Oct 27th, 10:45 AM - 12:15 PM

Efficiency of calculating GCD through computer with help of modified Euclidean algorithm

Emin Emini
University for Business and Technology, ee37811@ubt-uni.net

Azir Jusufi
University for Business and Technology, azir.jusufi@ubt-uni.net

Ruhada Emini
University for Business and Technology, re30018@ubt-uni.net

Efficiency of calculating GCD through computer with help of modified Euclidean algorithm

Emin Emini[1], Azir Jusufi[2], Ruhada Emini[3]

[1] Student of Computer Science and Engineering, Bachelor Degree ee37811@ubt-uni.net

[2] Professor of Mathematics and Discrete Structures azir.jusufi@ubt-uni.net

[3] Student of Computer Science and Engineering, Master Degree re30018@ubt-uni.net

Abstract. Euclid's algorithm remains to be the most efficient method for finding the greatest common divisor of two full numbers. This method for finding the greatest common division of two positive integers has been analysed radically for ages. Almost all mathematical segments use the GCD and the Euclid algorithm. If this algorithm is not applied, that means that the actual segment is not investigated in depth. This research paper is going to present a program made with Swift, as the most effective and modern programming language, which has improved the actual existing application for finding GCD. By using the meaning of congruences this research paper will make a modification in the note of Euclid's algorithm by removing some not important steps, but doing the modulation and simplification of every step. Here are discussed both classic and modified methods, by putting appropriate codes and observe the duration of the calculation through the computer.

Keywords: Algorithm, Euclid, GCD, modification, congruence.

1 Introduction

Nowadays we are living in the age of computer where all the sciences are connected with mathematics. The mathematics itself is divided into algebra and geometry. Almost every part of algebra is dependent from greatest common division (GCD) which uses Euclidean algorithm [3]. But not only algebra needs GCD, also sometimes the algorithm is used in geometry. Before using this method you have to know how to divide. When there are no more digits to divide, the final difference is the remainder. Mathematically, the greatest common divisor of two integers, is the largest integer that divides both two integers. In the sections below we are going to discuss about

GCD and its types, then the Euclidean algorithm will be explained in depth and in the end we are going to present our method of calculating GCD.

2 The Greatest Common Divisor (GCD) - Euclid’s Algorithm

There are three main methods to find GCD [1]. The first one is the easy method of inspection which is applied between two numbers a and b by finding the one number which divides both of them and so on until the result is found. The other method is prime factorization method. In this method the first step is to break each number into the prime factorization and then define all the factors that they have in common and after that by multiplying these together we find the GCD. And the las one is the method in which we were focused the most, it is the Euclidean algorithm method [2], which performs division first from smaller then to the larger of two numbers, followed by the reminder, until the reminder is zero.
This algorithm has been studied since the Gaussian time and nowadays with the development of new sciences the focus on this algorithm has grown [4]. In this research we are focused on the Euclidean algorithm for finding the greatest common division by simplifying the Euclidean algorithm with modulation. The GCD is described and defined in the mathematical way:

Definition 1.1 The largest common division of integers !1 2 n is called the largest natural number that completes each of the given numbers.
The greatest common division of numbers !1 2 n is symbolically marked with

!$(a_1, a_2, \ldots, a_n)$.

Definition 1.2 Two full numbers !1 and ! 2 are called simple between each other or mutually simple, if ! 1 2 n.

Theorem 1.1 *(The subdivision algorithm)* We have a given natural number b!. Each integer ! can only be presented in the form:

$$!a = bq + r \qquad (1)$$

Where ! and ! are integers and !$r = (1,2,3,\ldots, b-1)$.

In the section below we are going to prove that the equation $a! = bq + r$ appears always in this form. Where a is the first number of the algorithm for finding GCD, and b is the second number. In this case ! is called the quotient while! the remainder during divide of the number ! with !b.

Proof:

We mark **!**bq with the largest multiple of **!**b that does not exceed **!** , then we will have:

$$!bq \leq a < (q+1)b$$

Consequently, ! will be equal to one of the numbers:

$$!bq, bq+1, bq+2, \dots, bq+(b-1)$$

So, " can appear in the form !$a = bq + r$.

