International Journal of Pure and Applied Mathematics
Volume 118 No. 2 2018, 281-290
ISSN: 1311-8080 (printed version); ISSN: 1314-3395 (on-line version)
url: http://www.ijpam.eu
doi: 10.12732/ijpam.v118i2.13

ijpam.eu

A NOTE ON ADAPTATION OF THE KNUTH'S EXTENDED EUCLIDEAN ALGORITHM FOR COMPUTING MULTIPLICATIVE INVERSE

Anton Iliev[1 §], Nikolay Kyurkchiev[2], Asen Rahnev[3]
[1,2,3]Faculty of Mathematics and Informatics
University of Plovdiv Paisii Hilendarski
24, Tzar Asen Str., 4000 Plovdiv, BULGARIA

Abstract: In this note we refresh realization of Adaptation of the Knuth's Extended Euclidean Algorithm for Computing Multiplicative Inverse (EEACMI). The motivation of this work is that this algorithm is used in many directions [48], [32] as well as that this is variant of Euclidean algorithm which is given on many places as the oldest algorithm ever [1]-[27] and [29]-[48]. Internet sources gave many links and forums dealing with similar research of 'greatest common divisor'. In our implementation we reduce the number of iterations and now they are 50% of Adaption of Knuth's realization of EEACMI. For all algorithms we have use the implementations in Visual C# 2017 programming environment.

AMS Subject Classification: 11A05, 68W01
Key Words: greatest common divisor, extended Euclidean greatest common divisor, Knuth's algorithm, multiplicative inverse, reduced number of iterations

1. Introduction

In all implementations we will use as comment in example a = 420748418; b = 9659595. All algorithms work correctly for every a>0 and b>0. Our work is successor of research in [26]-[28]. It computes the multiplicative inverse of a modulo b, $a^{-1} \pmod{b}$, and returns either the inverse as a positive integer less than y, or zero if no inverse exists. The multiplicative inverse of a modulo

Received: 2017-08-23
Revised: 2018-02-04
Published: March 16, 2018

§Correspondence author

b exists if and only if a and b are coprime (i.e., if gcd(a, b) = 1). If the modular multiplicative inverse of a modulo b exists, the operation of division by a modulo b can be defined as multiplying by the inverse. Zero has no modular multiplicative inverse.

For testing we will use the following computer: processor - Intel(R) Core(TM) i7-6700HQ CPU 2.60GHz, 2592 Mhz, 4 Core(s), 8 Logical Processor(s), RAM 16 GB, Microsoft Windows 10 Enterprise x64.

In previous paper [26] we gave new native algorithm for Greatest Common Divisor (GCD). In [27] we gave its recursive variant which is about 30% faster than recursive implementation of Knuth's algorithm [32].

As long as the asymptotic of number of divisions of Knuth's revision of Euclid's GCD is known [32], [38] using CAS Mathematica here we will seek approximation of the data where first coordinate of every point is N and second coordinate is average CPU time in seconds. In the first case we will take the loop from $i = 1$ to N, $a = i$, $b = N + 2 - i$, and in second case we take the loop from $i = 1$ to N, $b = i$, $a = N + 2 - i$. We calculate the time taken by two algorithms and average CPU time taken in these two cases. *Data*1 are data taken from Knuth's algorithm [32] and *data*2 are data which we received from new algorithm [26]. The reader can be convinced of the benefits of the new method [26] (see Fig. 1).

