Overview and Implementation of Euclideans' Algorithm in Java.

Adeniran Adetunji
Physics Department, The Polytechnic Ibadan.
Ibadan, Nigeria.
Tunji4physics@gmail.com

Abstract Discussing computer applications and the sensitivity of the information that are continually exchanged across network and internet works of computers. We provide and overview of the theory behind secure exchange of this information, our discussion is particularly centered on ***Euclidean's' Algorithm*** an ancient but efficient theory. We discussed the theoretical concept behind the algorithm and implements with Java. We conclude by highlighting the various applications of the many techniques in number theory which includes *Euclidean's' Algorithm* in securing information and transactions
over the network.

Key words: Algorithm, Eculideans' Algorithm, Number Theory.

I. INTRODUCTION

Computers today are used for a multitude of sensitive applications. Customers utilize electronic commerce to make purchase and pay their bills. Businesses use the Internet to share sensitive company documents and interact with business partners. And academic institutions use networks of computers to store personal information about students and their grades. Such sensitive information can be potential damaging if it is altered, destroyed, or fall into the wrong hands. There are several powerful algorithmic techniques for protecting sensitive information. Many of the techniques for achieving these goals utilize ***number theory***. An important *number theory* concept is the ancient, yet surprisingly efficient, *Euclid's algorithm* for computing *greatest common divisors*. The paper discusses the concept of *Euclid's algorithm* and this algorithm is implemented in Java.

Our objective is to provide an overview of the concept and its practical implements in a Java program.

II. DEFININING EUCLIDEAN ALGORITHM

Firstly we may want to answer the question; what is *algorithm*? An ***algorithm*** is a step by step process (or recipe) of doing something.

The Euclidean algorithm (which comes down to us from Euclid's *Elements)* computes the greatest common divisor of two given integer. The greatest common divisor *gcd(m,n)* of two non-negative integers *m* and (not both zero) is defined as the larges positive integer that divides both *m* and *n*. If m>0 and n = 0 then *gcd(m,n)* =*m*. If m,n>0, then *gcd(m,n)* can be computed by finding the prime factorization of m and n and by determining the product of all common factors. *Euclidean algorithm* from about 300BC generally performs this operation much faster[2].

III. DIVISIBILITY AND GREATEST COMMON DIVISOR

To set some notation and to indicate its properties, we will provide a brief review of divisibility among integers and later provide that of the *Euclidean Algorithm*.

Divisibility relation:

When *a* and *b* are integers, we say *a divides b* if $b = ak$ *(for some* $k \in Z$*). Z denoting the set of integers* We then write $a|b$ (read as "*a* divides *b*").

For example: we have 2|6 (beacuse 6=2x3), 4|(-12), and 5|0. We have $\pm 1|b$ for every $b \in Z$. However, 6 does not divide 2 and 0 does not divide 5. Divisiblity is a relation, much like inequalities. In particular, the relation 2|6 is *not* the number 3,

even though 6 = 2x3. Such an error will be similar to the mistake of confusing the relation 5<9 with the number 9-5.

Divisibility Theorem:

The following three theorems about divisibility are simple applications of the definition of divisiblity.

Theorem 1: *Let a, b ϵ Z with a|b. Then a|bc for any c ϵ Z.*

Proof: Since a|b by our definition of divisibility, we have b = ak for some k ϵ Z. Therefore *bc = (ak)c = a(kc) and kc ϵ Z So a|bc. QED.*

This just implies that a factor of a number is a factor of any multiple of it. (or, equivalently, a multiple of a multiple is a multiple).

Theorem 2: If a|b and b|c then a|c.

Proof: Since a|b and b|c, then we have a = bk and and c = bl for some k, l ϵ Z. Then

$$C = bl = (ak)l = a(kl) \text{ and } kl \in Z, \text{ so } a|c. \text{ QED.}$$

Meaning, "a factor of a factor is a factor".

Theorem3: If a|b and a|c then a|(br+cs) for every r and s in Z. In particular, if a|b and a|c then a|(b+c) and a|(b-c).

Proof: we have $b = ak$ and $c = al$ for some $k,l \in Z$. Then

$$br + cs = akr + als = a(kr+ls)$$

and $(kr + ls) \in Z$, so $a|(br+cs)$.

Using the language of linea algebra, Theorem 3 says any factor of two integers is also a factor of any Z-linear combination of the two integers.

