Comparing different variants of gcd algorithms of large integers Summary

Sunday 12th January, 2020 - 19:41

Licina Esada
University of Luxembourg
Email: esada.licina.001@student.uni.lu

Volker Müller
University of Luxembourg
Email: volker.müller@uni.lu

***Abstract*—This summary of the final report made by the student Esada Licina and by the support of hers tutor Volker Müller.Here will be explain the scientific an technical part of the project.For more information about the project,read the final report once.**

1. Introduction

Maths is a difficult case for most people.Humans finds out some methods to solve them, one of them are this algorithms.How the title says,the project is about to compare different variants of gcd algorithms.Before beginning to talk more about the project,the terms 'gcd' and 'algorithms' will be explain.

The term 'gcd' is an abbreviation of the words 'Greatest Common Divisor'.By the term 'algorithm' is the speech of a process that solve problems

To compare the algorithms,the project will calculate the time of them to look which has the best performance.There are 5 different variants of gcd algorithms the Euclidean, the extended Euclidean,the Binary, the extended Binary and the Java BigInteger algorithm.

The project will have also a practical part where the algorithm will be program in Java with help of NetBeans.It will also have a Excl part where the results of NetBeans will be saved,to make a diagram of the duration.

2. Project description

Here will be described the three domains.

2.1. Scientific

James Gosling was the man who developed the programming language Java and it was published 1995 by the company Sun Microsoft.NetBeans facilitates the creation of Java swing application and that's why NetBeans and Java works together.Excel is a spreadsheet program that was developed by Microsoft and it has about 471 function.

2.2. Technical

Java has 3 classes.In the first class are the codes of the gcd algorithm with BigInteger, in the second are algorithms with integers and in the third is the data installation for the Excel sheet.In NetBeans will be create a virtual window to see the results and the function to save the data in Excel.After saving it on a Excel sheet, it will be made a diagram to see the performance of the algorithms.

3. Prerequisites

3.1. Scientific

Before beginning the project it is important to know how the algorithms works and to go behind their logic of maths. Then it is needed to understand the system of NetBeans,Java and Excel.Also knowing to transform the algorithm in Java language and how to create diagrams in Excel.

3.2. Technical

Here it is important to have experience with programming in Java and NetBeans.The most significant part in NetBeans is to know how to work with MainFrame because there is the virtual window.Then it is useful to have a little bit experience in Excel because it is not so difficult to learn it.

4. Scientific presentation to comparison of performance of several gcd algorithms

4.1. Requirements

Functional requirements:

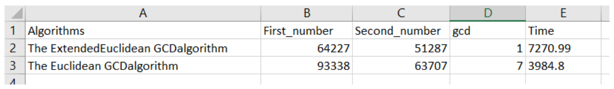
1) Calculation of the time.
2) Show the duration of the algorithms.
3) Add the time of the algorithms on Excel.

Non-Functional requirements:

1) User interface of NetBeans window.
2) User interface of the Excel sheet.

4.2. Design & Production

Here will be showed the Excel design.



	A	B	C	D	E
1	Algorithms	First_number	Second_number	gcd	Time
2	The ExtendedEuclidean GCDalgorithm	64227	51287	1	7270.99
3	The Euclidean GCDalgorithm	93338	63707	7	3984.8

Figure 1. Excel window of the results

It look like this when the data gets saved there.After that the user can make colors and create a diagram .He needs also to do one column with the length of the numbers, he can use this formula: **=SUMMME(LÄNGE(place of the number))**.

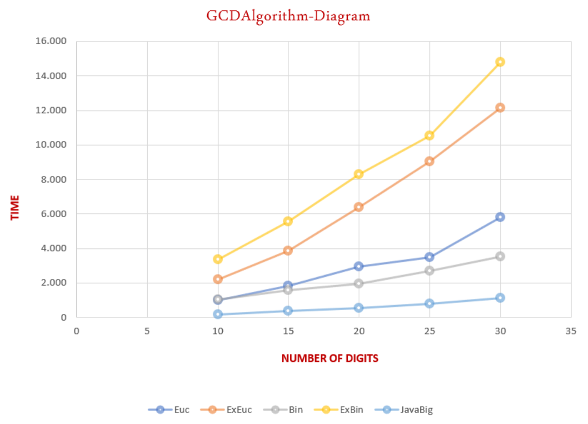


Figure 2. Excel diagram of the gcd algorithms

As seen the **extended Binary** algorithm is the slowest because his code is complex and he works with for variables.If he calculates with smaller numbers then he is more faster.Then the fastest is the **Java BigInetger**, but there is no code because it is a algorithm made by Java.So there is no explanation.The second fastest is the **Binary** gcd algorithm because he works with the **RightShiftOperation**, that means that he divides the numbers by 2.The **Euclidean** is slower because he use the **modulo operation**.

4.3. Assessment

The requirements were successfully met.As can be seen on the design and production part, the final images shows that it works.