*data*1:={{1000000,0.2255},{2000000,0.397},{3000000,0.619},
{4000000,0.7925},{5000000,0.9855},{6000000,1.2},
{7000000,1.4425},{8000000,1.598},{9000000,1.87},
{10000000,2.066},{11000000,2.26},{12000000,2.482},
{13000000,2.715},{14000000,2.9145},{15000000,3.1605},
{16000000,3.4355},{17000000,3.5565},{18000000,3.8985},
{19000000,4.087},{20000000,4.187},{21000000,4.5255},
{22000000,4.8055},{23000000,4.9525},{24000000,5.2615},
{25000000,5.419},{26000000,5.556},{27000000,5.8255},
{28000000,6.185},{29000000,6.308},{30000000,6.6575},
{31000000,6.8445},{32000000,6.88},{33000000,7.4685},
{34000000,7.5005},{35000000,7.643},{36000000,8.03},
{37000000,8.3145},{38000000,8.323},{39000000,8.7915},
{40000000,9.064},{41000000,8.984},{42000000,9.5235},
{43000000,9.793},{44000000,9.863},{45000000,10.246},
{46000000,10.3745},{47000000,10.6275},{48000000,10.894},
{49000000,11.141},{50000000,11.0995},{51000000,11.547},
{52000000,11.834},{53000000,11.802},{54000000,12.1345},
{55000000,12.397},{56000000,12.6275},{57000000,13.0035},

{58000000,13.3595},{59000000,13.411},{60000000,13.7565},
{61000000,14.0055},{62000000,13.6815},{63000000,14.471},
{64000000,14.6175},{65000000,14.4065},{66000000,15.0455},
{67000000,15.4675},{68000000,15.4285},{69000000,15.5275},
{70000000,16.041},{71000000,15.9805},{72000000,16.6865},
{73000000,16.7475},{74000000,16.7855},{75000000,17.324},
{76000000,17.163},{77000000,17.2245},{78000000,18.08},
{79000000,18.267},{80000000,18.0775},{81000000,18.894},
{82000000,19.192},{83000000,18.549},{84000000,19.622},
{85000000,19.825},{86000000,19.662},{87000000,20.2685},
{88000000,20.2325},{89000000,20.279},{90000000,20.492},
{91000000,21.2085},{92000000,21.0485},{93000000,21.558},
{94000000,21.9795},{95000000,21.81},{96000000,22.409},
{97000000,22.5145},{98000000,22.2405},{99000000,23.3585},
{100000000,23.408}};
*data*2:={{1000000,0.202},{2000000,0.3605},{3000000,0.561},
{4000000,0.7345},{5000000,0.9145},{6000000,1.129},
{7000000,1.343},{8000000,1.499},{9000000,1.734},
{10000000,1.9285},{11000000,2.1015},{12000000,2.311},
{13000000,2.5235},{14000000,2.733},{15000000,2.9545},
{16000000,3.196},{17000000,3.3295},{18000000,3.6475},
{19000000,3.8285},{20000000,3.953},{21000000,4.2535},
{22000000,4.5665},{23000000,4.595},{24000000,4.9295},
{25000000,5.0565},{26000000,5.204},{27000000,5.449},
{28000000,5.792},{29000000,5.8835},{30000000,6.2245},
{31000000,6.3715},{32000000,6.4015},{33000000,6.8585},
{34000000,7.0085},{35000000,7.1175},{36000000,7.4765},
{37000000,7.715},{38000000,7.799},{39000000,8.2745},
{40000000,8.475},{41000000,8.3325},{42000000,8.7855},
{43000000,8.9695},{44000000,9.0745},{45000000,9.473},
{46000000,9.653},{47000000,9.762},{48000000,10.0725},
{49000000,10.3105},{50000000,10.3945},{51000000,10.7625},
{52000000,11.017},{53000000,10.993},{54000000,11.3355},
{55000000,11.58},{56000000,11.78},{57000000,12.0815},
{58000000,12.3235},{59000000,12.346},{60000000,12.7495},
{61000000,12.9755},{62000000,12.751},{63000000,13.3885},
{64000000,13.4925},{65000000,13.348},{66000000,14.002},
{67000000,14.2905},{68000000,14.2775},{69000000,14.401},
{70000000,14.9405},{71000000,14.7915},{72000000,15.3485},