Greatest common divisor

For two nonzero integers a and b, their greatest common divisor is the largest integer which is a factor of both of them. It is denoted gcd(a,b). For instance, gcd(12,18) = 6 and gcd(-9, 15) = 3. The number 1 is always a common divisor, and it is the greatest common divisor exactly when a and b a relatively prime. The naive method of finding the greatest common divisor of two integers is to factor each into primes and extract the greates common divisor from the prime power factors that appear.

For example: gcd (36, 90), consider;

$a = 36 = 2^2x3^2$

$b = 90 = 2x3^2x5$

$$gcd(36,90) = 2 \times 3^2 = 18$$

Example 2. *gcd(19088597, 39083)*

$a = 19088597 = 11^2x19^3x23$

$b = 39083 = 11^2x17x19$

$gcd(19088597, 39083) = 11^2 \times 19 = 2299.$

This factoring method is hard (even on a computer when the integer has several hundred digits), so this method of computing gcd(a,b) is not good when a and b are large. A method of computing gcd(a,b) which voids the need to factor at all is the *Euclid* which brought us to the focus of this paper, ***Euclidean's' algorithm***.

IV. EUCLIDEANS ALGORITHM

Theorem 4 (Euclid): *Let a and b be nonzero integers. Divide b into a and carry out further divisions according to the following method, where the older remainder becomes the new divisor.*

$$a = b.q_1 + r_1, \quad 0 \le r_1 < |b|,$$
$$b = r_1.q_2 + r_2, \quad 0 \le r_2 < r_1$$
$$r_1 = r_2.q_3 + r_3, \quad 0 \le r_3 < r_2,$$
$$\vdots$$

The non-negative remainder r1, r2, . . . are strictly decreasing, and thus must eventually become 0. *The last nonzero remainder is the greatest common divisor.*

This algorithm of Euclid for finding gcd(a,b) can be carried out very rapidly on a computer even for very large integers which are not easy to factor into primes.

Proof: The key idea of Euclidean's' algorithm work is this:

If a = b +mk for some k ϵ Z, then (a,m) = (b,m). That is, two numbers whose difference is a multiple of *m* have the same gcd with *m*. That is, two numbers whose difference is a multiple of *m* have the same gcd with *m*. Indeed, any common divisor of *a* and *m* is a divisor of $b = a - mk$, and therefore is a common divisor of *b* and *m*. This shows that (a,m) ≤ (b,m). Similarly, any common divisor of *b* and *m* is a divisor of $a = b$

+ mk, and therefore is a common divisor of a and m. Thus (b,m) ≤ (a,m) too, so $(a,m) = (b,m)$.

$m|(a-b) \rightarrow (a,m)=(b,m)$ ..... (1)

Now back to our ***Euclid's algorithm***:

$$a = b.q_1 + r_1, \quad 0 \le r_1 < |b|.$$
$$b = r_1.q_2 + r_2, \quad 0 \le r_2 < r_1$$
$$r_1 = r_2.q_3 + r_3, \quad 0 \le r_3 < r_2,$$
. . .
. . .
. . .

The first equation says $b|(a-r1)$, so by (1) we have $(a,b) = (r1,b)$. The second equation say r1|(b-r2), so again by (1) we have $(b, r1) = (r2, r1)$. The third equation says $r2|(r1-r3)$, so again by (1) we have $(r1, r2) = (r3, r2)$. Comparing these results,

$$(a,b) = (b,r1)=(r1,r2)=(r2,r3),$$

So the later greatest common divisors continue to be equal to (a,b). The last equation in Euclid's algorithm is:

$rn = rn+1qn+2+rn+2, \ 0<rn+2<rn+1,$

$rn+1 = rn+2qn+3 + 0.$

Thus

$(a,b) = (b,r1) = \ldots = (rn, rn+1) = (rn+1, rn+2)$

The final equation in Euclid's algorithm tells us $(rn+1, rn+2) = rn+2$, so (a,b) equals $rn+2$, which is the last nonzero remainder.

