For every k = 1,... , r, (−1)k−1 z1zk is the multiplicative inverse of pr−k+1 modulo pr−k , i.e., if (k is odd and z1 = 1) or (k is even and z1 = −1),
then x := zk else x := pr−k − zk ;
if k := r and z1 = (−1)r−1, then x := zr , i.e., p1 zr = 1 mod p0 . (2.7)
Proof follows from Propositions 2.3–2.5.

3. NEA for MMI

The proposed algorithm uses stack as a data structure. It solves Eq. (1.1).
vars: r, L, M , S, t: all integer numbers, b: boolean,
proc FORWARD :
assign L := p0 , M := p1 , b := 0,
{r := 0, height of the stack, r is used only for the analysis of the algorithm},
repeat t := ×L/M ∗, S := L − M t, b := 1 − b, {r := r + 1}, (3.1)
push t {onto the top of the stack}, L := M, M := S, (3.2)
until S = 1, (if S = 0, then gcd(p0 , p1) = t; no MMI)

proc BACKTRACKING :
assign S := 0; M := (−1)b (by (2.7) in Theorem 2.6),
repeat pop t {from the top of the stack};
L := Mt + S, S := M, M := L, (3.3)
until the stack is empty ; output x := L; {if x < 0, then x := x+p0 }.

Table 3.1: NEA in progress

p1=1973	p0=1777	196	13	1
Stack	1	9	15	—
151	136	15	1	0

Table 3.2: NEA algorithm with even number of columns

2013	1976	37	15	7	1
Stack	1	53	2	2	—
272	267	5	2	1	0

Example 3.1. Let p0 and p1 be relatively prime integers; let us find an integer number x that is a MMI, i.e., satisfying the equation p1 x mod p0 = 1 (1.1). Suppose that p0 = 1777 and p1 = 1973. Table 3.1 shows the algorithm in progress. Since the right-most element in the first row is equal one, hence the MMI exists. The second row stores the stack and the left-most element in the third row is equal to either x, if the number of columns is even, or it is equal to p1 − x if the number of columns is odd. Thus, in this example x =1973 − 151 = 1822. Indeed, 1777 × 1822 mod 1973 = 1.

Example 3.2. Let now p0 = 1976 and p1 = 2013, let us determine an integer x that satisfies Eq. (1.1). Table 3.2 shows the algorithm in progress. Since the number of columns is even, hence x = 272. Indeed, 1976 × 272 mod 2013 = 1.
Notice that the lengths of the stacks are very short in both examples: we need to store only three and four elements, respectively.

4. Complexity Analysis of MMI Algorithm

Let us consider four integer non-negative arrays: {pi} and {ci} as they defined in (2.2), and {qk } and {dk } defined in accordance with the rules:

d_k :=		(4.1)
p_{i+1} :=	q_{k+1} :=	(4.2)

Here {ci} and {dk } are quotients; {pi} and {qk } are remainders. It is clear from (2.2) and (4.2) that pi+1 := pi−1 − pici = pi−1 mod pi . Hence, both arrays {pi} and {qk } are strictly decreasing and all terms of the corresponding arrays {ci} and {dk } are positive integers.

Definition 4.1. {xj }s is a (s + 1)-dimensional vector, consisting of first s + 1 terms of an array x0 , x1 ,..., xj−1 , xj ,.. ., i.e., {xj }s := (x0 , x1 ,... , xs−1 , xs).

Theorem 4.2. Consider {ci}r ≥ 1, {pi}s , {dk }s , {qk }s ≥ 1 and {pi}r ≥ 1. Let p0 = q0 , {ci}s ≤ {dk }s , i.e., for every j = 1,... ,s there is at least one j = l such that cl < dl ; then for every 1 ≤ j ≤ s the following inequalities hold :
if 1 ≤ j ≤ l − 1, then pj ≥ qj else pj > qj . (4.3)

Proof. Assuming that the statement (4.3) holds for every i ≤ j − 1, let us demonstrate by induction that it also holds for i = j.
Consider tj = dj − cj = ×qj−1 /qj ∗− ×pj−1 /pj ∗ ≤ ×pj−1 /qj ∗− ×pj−1 /pj ∗. (4.4) If j ≤ l − 1, then tj ≥ 0 else tj > 0. Hence, (4.4) implies that if j ≤ l − 1, then pj ≥ qj else pj > qj . Since p0 = q0 , therefore, (4.3) holds for every j ≤ s.