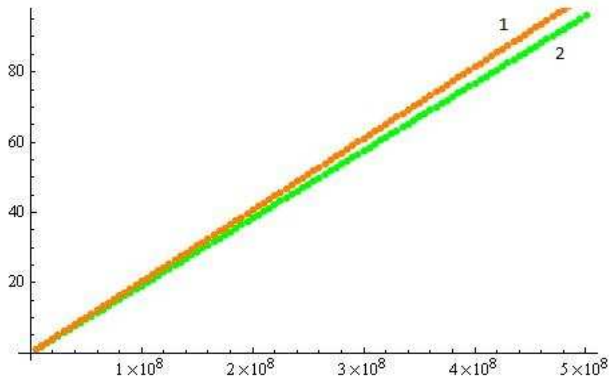

Figure 1: Knuth's algorithm (1 - orange line), Iliev-Kyurkchiev's algorithm (2 - green line)

{73000000,15.5585},{74000000,15.592},{75000000,16.0775},
{76000000,15.9125},{77000000,16.068},{78000000,16.797},
{79000000,16.968},{80000000,16.7855},{81000000,17.449},
{82000000,17.6085},{83000000,17.076},{84000000,18.248},
{85000000,18.4785},{86000000,18.291},{87000000,18.6435},
{88000000,18.7635},{89000000,18.8315},{90000000,19.0805},
{91000000,19.748},{92000000,19.6205},{93000000,20.2985},
{94000000,20.6625},{95000000,20.6285},{96000000,20.8785},
{97000000,20.803},{98000000,20.555},{99000000,21.47},
{100000000,21.6535}};

Adaptation of the Knuth's Extended Euclidean Algorithm for Computing Multiplicative Inverse is given as:

Algorithm 1.

```
//a = 420748418; b = 9659595;
u1 = 1; u3 = a; v1 = 0; v3 = b; iter = 1;
while (v3 != 0)
{ q = u3 / v3; t3 = u3 % v3; t1 = u1 + q * v1;
u1 = v1; v1 = t1; u3 = v3; v3 = t3; iter = -iter; }
if (iter < 0) eeacmi = b - u1; else eeacmi = u1;
if (u3 != 1) eeacmi = 0;
```

which is widely spread via many sources.

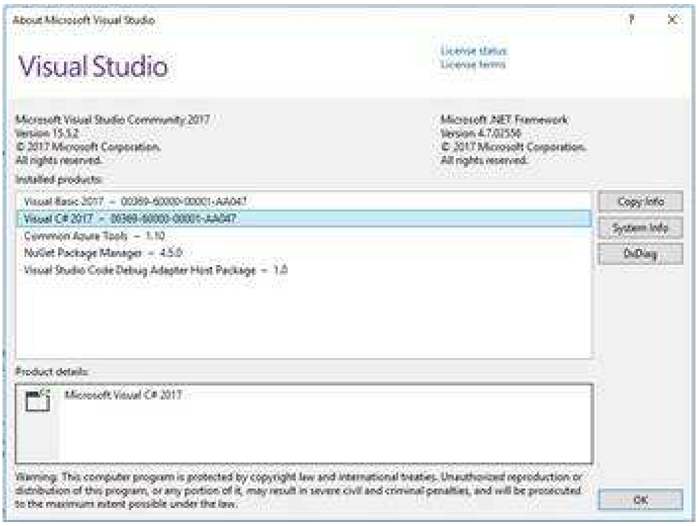

Figure 2: Visual C# 2017

2. Main Results

Now we set the task to optimize Adaptation of the Knuth's EEACMI algorithm with the following programing environment (see Fig. 2.).

We suggest the following iteration processes.

