Descripción, Análisis y Comparación de Diversas Formulaciones del Algoritmo Simple y Extendido de Euclides

Eileen Karin Apaza Coaquira , Renzo Leonardo Gallegos Vilca, Xiomara Leonor Puma Torres, Santiago Alonso San Roman Olazo

Programa Profesional de Ciencia de la Computación,
Universidad Católica de San Pablo
Arequipa, Perú

eileen.apaza@ucsp.edu.pe

renzo.gallegos.edu.pe

xiomara.puma@hotmail.com

santiago.sanroman@hotmail.com

Aportes de cada integrante:

Eileen Karin Apaza Coaquira:

-Algoritmo de Euclides con menor resto

-Complementó los algoritmos extendidos de Euclides(iterativo, recursivo).

-Parte del Analisis de Algoritmos

Renzo Leonardo Gallegos Vilca:

-Algoritmo binario del mcd

-Parte del Análisis de Algoritmos.

Xiomara Leonor Puma Torres:

Algoritmo de Euclides Clásico

Santiago Alonso San Román Olazo

-Códigos de los algoritmos extendidos de euclides(iterativo,recursivo)

***Abstract*—En el presente documento se describe y analiza diversas formulaciones del algoritmo simple y extendido de Euclides, con el fin de comparar el tiempo de ejecución usando enteros con diferente número de bits. Como se puede observar, cada algoritmo tiene un rendimiento diferente que varía según el tamaño del entero, la mayoría de las variaciones del algoritmo de Euclides tiene un buen desempeño, en cambio el algoritmo extendido de Euclides recursivo muestra un mayor tiempo de ejecución debido a la mayor cantidad de operaciones.**

Keywords—Algoritmo extendido de Euclides, algoritmo binario, máximo común divisor (mcd).

I. Introducción

El algoritmo euclidiano es un algoritmo antiguo, conocido y eficiente para encontrar el máximo común divisor de dos enteros. Si a, b son enteros no nulos y $r = a \bmod b$, sabiendo que si $mcd\,(a, b) = 1$ se dice que a y b son relativamente primos [2], entonces

$$mcd\,(a, b) = mcd\,(b, r). \tag{1}$$

Por el teorema del cociente-residuo, existen q y r que satisfacen

$$a = bq + r \qquad 0 \leq r < b.$$

Se demuestra que el conjunto de divisores comunes de a y b es igual al conjunto de divisores comunes de b y r, lo que prueba el teorema. Ahora bien, sea c un divisor común de a y b. Por el Teorema (1), $c|bq$. Como $c|a$ y $c|bq$, $c|a - bq$ ($= r$). Entonces c es un divisor común de b y r. Inversamente, si c es un divisor común de b y r, entonces, $c|bq$ y $c|bq + r$ ($= a$) y c es un divisor común de a y b. Así, el conjunto de divisores comunes de a y b es igual al conjunto de divisores comunes de b y r. Por lo tanto, $mcd\,(a, b) = mcd\,(b, r)$ [1]. Denotamos con D_a al conjunto de divisores de a y con D_b el conjunto de divisores de b. Estos conjuntos no son vacíos pues al menos $1 \in D_a$ y $1 \in D_b$. El máximo común divisor común de a y b es el más grande entero positivo del conjunto $D_a \cap D_b$ [2].

En este documento, se analizamos las variaciones más conocidas del algoritmo de Euclides comparando el tiempo de ejecución según el número de bits, además describimos el enfoque matemático que tiene como base al teorema de la división.

II. Contenido Teorico

A. Algoritmo de Euclides clásico

1) Definición

El algoritmo de Euclides se basa en la aplicación sucesiva del teorema de la división [2].

2) Enfoque matemático

Sean a y b números enteros, tal que $b \neq 0$. Aplicando el teorema de la división se obtiene una sucesión finita de la forma a, $r_0 = b, r_1, r_2, \ldots, r_n$, 0 definida por

$$\begin{aligned} a &= r_0 q_1 + r_1, && 0 \leq r_1 < r_0 \\ r_0 &= r_1 q_2 + r_2, && 0 \leq r_2 < r_1 \\ r_1 &= r_2 q_3 + r_3, && 0 \leq r_3 < r_2 \\ &\vdots \end{aligned}$$

$$r_{n-2} = r_{n-1}\, q_n + r_n, \qquad 0 \le r_n < r_{n-1}$$
$$r_{n-1} = r_n\, q_{n+1} + 0, \qquad r_{n+1} = 0$$

El último término es $r_n = mcd\,(a, b)$ [2].