Consider a pair of relatively prime seeds p0 and p1 that generates an array {ci}r = 1. Let us also consider another pair of relatively prime seeds p0 and q1 that generates an array {dk }s ≥ 1, i.e., such that not every term is equal to one. Let r and s be the number of steps required respectively to find the MMIs for the first and the second pair using either the XEA or the NEA. This assumption implies that qs = 1. Therefore, by Theorem 2.6 {pi}s ≥ {qk }s and ps > qs = 1. Hence, r > s.

Corollary 4.3. A pair of seeds that is required for a given p0 , which is the maximal number of steps for computation of a MMI, generates an unary array of quotients, where every components in {ci}r = 1. Thus, as it follows from (2.2) and (4.3), this pair of seeds must generate the following array of integer numbers: p2 := p0 − p1 , p3 := p1 − p2,..., pr := pr−2 − pr−1 = 1. For instance, the array of the Fibonacci numbers {Fr+2 , Fr+1 ,... , F4 , F3 , F2 }
generates the former array where for every i = 0,... ,r pi := Fr+2−i

Remark 4.4. The pair p0 = Fr+2 , p1 = Fr+1 is not the only one that generates (a) an unary array of quotients; (b) a decreasing integer array with the rth remainder equal to one.
The following pairs of seeds have the same property {for all non- negative integer numbers t and u}:
1. p0 = Fr+2 +tFr , p1 = Fr+1 +tFr−1, for t = 1, {pi} = {L1, L2 ,..., Lr+1} is a sequence of the Lucas numbers 1, 3, 4, 7, 11, 18,...
2. p0 = tFr+2 + (1 − t)Fr−1 , t ≥ 1, p1 = tFr+1 + (1 − t)Fr−2 .
3. p0 = Fr+1 + tFr , t ≥ 1, p1 = Fr + tFr−1 .
4. p0 = (1 + t)Fr+2 + tFr−2 + uFr , p1 = (1 + t)Fr+1 + tFr−3 + uFr−1 .

Here the Fibonacci numbers with zero and negative indices are computed m−1 with the formula: F−m = (−1) Fm . For all pairs, listed above, exactly r steps are required to find the MMI. However, all these pairs are special cases of a pair of seeds where p0 = bFr + Fr−1 and p1 = bFr−1 + Fr−2 . Then for all 0 ≤ i ≤ r, pi = bFr−i + Fr−i−1 , pr−1 = b and pr = 1.
Let v = (1 − √5)/2 and w = (1 + √5)/2. Using a z-transform approach we deduce that for all 0 ≤ k ≤r
pr−k = [(b − v)w $_k$+ (w − b)v^k √5] (4.5)

Therefore, for a large r
pr−k √5 − (b − v)w^k= (w − b)$(−1)^k$ |v|k−−−→ 0, since |v| < 1. (4.6)
k→r

The relation (4.6) implies that for a large r,
p0 = [w^r (b − v)/5] [1 + o(w)]. (4.7)
Let z := maxb≥2 r(p0 , b).
Then

z ≈ max logw [b≥2 5p0 /(b − v)] = logw [5p0 /(5 − 1)] = ×logw p0 ∗[1 + o(p0)]. (4.8)

From (4.8) it follows that the height of a stack satisfies the following inequality:
r ≤ (×logw p0 ∗)[1 + o(p0)] (Silverman and Tate, 1995). (4.9)

Remark 4.5. Although this upper bound is achievable if p0 = bFr + Fr−1 and p1 = bFr−1 + Fr−2 , for this pair of seeds the MMI can be computed explicitly and is equal to (−1)r−1Fr .

Remark 4.6. If in the RSA public-key encryption (Rivest et al., 1978), p0 = c × 10100 , then r ≤ 100/ log10 w + log10 c or r ≤ 479 + log c. Over a 1,000 computer experiments demonstrated that an average height of the stack is about 40% smaller than the upper bound in (4.9).