Algorithm 2.

```
iter = 1; //a = 420748418; b = 9659595;
if (a > b) { u1 = 1; u3 = a; v1 = 0; v3 = b;
do { if (v3 < 1) { if (u3 > 1) eeacmi = 0; else
if (iter < 0) eeacmi = b - u1; else eeacmi = u1; break; }
q = u3 / v3; u3 %= v3; t1 = u1 + q * v1; u1 = v1;
v1 = t1; iter = -iter;
if (u3 < 1) { if (v3 > 1) eeacmi = 0; else
if(iter < 0) eeacmi = b - u1; else eeacmi = u1; break; }
q = v3 / u3; v3 %= u3; t1 = u1 + q * v1; u1 = v1;
v1 = t1; iter = -iter; } while (true) ; }
else { u1 = 0; u3 = a; v1 = 1; v3 = b;
do { if (u3 < 1) { if (v3 > 1) eeacmi = 0; else
```

```
if (iter < 0) eeacmi = u1; else eeacmi = b - u1; break; }
q = v3 / u3; v3 %= u3; t1 = u1 + q * v1; u1 = v1;
v1 = t1; iter = -iter;
if (v3 < 1) { if (u3 > 1) eeacmi = 0; else
if (iter < 0) eeacmi = u1; else eeacmi = b - u1; break; }
q = u3 / v3; u3 %= v3; t1 = u1 + q * v1; u1 = v1;
v1 = t1; iter = -iter; } while (true); }
```

The recursive variants of Algorithms 1 and 2 are Algorithms 3 and 4 respectively:

Algorithm 3.

```
static long Euclid(long a, long b, ref long x,
ref long y, ref long iter)
{ if (b < 1) { x = 1; y = 0; return a; }
long q = a / b; long r = a % b; iter = -iter;
long d = Euclid(b, r, ref y, ref x, ref iter);
y += q * x;
return d; }
```

Algorithm 4.

```
static long Euclid(long a, long b, ref long x,
ref long y, ref long iter)
{ long r = a % b; long q1 = a / b; iter = -iter;
if (r < 1) { x = 1; y = 0; return b; }
long u = b % r; long q2 = b / r; iter = -iter;
if (u < 1) { x = q1; y = 1; return r; }
long d = Euclid(r, u, ref x, ref y, ref iter);
y += q2 * x; x += q1 * y;
return d; }
```

The Algorithm 3 can be call with

```
iter = 1;
gcd = Euclid(a, b, ref x, ref y, ref iter);
if (gcd > 1) eeacmi = 0; else
if (iter < 0) eeacmi = bo - x; else eeacmi = x;
```

The Algorithm 4 can be used by

```
iter = 1;
gcd = Euclid(a, b, ref y, ref x, ref iter);
```

```
if (gcd > 1) eeacmi = 0; else
if (iter < 0) eeacmi = bo - x; else eeacmi = x;
```

Note that in all algorithms 'ao' and 'bo' are initial values of 'a' and 'b' respectively.

Numerical experiments.
Part 1. We will use the following task:

```
long a, b, q, eeacmi, iter = 1, u1, v1, u3, v3, t1, t3, gcd, x = 0, y = 0, ao, bo;
//a = 420748418; b = 9659595;
long d;
d = 0;
for (int i = 1; i < 100000001; i++)
{ b = i; a = 200000002 - i; bo = i; ao = 200000002 - i;
//here is the source code of every one of Algorithms 1-4
d += eeacmi; }
Console.WriteLine(d);
```

Results of Part 1.: Time of Algorithm 1: 31.474 sec.;
Time of Algorithm 2: 29.100 sec.;
Time of Algorithm 3: 62.722 sec.;
Time of Algorithm 4: 48.087 sec.

Part 2. We will use the following task where we swapped the values of 'a' and 'b':

```
long a, b, q, eeacmi, iter = 1, u1, v1, u3, v3, t1, t3, gcd, x = 0, y = 0, ao, bo;
//a = 420748418; b = 9659595;
long d;
d = 0;
for (int i = 1; i < 100000001; i++)
{ a = i; b = 200000002 - i; ao = i; bo = 200000002 - i;
//here is the source code of every one of Algorithms 1-4
d += eeacmi; }
Console.WriteLine(d);
```

Results of Part 2.: Time of Algorithm 1: 34.094 sec.;
Time of Algorithm 2: 29.478 sec.;
Time of Algorithm 3: 58.811 sec.;

Time of Algorithm 4: 45.585 sec.