3) Pseudocódigo

Algoritmo 2.2: Máximo común divisor

```
Datos: a, b ∈ Z, b ≠ 0
Salida: mcd(a,b)
1 c = |a|, d = |b|;
2 while d ≠ 0 do
3 |   r = mod(c,d);
4 |   c = d;
5 |   d = r;
6 return mcd(a,b) = |c|;
```

4) Seguimiento númerico:

i	1	2	3	4	5	6	7
a	412	260	152	108	44	20	4
b	260	152	108	44	20	4	0

Ejemplo:

mcd(412, 260) =4

$$412 = 1 \cdot 260 + 152$$
$$260 = 1 \cdot 152 + 108$$
$$152 = 1 \cdot 108 + 44$$
$$108 = 2 \cdot 44 + 20$$
$$44 = 2 \cdot 20 + 4$$
$$20 = 5 \cdot 4 + 0.$$

5) Implementación:

```
#include <iostream>
#include <NTL/ZZ.h>

NTL::ZZ mod(NTL::ZZ a, NTL::ZZ n) {
   NTL::ZZ r = a - (a / n) * n;
   if (r < 0) {
      r = r + n;
   }
   return r;
}

NTL::ZZ mcd(NTL::ZZ a, NTL::ZZ b) {
   NTL::ZZ c = abs(a);
   NTL::ZZ d = abs(b);

   while (d != 0) {
      NTL::ZZ r = mod(c, d);
      c = d;
      d = r;
   }
   return abs(c);
}
```

B. Algoritmo de Euclides con menor resto

1) Definición

Es una variación del algoritmo clásico de Euclides que busca reducir el número de divisiones. Kronecker estableció en 1901 que el número de divisiones en el algoritmo “con menor resto” es menor o igual que el número de divisiones en el algoritmo clásico de Euclides [2].

2) Enfoque matemático

Ceil(x): Devuelve el menor entero mayor o igual a x.

x=Min {n ∈ Z | n >= x}

Ejemplo:

ceil(2.25) = 3 , ceil(2)=2 ,Ceil (-2.25)=-2

Floor(x): Devuelve el más grande entero menor o igual a x.

x=Max {n ∈ Z | n <= x}

Ejemplo:

Floor(2.8) =2 , Floor(-2)=-2 , Floor(-2.3)=-3

Notemos que x = **x** si y sólo si x es entero, en otro caso x= **x** + 1.

Usamos Ceil(x) y Floor(x), para poder encontrar el menor resto .

3) Pseudocódigo

```
function mcdMenorResto(a,b)
    long c, d,r
    Si a=0 then
        c=b
    Sino
        c=a
        d=b
    Mientras d<>0
        r=c-d*(c/d+1/2)
        c=d
        d=r
    endMientras

    mortrar abs(c)
endFunction
```

4) Seguimiento númerico:

Algoritmo de Euclides con menor resto:

• Seguimiento Numérico:

mcd(a,b) = mcd(|a|,|b|): se usara divisor y dividendo positivo.

$144 = 89 * 2 - 34$ $\Rightarrow$ mcd(89,144) = mcd(89,144)

$89 = 34 * 3 - 13$ = mcd(34,13)

$34 = 13 * 3 - 5$ = mcd(13,5)

$13 = 5 * 3 + 2$ = mcd(5,2)

$5 = 2 * 2 + 1$ = mcd(2,1)

$2 = 1 * 2 + 0$ = mcd(1,0) = 1

$\Rightarrow$ mcd(89,144) = 1

5) Implementación:

```
#include <iostream>
#include <NTL/ZZ.h>

NTL::ZZ mod(NTL::ZZ a, NTL::ZZ n) {
   NTL::ZZ r = a - (a / n) * n;
   if (r < 0) {
      r = r + n;
   }
   return r;
}

NTL::ZZ mcdMenorResto(NTL::ZZ a, NTL::ZZ b) {
   NTL::ZZ c;
   NTL::ZZ d;
   NTL::ZZ r;

   if (a==0) {
      c = b;
   }
   else {
      c = a;
      d = b;
      while (d != 0) {
         r = c - d * (c/d + NTL::ZZ(1/2));
         c = d;
         d = r;
      }
   }
   return abs(c);
}
```

C. Algoritmo binario del MCD

1) Definición:

Mediante el algoritmo binario se busca modificar y reemplazar los valores de entrada del máximo común divisor por el *mcd* de valores reducidos por medio de la resta y división por 2, según sea el caso.

2) Enfoque matemático:

Mora, 2010 [2] define los teoremas con los que opera el algoritmo:

Regla 1. Si a, b son pares, $\operatorname{mcd}(a,b) = 2\operatorname{mcd}\left(\frac{a}{2}, \frac{b}{2}\right)$

Regla 2. Si a es par y b impar, $\operatorname{mcd}(a,b) = \operatorname{mcd}\left(\frac{a}{2}, b\right)$

Regla 3. Si a, b son impares, $\operatorname{mcd}(a,b) = \operatorname{mcd}\left(\frac{|a-b|}{2}, b\right) = \operatorname{mcd}\left(\frac{|a-b|}{2}, a\right)$