5. Extended-Euclid Algorithm (XEA)
XEA finds a multiplicative inverse of p1 modulo p0 provided that gcd(p0 , p1) = 1.
1. (X 1,X 2,X 3) := (1, 0, p0), (Y 1,Y 2,Y 3) := (0, 1, p1),
2. if Y 3 = 0 return X 3 =gcd(p0, p1); no inverse,
3. if Y 3 = 1 return Y 3 =gcd(p0 , p1), the multiplicative inverse Y 2,
4. Q := ×X 3/Y 3∗, (5.1)
5. (T 1,T 2,T 3) := (X 1 − QY 1,X 2 − QY 2,X 3 − QY 3), (5.2)
6. (X 1,X 2,X 3) := (Y 1,Y 2,Y 3), (5.3)
7. (Y 1,Y 2,Y 3) := (T 1,T 2,T 3),
8. goto 2 (Knuth, 1997; Silverman and Tate, 1992). (5.4)

5. Comparative Analysis of NEA vs. XEA

Both algorithms require the same number of steps, r, to compute all quotients: the values of t in the FORWARD procedure in (3.1), and Q in (5.1), respectively. In addition, the NEA requires r more steps in the BACKTRACKING procedure to compute the values of L in (3.3). Thus, the r steps of the XEA require r divisions, 3r multiplications, 3r long algebraic additions and 10r assignments, see (5.1)–(5.4). The XEA uses 10 variables. Yet in both procedures the NEA uses r divisions, 2r multiplications, 2r long additions, 2r stack operations, (push and pop), and 8r assignments, see (3.1)–(3.3). The NEA uses four integer variables, one binary variable and, in addition, O(logw p0) of memory to store the stack. Note that if a MMI does not exist, then there is no necessity to use the BACKTRACKING procedure in the NEA. In this case, the NEA requires even fewer operations than the XEA: one division, one multiplication, one addition, one push operation and five assignments per every step. Yet the XEA still requires the same number of operations per step as in the case if a MMI does exist.

Thus, in overall the XEA uses more multiplications, more additions, more assignments and twice more variables than the proposed algorithm.

6. Conclusion

If both seeds p0 and p1 are chosen randomly, then the probability that gcd(p0 , p1) = 1 is equal 6/π2 = 0.608 (Chesaro, 1881). Let us consider the following notations:

wxea -Worst-case specific complexity (per step) of XEA; wnea -Worst-case specific complexity of NEA; axea -Average-case specific complexity of XEA; anea -Average-case specific complexity of NEA;

td , tm , ta , ts , tst -time complexities of the operations of division, multi- plication, addition, assignment and stack operations push and pop, respectively.

Then,Notice that
Thus,
wxea = td + 3tm + 3ta + 10ts, (7.1)
wnea = td + 2tm + 2ta + 8ts + 2tst , (7.2)
anea = (td + 2tm + 2ta + 8ts + 2tst) × 6/π2
+(td + tm + ta + 5ts + tst) × (1 − 6/π2). (7.3)
axea = wxea , and td ≈ tm ta ≈ ts ≈ tst . (7.4)
R := axea /anea = 2π2 /(3 + π2) = 1.533785. (7.5)
Therefore, the execution of the NEA requires significantly less time than the execution of the XEA.

References

[1] Altintas Y. and Spence A. (1991), End milling force algorithms for CAD systems. Annals of the CIRP 1991; 40(1), 31–34.
[2] Aytek A., (2009), Co-active neurofuzzy interference system for evapotranspiration modelling. In Soft Comput.volume (13), 691–700.
[3] Brown, F., Harris, M.G., and Other, A.N. (1998). Name of paper. In Name(s) of editor(s) (ed.), Name of book in italics, page numbers. Publisher, Place of publication.
[4] Buckley J., Hayashi Y., (1994), Can fuzzy neural networks approximate continuous fuzzy functions? In Fuzzy Sets and Systems, volume (61), 43 52.
[5] Campatellia G., Scippa A. (2012), Prediction of milling cutting force coefficients for Aluminum 6082-T4, Procedia CIRP 1 (2012), 580 – 585.
[6] Castro J. R., Castillo O., and Other, (2009), Universal Approximation of a Class of Interval Type-2 Fuzzy Neural Networks Illustrated with the Case of Non-linear Identification. In IFSA-EUSFLAT, 1382 1387.
[7] Cybenko G., (1989), Approximations by superpositions of sigmoidal functions. In Mathematics of Control, Signals, and Systems, volume (4), 303 314.
[8] Fang N., Wu Q., (2009), A comparative study of the cutting forces in high speed machining of Ti–6Al–4V.