Part 3.
Average time of performance EN = (Part 1.Algorithm N + Part 2.Algorithm N) / 2, where N = 1 to 4 denotes using of Algorithms 1 to 4.
E1 = 33.207 sec.
E2 = 29.289 sec.
E3 = 60.7665 sec.
E4 = 46.836 sec.
So you can see that our new Algorithms 2 and 4 are considerably faster than Algorithms 1 and 3 respectively.

Acknowledgments

This work has been supported by the project FP17-FMI008 of Department for Scientific Research, Paisii Hilendarski University of Plovdiv.

References

[1] A. Aho, J. Hopcroft, J. Ullman, *The Design and Analysis of Computer Algorithms*, 1st ed., Addison-Wesley, Boston (1974).

[2] A. Aho, J. Ullman, J. Hopcroft, *Data Structures and Algorithms*, 1st ed., Addison-Wesley, Boston (1987).

[3] A. Akritas, A new method for computing polynomial greatest common divisors and polynomial remainder sequences, *Numerische Mathematik*, 52 (1988), 119-127.

[4] A. Akritas, G. Malaschonok, P. Vigklas, On the Remainders Obtained in Finding the Greatest Common Divisor of Two Polynomials, *Serdica Journal of Computing*, 9 (2015), 123-138.

[5] M. Alsuwaiyel, *Algorithms: Design Techniques and Analysis*, Lecture Notes Series on Computing, revised ed., World Scientific Publishing Company, Hackensack (2016).

[6] L. Ammeraal, *Algorithms and Data Structures in C++*, John Wiley & Sons Inc., New York (1996).

[7] T. M. Apostol, *Introduction to Analytic Number Theory*, Springer-Verlag, New York (1976).

[8] S. Baase, A. Gelder, *Computer Algorithms, Introduction to Design and Analysis*, 3rd ed., Addison-Wesley, Boston (2000).

[9] G. Brassard, P. Bratley, *Fundamentals of Algorithmics*, international ed., Pearson, (2015).

[10] D. Bressoud, *Factorization and primality testing*, Springer Verlag, New York (1989).

[11] F. Chang, Factoring a Polynomial with Multiple-Roots, *World Academy of Science, Engineering and Technology*, 47 (2008), 492-495.

[12] H. Cohen, *A Course in Computational Algebraic Number Theory*, Springer, New York (1996).

[13] Th. Cormen, *Algorithms Unlocked*, MIT Press, Cambridge (2013).

[14] Th. Cormen, Ch. Leiserson, R. Rivest, Cl. Stein, *Introduction to Algorithms*, 3rd ed., The MIT Press, Cambridge (2009).

[15] J. D. Dixon, The number of steps in the Euclidean algorithm, *J. Number Theory*, 2 (1970), 414-422.

[16] A. Drozdek, *Data Structures and Algorithms in C++*, 4th ed., Cengage Learning, Boston (2013).

[17] J. Erickson, *Algorithms*, University of Illinois Press (2009).

[18] J. Gareth, J. Jones, *Elementary Number Theory*, Springer-Verlag, New York (1998).

[19] K. Garov, A. Rahnev, *Textbook-notes on programming in BASIC for facultative training in mathematics for 9.-10. grade of ESPU*, Sofia (1986). (in Bulgarian)

[20] S. Goldman, K. Goldman, *A Practical Guide to Data Structures and Algorithms Using JAVA*, Chapman & Hall/CRC, Taylor & Francis Group, New York (2008).

[21] A. Golev, *Textbook on algorithms and programs in C#*, University Press "Paisii Hilendarski", Plovdiv (2012).

[22] M. Goodrich, R. Tamassia, D. Mount, *Data Structures and Algorithms in C++*, 2nd ed., John Wiley & Sons Inc., New York (2011).

[23] R. Graham, D. Knuth, O. Patashnik, *Concrete Mathematics: A Foundation for Computer Science*, 2nd ed., Addison-Wesley, Boston (1994).