5. A NetBeans/Java program to calculate the gcd of big numbers

5.1. Requirements

Functional requirements:

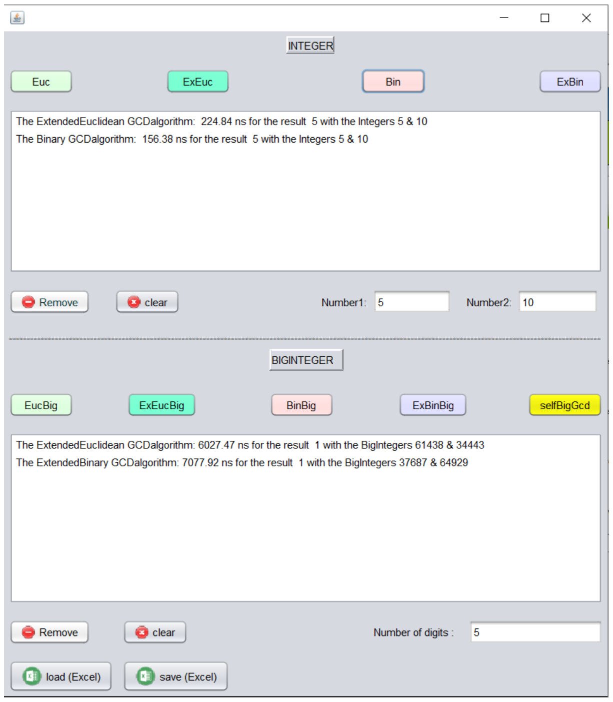
1) Calculation of the random numbers.
2) Calculation of the greatest common divisor.

Non-Functional requirements:

1) The TextField function.
2) The availability of function in NetBeans.

5.2. Design

The NetBeans window design will be shown here.



```
The ExtendedEuclidean GCDalgorithm:  224.84 ns for the result  5 with the Integers 5 & 10
The Binary GCDalgorithm:  156.38 ns for the result  5 with the Integers 5 & 10
```

Number1: 5 Number2: 10

```
The ExtendedEuclidean GCDalgorithm: 6027.47 ns for the result  1 with the BigIntegers 61438 & 34443
The ExtendedBinary GCDalgorithm: 7077.92 ns for the result  1 with the BigIntegers 37687 & 64929
```

Number of digits : 5

Figure 3. NetBeans window design

As can be seen there are two parts one with INTEGER and one with BIGINTEGER.The first part is just for fun and to check if the algorithms gives the right gcd back.The user can input his own number (under 32 bites) that he wants to calculate.By the second part he can just input the length of the random number.BigInteger can calculate enormous numbers but integer can't.

5.3. Production

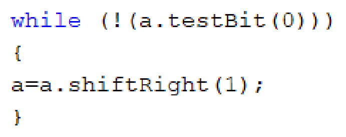
The code below is from the second fastest gcd algorithm.

```
int k=0;
while((!(a.testBit(0))) && (!(b.testBit(0))))
{
    a=a.shiftRight(1);
    b=b.shiftRight(1);

    k++;
}
```

Figure 4. Binary code part1

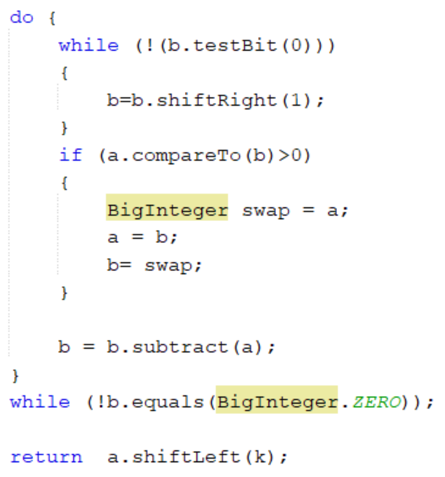
1) The code checks if the numbers **a** and **b** are not equal to 0 (That's not shown on the picture). If one of the numbers equals 0 , then the gcd is **a** or **b**.
2) If not he checks with the **while loop** if the numbers are even with this method **!(number.testBit(0))**.It checks if the first digit of the binary number is not 1.That's the binary way to look if a number is even or not.All binary numbers that are even have the number 0 on the first digit.
3) If they are even he divides the numbers by 2 with the **RightShiftOperation** until they get odd numbers.The **RightShiftOperation** moves the binary number 1 digit on the right side.
4) Every time when the code divide the numbers by 2, he will add 1 by the variable **k**.



```
while (!(a.testBit(0)))
{
a=a.shiftRight(1);
}
```

Figure 5. Binary code part2

5) If the **while loop** stops because one of them is not even,he looks if the number **a** is even.If yes he divides **a** by 2 until **a** gets an odd number.



```
do {
    while (!(b.testBit(0)))
    {
        b=b.shiftRight(1);
    }
    if (a.compareTo(b)>0)
    {
        BigInteger swap = a;
        a = b;
        b= swap;
    }

    b = b.subtract(a);
}
while (!b.equals(BigInteger.ZERO));

return  a.shiftLeft(k);
```

Figure 6. Binary code part3

6) Here is a **do-while loop** and he checks the condition in the end.So the code can run one time without being check if it is true or false.
7) The **while loop** looks if **b** is even,if not he checks if **a** is bigger then **b**.
8) If yes the numbers will be swapped, because after that **b** gets a new value by subtract **b** from **a**.
9) Then the **do-while loop** checks if **b** is not 0 ,if no he repeats the code until **b** is 0 and if **b** is 0 then the codes return the gcd with multiplying **a** by **k** with the **LeftShiftOperation**.

5.4. Assessment

Also here were the requirements successfully met.There are pictures of the NetBeans window and of the algorithms codes on the design and production part.

6. Acknowledge and conclusion

Finally the goal is reached, the project shows that the Java BigInteger algorithm is the fastest and the second fastest is the Binary algorithm.But the slowest one was the extended Binary gcd algorithm.

I,Esada Licina would to thanks a lot my project tutor Volker Müller,the BICS team, my family and my friend for their support during this semester.