A NOTE ON KNUTH’S IMPLEMENTATION OF EUCLID’S GREATEST COMMON DIVISOR ALGORITHM

Anton Iliev¹ §, Nikolay Kyurkchiev²

¹,²Faculty of Mathematics and Informatics
University of Plovdiv Paisii Hilendarski
24, Tzar Asen Str., 4000 Plovdiv, BULGARIA

Abstract: In this note we give new and faster natural realization of Euclid’s Greatest Common Divisor (GCD) algorithm. The reason of interest in this topic is widely application of this algorithm in various mathematics and computer science topics [13]. Particularly via Google you can see that there are more than 400 000 pages indexed for keyword ‘greatest common divisor’. In our implementation we reduce the number of iterations and now they are 50% of Knuth’s realization of Euclid’s GCD. For all algorithms we have use the implementations in Visual C# 2017 programming environment.

To the bright memory of Prof. Iliya Iliev

AMS Subject Classification: 11A05, 68W01

Key Words: greatest common divisor, Euclid’s algorithm, Knuth’s algorithm, 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. We will mention that searching of new modifications of classical algorithms is serious task see for example our previous work on number of primes [12].

In his book Knuth [13] proposed the following iteration process:
Algorithm 1.

\[
\text{long } a, b, \text{ob, gcd}; //a = 420748418; b = 9659595;
\]

\[
\text{while } (b > 0) \{ \text{ob} = b; \ b = a \% b; \ a = \text{ob}; \}
\]

\[
gcd = a;
\]

which is the most commonly used and can be seen in many sources and books [3]–[11], [13]–[20].

The following algorithm is given by Schmidt [18]:

Algorithm 2.

\[
\text{long } a, b, \text{gcd}; //a = 420748418; b = 9659595;
\]

\[
\text{while } (a > 0 && b > 0) \text{ if } (a > b) \ a \%= \ b; \text{ else } b \%= a;
\]

\[
gcd = a + b;
\]

The next implementation is given by Stepanov [20]:

Algorithm 3.

\[
\text{long } a, b, \text{gcd}; //a = 420748418; b = 9659595;
\]

\[
\text{while } (\text{true}) \{ \text{if } (b < 1) \{ \text{gcd} = a; \text{ break; } \}
\]

\[
a \%= b; \text{ if } (a < 1) \{ \text{gcd} = b; \text{ break; } \} \ b \%= a; \}
\]

2. Main Results

Now we set the task to optimize all implementations of Euclid’s algorithm. 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 programming environment (see Fig. 1).
Algorithm 4.
long a, b, gcd;
//a = 420748418; b = 9659595;
if (a > b) do { a %= b; if (a < 1) { gcd = b; break; } } b %= a; if (b < 1) { gcd = a; break; } else do { b %= a; if (b < 1) { gcd = a; break; } } a %= b; if (a < 1) { gcd = b; break; } } while (true);

Numerical experiments.

Part 1. We will use the following task:

long d;
d = 0;
for (int i = 1; i < 1000000001; i++) { b = i; a = 2000000002 - i;
//here is the source code of every one of Algorithms 1-4
d += gcd; } Console.WriteLine(d);
Results from Algorithms 1-4 (see Fig. 2 - Fig. 5).
Figure 2: Algorithm 1. \( d = 38332157136 \), time: 4 min. 10.016 sec.

Figure 3: Algorithm 2. \( d = 38332157136 \), time: 4 min. 15.994 sec.

Figure 4: Algorithm 3. \( d = 38332157136 \), time: 4 min. 2.534 sec.

Figure 5: Algorithm 4. \( d = 38332157136 \), time: 3 min. 58.914 sec.

Figure 6: Algorithm 1. \( d = 38332157136 \), time: 4 min. 31.401 sec.

Figure 7: Algorithm 2. \( d = 38332157136 \), time: 4 min. 17.606 sec.

Figure 8: Algorithm 3. \( d = 38332157136 \), time: 4 min. 17.366 sec.
Part 2. We will use the following task where we swapped the values of ‘a’ and ‘b’ from Part 1:

```csharp
long d;
d = 0;
for (int i = 1; i < 1000000001; i++) {
    a = i; b = 2000000002 - i;
    //here is the source code of every one of Algorithms 1-4
    d += gcd;
}
Console.WriteLine(d);
```

Results from Algorithms 1-4 (see Fig. 6 - Fig. 9).

Part 3. Average time of performance

\[ EN = \frac{(\text{Part 1.Algorithm}\_N + \text{Part 2.Algorithm}\_N)}{2}, \]

where \( N = 1 \) to 4 denotes using of Algorithms 1 to 4.

- \( E_1 = 4 \text{ min. } 20.713 \text{ sec.} \)
- \( E_2 = 4 \text{ min. } 16.800 \text{ sec.} \)
- \( E_3 = 4 \text{ min. } 9.950 \text{ sec.} \)
- \( E_4 = 4 \text{ min. } 1.898 \text{ sec.} \)

So you can see that our new Algorithm 4 is faster than all others. This modification can be used for polynomial factorization [1], [5] and [2].

Acknowledgments

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

References