Let's show that the appearance of the number ! in the form $a! = bq + r$ is single. Suppose that the number " can also appear in the form

$$!a = bq1 + r1 \qquad (2)$$

Where ! 1 and !1 are integers and !$r1 = (0,1,2,\ldots,b-1)$.

From (1) and (2) we have:

$$!bq + r = bq1 + r1$$

$$!b(q - q1) = r - r1$$

Therefore ! 1 is multiple of !b, but since ! $r - r1 | < b$ and ! 1 is multiple of !b, then we must have !$r - r1 = 0$, ! 1 consequently ! 1.

In this way we provide the existence and uniqueness of the appearance of the number ! in the form !$a = bq + r$.

Example 1.1 Let it be !$b = 4$. Now we have:

!	$16 = 4 \times 4 + 0$	$0 = 0 < 4$
	$! 3 = 4 \times 3 + 1$	$0 < 1 < 4$
	$!-10 = 4 \times (-3) + 2$	$0 < 2 < 4$

There are several methods to find the GCD of two numbers, one of which is the Euclid algorithm.

Based on the *Theorem 1.1*, during division of the number ! and !b we have:

$$!a = bq_1 + r_2 \qquad !0 \le r_2 < b$$

where ! 1 is the queue and !2 is the remaining partition of the number ! with !b.

If !$r_2 = 0$ process is considered completed, if !$r_2 \neq 0$ we divide !b with !2 and we have:

$$!b = r_2q_2 + r_3 \qquad !0 \le r_3 < r_2$$

If !$r_3 = 0$ process is considered completed, if !$r_3 \neq 0$ we divide ! 2 with !3 and we have:

$$!2 = r_3q_3 + r_4 \qquad !0 \le r_4 < r_3 \text{ etc.}$$

It continues in this way until the residue becomes zero. The final reconciliation of this process will be:

! $_{n-1}$ $_{n}$ $_{n}$

In this way, from the above process, we draw the reconciliations:

$$\begin{array}{ll}
!a = bq_1 + r_2 & !0 \le r_2 < b \\
!b = r_2q_2 + r_3 & !0 \le r_3 < r_2 \\
!2 = r_3q_3 + r_4 & !0 \le r_4 < r_3 \\
\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots & \ldots\ldots\ldots\ldots\ldots \\
!_{n-2} = r_{n-1}q_{n-1} + r_n & !0 \le r_n < r_{n-1} \\
!_{n-1n\ n} &
\end{array} \qquad (1)$$

Let us consider equalization (1). While taking in consideration given *Theorem 1.1.1* and *Theorem 1.1.2*, it is easy to notice that the common divisor of numbers ! and !b matches with the common divisor of numbers b! and !2, likewise they are common divisors of numbers !2 and !3, numbers !3 and !4 , numbers !$n-1$ and ! n, finally with the divisors of the number ! n

In addition we have:

$$!\ a\ , b) = (b, r_2) = \ldots = (r_{n-1}, r_n) = r_n$$

So, !n is the last remaining different from zero and basically this is the largest common divisor of numbers ! and !b.

Example 1.2 by using the **Euclid'**s algorithm, find the largest common divisor of numbers !520 and !125.

!

$$!520 = 125 \cdot 4 + 20$$
$$125 = 20 \cdot 6 + 5$$
$$!20 = 5 \cdot 4 + 0$$

The last residue different from zero is 5. Therefore, the largest common divisor of numbers 520 and 125 is 5.

Theorem 1.2 ([1], page 12). To find the largest common divisor of numbers

!
1 2 n we calculate the following:

$!(a_1, a_2) = d_2$, $!(d_2, a_3) = d_3$, …, $!(d_{n-1}, a_n) = d_n$, so $!(a_1, a_2, …, a_n) = d_n$

Example 1.3 Find the largest common divisor of numbers:

"0,14,32,98.