[24] D. H. Greene, D. E. Knuth, *Mathematics for the Analysis of Algorithms*, 2nd ed., Birkhauser, Boston (1982).

[25] H. A. Heilbronn, On the average length of a class of finite continued fractions. In: *Number Theory and Analysis* (Turan, P., ed.), 87-96, Plenum Press, New York (1969).

[26] A. Iliev, N. Kyurkchiev, A Note on Knuth's implementation of Euclid's Greatest Common Divisor Algorithm, *International Journal of Pure and Applied Mathematics*, 117 (2017), 603-608.

[27] A. Iliev, N. Kyurkchiev, A. Golev, A Note on Knuth's implementation of Extended Euclidean Greatest Common Divisor Algorithm, *International Journal of Pure and Applied Mathematics*, 118 (2018), 31-38.

[28] A. Iliev, N. Valchanov, T. Terzieva, Generalization and Optimization of Some Algorithms, *Collection of scientific works of National Conference "Education in Information Society"*, Plovdiv, ADIS, May 12-13, (2009), 52-58 (in Bulgarian), http://sci-gems.math.bas.bg/jspui/handle/10525/1356

[29] E. Kaltofen, H. Rolletschek, Computing greatest common divisors and factorizations in quadratic number fields, *Math. Comp.*, 53 (1990), 697-720.

[30] J. Kleinberg, E. Tardos, *Algorithm Design*, Addison-Wesley, Boston (2006).

[31] D. E. Knuth, Evaluation of Porters constant, *Comp. Maths. Appls.*, 2 (1976), 137-139.

[32] D. Knuth, *The Art of Computer Programming, Vol. 2, Seminumerical Algorithms*, 3rd ed., Addison-Wesley, Boston (1998).

[33] Hr. Krushkov, *Programming in C#*, Koala press, Plovdiv (2017). (in Bulgarian)

[34] A. Levitin, *Introduction to the Design and Analysis of Algorithms*, 3rd ed., Pearson, Boston (2011).

[35] A. Menezes, P. Oorschot, S. Vanstone, *Handbook of Applied Cryptography*, 5th ed., CRC Press LLC, New York (2001).

[36] P. Nakov, P. Dobrikov, *Programming =++Algorithms*, 5th ed., Sofia (2015). (in Bulgarian)

[37] G. H. Norton, A shift-remainder GCD algorithm. In: Applied Algebra. Algebraic Algorithms and Error Correcting Codes (Huguet, L., Poli, A., eds.), Springer LNCS, 356 (1989), 350-356.

[38] G. H. Norton, On the Asymptotic Analysis of the Euclidean Algorithm, *J. Symbolic Computation*, 10 (1990), 53-58.

[39] J. W. Porter, On a theorem of Heilbronn, *Mathematika*, 22 (1975), 20-28.

[40] A. Rahnev, K. Garov, O. Gavrailov, *Textbook for extracurricular work using BASIC*, MNP Press, Sofia (1985). (in Bulgarian)

[41] A. Rahnev, K. Garov, O. Gavrailov, *BASIC in examples and tasks*, Government Press "Narodna prosveta", Sofia (1990). (in Bulgarian)

[42] H. Rolletschek, On the number of divisions of the Euclidean algorithm applied to Gaussian integers, *J. Symbolic Computation*, 2 (1986), 261-291.

[43] H. Rolletschek, Shortest division chains in imaginary quadratic number fields. In: *Symbolic and Algebraic Computation* (Gianni, P., ed.). Springer LNCS 358 (1990), 231-243.

[44] D. Schmidt, *Euclid's GCD Algorithm* (2014).

[45] R. Sedgewick, K. Wayne, *Algorithms*, 4th ed., Addison-Wesley, Boston (2011).

[46] S. Skiena, *The Algorithm Design Manual*, 2nd ed., Springer, New York (2008).

[47] A. Stepanov, *Notes on Programming* (2007).

[48] E. Weisstein, *CRC Concise Encyclopedia of Mathematics*, Chapman & Hall/CRC, New York (2003).