V JAVA IMPLEMENTATION

```
package euclideanalgorithm;
import java.util.Scanner;

/**
 *
 * @author Adetunji
 */
public class Euclidean Algorithm {

    public static void main(String[] args) {

        Scanner keyboard1 = new Scanner(System.in);
        Scanner keyboard2 = new Scanner(System.in);

        System.out.println("Enter first Interger"+" "+"a"+":");
        int a1= keyboard1.nextInt();
 System.out.println("Enter Second Interger"+" "+
"b"+":");
        int b1=keyboard2.nextInt();

        int a2 = Math.abs(a1); // This code takes care
        int b2= Math.abs(b1);  // negative integers

        int a; int b; int Ia; int Ib;
        int lastR=0;
        int result=0;
        int r=0;

        if (a2>=b2){

          a = a2;
          b = b2;

        }else {

          a = b2;
          b=a2;
        }

        Ia=a;
        Ib = b;

        System.out.println("a"+"="+a);
        System.out.println("b"+ "="+ b);

/** The above codes assign the largest number *between
the two whose greatest common divisor we *want to find
to "a" and the other number is assigned *to "b". The code
ensures that "a" is greater than or
* equal to "b".
*/

if (a%b==0){

result = b; // Since a>=b, b|a iff a%b==0, if b|a then
gcd(a,b)=0.

    } else {

r = a%b;   /**if a%b!=0, then r=a%b, for a=b.q+r,
            * for some integer q.
           */
    }
        while (r>0){
```

```
        a = b;
        lastR = b;
        b = r;
        r = a%b;
        }

/** The while loop check if r>0 for a=b.q+r.
* From Euclidean Algorithm; gcd(a,b)=gcd(b,r), i.e in
* code does this by assigning the value of "b" to "a" and
* "r" to "b". Repeats thi
* until the condition r>0 is false, i.e r=0.
* a = b.q1+r1
* b = r1.q2+r2
* r1=r2.q3+r3
* r2=r3.q4+r4 ....
* rn-1=(rn.qn+1)+ rn+1. where rn+1=0, then the loop
* terminates
* and proceeds to the next command.
*/

result = (a*b)/a;  /** since a = b.q+r, now r = 0;
                    * and gcd(a,b)=b. To determine b.
                    *  we know that q = a/b,
                    * Therefore a = b.(a/b),
                    * Then b = (a*b)/a.
                    * Assign b to integer "result"
                    */

      //display result

System.out.println("gcd"+"("+a1+","+b1+")"+"="+result);

    }
  }
```

V. PROGRAM OUTPUT IN NETBEANS IDE 7.0

```
1.      run:
Enter first Integer a:
1148
Enter Second Integer b:
-124
a=1148
b=124
gcd(1148,-124)=4
BUILD SUCCESSFUL (total time: 19 seconds)

2.      run:
Enter first Integer a:
19088597
Enter Second Integer b:
39083
a=19088597
b=39083
gcd(19088597,39083)=2299
BUILD SUCCESSFUL (total time: 43 seconds)
```

VI. APPLICATIONS OF EUCLIDEANS' ALGORITHM

As discussed earlier, Euclidean's' Algorithm is one of the many concept and algorithms of *Number Theory*. Number theory is the concept behind secure online communication and transactions. The concept is applied to achieve the following goals:

Date integrity: Information should not be altered without detection. For example, it is important to prevent the modification of purchase orders or other contractually binding documents transmitted electronically.

Authentication: Individuals and organizations that are accessing or communicating sensitive information must be correctly identified, that is, authenticated. For example, corporations offering telecomm uniting arrangements to their employees should set up an authentication procedure for accessing corporate database through the Internet.

Authorization. Agents that are performing computations involving sensitive information must be authorized to perform those computations.

Non repudiation: In transactions that imply a contract, the parties that have agreed to that contract must not have the ability of backing out of the their obligations without being detected.

Confidentiality: Sensitive information should be kept secret from individuals who are not authorized to see that information. That is, we must ensure that data is viewed by the sender and by the receiver, but not by unauthorized parties who can eavesdrop on the communication. For example, many email messages are meant to be confidential.

VII. REFERENCES

[1] Arjen K Lestra and Citibank N A (2002): *"Computational Methods in Public Key Cryptology"*, 1 North Gate Road, Mendham, NJ 07945-3104, U.S.A. 973 543 5091, 973 543 5094 (fax). Arjen.lenstra@citigroup.com. pp16.

[2] Christopher M. Bourke, Berthe Y(2006). Choueiry: *"Number Theory. Application"*, Computer Science & Engineering 235 Introduction to Discrete Mathematics Sections 2.4-2.6 of Rosen. Cse235@cse.unl.edu.