!(10,14) = 2, !(2,32) = 2, !(2,98) = 2, so!(10,14,32,98) = 2

3 New Proposal - Modifying the Euclidean algorithm

As this algorithm is one of the most discussed algorithms of mathematics it also has many theoretical and practical applications. One of its approaches is a key element of the RSA algorithm, a public-key encryption method used in e-commerce. The modification is based on the use of remainder, where appropriate, which can reduce the number of iteration steps substantially. There are modified Euclidean algorithms which extend it to find the result but with lest steps to complete it. If we consider the following system of modular equations

$$"x \equiv c1(m\ od\ b1), "x \equiv c\ 2(m\ od\ b\ 2), …, "x \equiv cn(m\ od\ bn).$$

Where "bi and "ci are integers, and "bi, "i= 1,2,...,n are pairwise relatively prime [5]. The result is used to solve the system of linear modular equations in one variable.

Definition 2.1. Let it be "$m \in \mathbb{N}$ and "$\forall a,b \in \mathbb{Z}$, we say that " is "$b$ congruent with the module " then and only then if "$m \mid (a - b)$.
If " is congruent with "b based on the module, this fact symbolically is marked:

$$"a \equiv b \pmod{m}$$

By using the congruence we make a modification in the presentation of the Euclid's algorithm.

Let suppose that "$a, b \in N$ and "$a > b$.

1. We calculate "$a \equiv r_1 (m\ od\ b)$, where "$0 \le r_1 < b$, if "$r_1 > 0$, if, then

2. We calculate "$b \equiv r_2 (m\ od r_1)$, where "$0 \le r_2 < r_1$ if "$r_2 > 0$, then

3. We calculate "$r_1 \equiv r_3 (m\ od r_2)$, where "$0 \le r_3 < r_2$ if "$r_3 > 0$, then

4. We calculate "$r_2 \equiv r_4 (m\ od r_3)$, where "$0 \le r_4 < r_3$ if "$r_4 > 0$, then

....

5. We calculate $r"_{k-2} \equiv r_k (m\ od r_{k-1})$, where $0" \le r_k < r_{k-1}$, since the residues are coming down, then
There is the moment when we take "$r_{k-1} \equiv 0 (m\ od r_k\ )$.

The last remaining different from zero, so "k is GCD of numbers " and "b.

Example 2.1: Calculate GCD (3768, 1701)

$3768 \equiv 366 \pmod{1701}$

$1701 \equiv 237 \pmod{366}$
$366 \equiv 129 \pmod{237}$
$237 \equiv 108 \pmod{129}$
$129 \equiv 21 \pmod{108}$
$108 \equiv 3 \pmod{21}$
$21 \equiv 0 \pmod{3}$

So, GCD (3768, 1701) = 3

3.1 Test and Results

Pseudo-code: The next section will explain the code that enables the GCD calculation. As shown below, we have three different functions for computation and reaching to the result, where in Figure 1 there is a function without recursion, in the second with recursion, whereas the third represents the function of the Euclid's algorithm step by step.

```
func gcdModified(a: Int, b: Int) -> Int
    { var a = abs(a)
    var b = abs(b)

    repeat {
        let x = a % b
        a = b
        b = x
    } while (b > 0)

    return a
}
```

Fig. 1 - Function for GCD without recursion

From *Fig. 1* we see that a function (method) is created, which returns the integer value and accepts two parameters a and b, which are also integer value. Then two variables a and b are declared which values take from the function parameters by attaching the absolute value that we use in this case as: *var a = abs(a)*.

Further, a loop that is repeated is the variable b to reach the value 0. Within the loop overwrite the value of variables a and b, where in the first case the variable a receives the value of the variable b, whereas the variable b is equal to the module of variables a of b. Once the loop is initialized, the result is returned which will be stored in the variable a and at the end the function returns the value of the result through the variable a.

```
func gcdRecursion(a: Int, b: Int) -> Int
    { if b == 0 {
        return a
    }
    return gcdRecursion(a: b, b: a % b)
}
```

Fig. 2 - Function of GCD with recursion

Fig. 2 presents the calculation of GCD with recursive method, which also in its function contains two variables a and b which are integer type and also the whole function returns integer value. Then an if statement is created, in which condition we have the equation b with the zero value, so this condition will only occur when b is the value 0. If this condition is valid then no additional calculations are performed and the function ends by returning the variable a. But if the condition is not met, then the function with different parameters or variables is rewritten by assigning the parameter a to the parameter value b, while parameter b modulating a and b.

The function shown in *Figure 3* as mentioned above is the introduction of the GCD result through the Euclid's algorithm, which also initially creates a method that returns the integer value and also accepts two parameters a and b which are equal to the integer value. Furthermore, the variables a and b are stated, which are taken from the function. The same loop is used as in the above function which repeats until b gets the value 0. The loop contains a variable that gains the value by dividing a with b and another variable r whose value is derived from the residue of a and b.

Then the inscription of the variable a with the variable b, and the variable b with that r. The result of the function is also stored in the variable a which is given once the function is executed.

```
func gcdEuclid(a: Int, b: Int) -> Int
   { var a = abs(a)
   var b = abs(b)

   repeat {
       let q: Int = a / b
       let r: Int = a - (b * q)

       a = b
       b = r
   } while (b > 0)

   return a
}
```

Fig. 3 - Euclid's Algorithm for calculating GCD

Table 1. Classical Euclid's Algorithm and modified one for finding GCD.

GCD	Tens Input	Hundrends Input	Thousands Input	Longer Input
Classical	0.001507 sec	0.001485 sec	0.001376 sec	0.012106 sec
Modified	0.000218 sec	0.000420 sec	0.000389 sec	0.000781 sec

Table 1 presents the calculation time of GCD classical Euclidean algorithm and modified one. According to the table we see that the modified algorithm is faster up to 15 times than the classical one. In this example are used different inputs from the small values until the bigger ones. From this experiment the classical method takes more time to execute the result, while the modified one reminds almost the same, or increases just for 0.0004 sec.

Programmed in *Swift* language with *CoreFoundation* and *UIKit* library. Tested in Apple MacBook 2013 Early, with performance:
Processor: 2,4 GHz Intel Core i7
Memory: 8 GB 1600 MHz DDR3

4 Conclusion

The focus of this dissertation was on one of the most famous algorithms used to find the greatest common divisor. It is our hope that this paper has been of interest to teachers of mathematics and to students of mathematics, computer science and other sciences where the algorithm is applied.
The introduction to the Euclidean algorithm should be of interest as it is an easy and most usable way to find the greatest common divisor.
Computer programme (in Swift) has been written to calculate the greatest common divisors of two integers by the Euclidean algorithm. By using the congruences this research paper has made a brief explanation of modified Euclid's algorithm by removing some not important parts of it but doing the modulation and simplification of every step.

References

1. Cohen, E. Arithmetical Functions of a Greatest Common Divisor, III. Cesàro's Divisor Problem. *Proceedings of the Glasgow Mathematical Association*, 5(02), p.67, 1961
2. Altarawneh, H. A Comparison of Several Greatest Common Divisor 'GCD' Algorithms. *International Journal of Computer Applications*, 26(5), pp.24-31, 2011
3. Peck, J. Algorithm 237: Greatest common divisor. *Communications of the ACM*, 7(8), p.481, 1964
4. Gathern, J. V. and Gerhard, J. Modern Computer Algebra. Cambridge University Press, second edition, 2003.
5. I. Z. Milovanovic, C. B. Dolicanin, M. K. Stojcev, E. I. Milovanovic, Modification of Euclidian Algorithm for Solving Modular Equations vol. 4, 2, 41-44, 2012

Bibliography

1. William Stein, Elementary Number Theory: Primes, Congruences, and Secrets, 2017
2. Kenneth H. Rosen, Elementary Number Theory and lts Applications, ADDISON-WESLEY PUBLISHING COMPANY, 1986
3. Lars-˚Ake Lindahl, Lectures on Number Theory, 2002
4. J.S. Milne, Algebraic Number Theory, 2017
5. J. Dine, K. Hila, Teoria e numrave, 2015
6. Apple Inc., The Swift Programming Language, 2014