3) *Pseudocódigo*

Algoritmo 8.2: Algoritmo binario para el mcd

```
Datos: a, b ∈ Z, a ≥ 0, b > 0
Salida: mcd(a,b)
g = 1;
while a mod 2 = 0 And b mod 2 = 0 do
    a = quo(a,2), b = quo(b,2);
    g = 2g //removiendo potencias de 2
while a ≠ 0 do // Ahora, a o b es impar

    if a mod 2 = 0 then
        a = quo(a,2)
    else if b mod 2 = 0 then
        b = quo(b,2)
    else ; // ambos impares

        t = quo(|a − b|,2);
        if a ≥ b then ; // reemplazamos máx{a,b} con quo(|a − b|,2)

            a = t
        else
            b = t

return g · b;
```

4) *Seguimiento númerico:*

$$
\begin{aligned}
\operatorname{mcd}(89,44) &= \operatorname{mcd}(22,89), && \text{por Regla 2}\\
&= \operatorname{mcd}(11,89), && \text{por Regla 2}\\
&= \operatorname{mcd}(39,11), && \text{por Regla 3}\\
&= \operatorname{mcd}(14,11), && \text{por Regla 3}\\
&= \operatorname{mcd}(7,11), && \text{por Regla 2}\\
&= \operatorname{mcd}(2,7), && \text{por Regla 3}\\
&= \operatorname{mcd}(1,7), && \text{por Regla 2}\\
&= \operatorname{mcd}(3,1), && \text{por Regla 2}\\
&= \operatorname{mcd}(1,1), && \text{por Regla 3}\\
&= \operatorname{mcd}(0,1), && \text{por Regla 3}\\
&= 1
\end{aligned}
$$

$$
\begin{aligned}
\operatorname{mcd}(8,48) &= 2\cdot\operatorname{mcd}(4,24), && \text{por Regla 1}\\
&= 4\cdot\operatorname{mcd}(2,12), && \text{por Regla 1}\\
&= 8\cdot\operatorname{mcd}(1,6) && \text{por Regla 1}\\
&= 8\cdot\operatorname{mcd}(1,3), && \text{por Regla 2}\\
&= 8\cdot\operatorname{mcd}(1,1), && \text{por Regla 3}\\
&= 8\cdot\operatorname{mcd}(0,1), && \text{por Regla 3}\\
&= 8
\end{aligned}
$$

5) *Implementación:*

```cpp
#include <iostream>
#include <NTL/ZZ.h>

NTL::ZZ mod(NTL::ZZ a, NTL::ZZ n) {
   NTL::ZZ r = a - (a / n) * n;
   if (r < 0) {
      r = r + n;
   }
   return r;
}

NTL::ZZ mcdBinario(NTL::ZZ a, NTL::ZZ b) {
   NTL::ZZ g = NTL::ZZ(1);
   while ((mod(a, NTL::ZZ(2)) == 0) &&
                    (mod(b, NTL::ZZ(2)) == 0)) {
      a = a / 2;
      b = b / 2;
      g = 2 * g;
   }
   while (a != 0) {
      if (mod(a, NTL::ZZ(2)) == 0) {
         a = a / 2;
      }
      else if (mod(b, NTL::ZZ(2)) == 0) {
         b = b / 2;
      }
      else {
         NTL::ZZ t = abs(a - b) / 2;
         if (a >= b) {
            a = t;
         }
         else {
            b = t;
         }
      }
   }
   return g * b;
}
```

ALGORITMO EXTENDIDO DE EUCLIDES:

Definición:

El algoritmo euclidiano extendido actualiza los resultados de gcd (a, b) utilizando los resultados calculados por la llamada recursiva gcd (b% a, a). Deje que los valores de xey calculados por la llamada recursiva sean x 1 e y 1 . x e y se actualizan utilizando las siguientes expresiones.

A. Algoritmo extendido de euclides (ITERATIVO)

1) *Enfoque matemático:*

Encuentra los coeficientes por los que el MCD de dos números se expresa en términos de los números mismos.

x	y	$(\operatorname{rem}(x,y))$	$=$	$x - q\cdot y$
259	70	49	$=$	$259 - 3\cdot 70$
70	49	21	$=$	$70 - 1\cdot 49$
			$=$	$70 - 1\cdot(259 - 3\cdot 70)$
			$=$	$-1\cdot 259 + 4\cdot 70$
49	21	7	$=$	$49 - 2\cdot 21$
			$=$	$(259 - 3\cdot 70) - 2\cdot(-1\cdot 259 + 4\cdot 70)$
			$=$	$\boxed{3\cdot 259 - 11\cdot 70}$
21	7	0		

2)*Pseudocódigo:*

```
function eucExt(a,b,x,y)
   x=1 y=0
   long x1=0 y1=1 a1= a b1=b
   Mientras(b1) entonces
      long q = a1/b1
      tie (x,x1) = (x1, x - q *x1)
      tie (y,y1) = (y1, x - q *y1)
      tie (a1,b1) = (b1, a1 - q *b1)
   endMientras

   muestra a1
endFunction
```

3)*Seguimiento Numérico:*

Seguimiento Numérico:

(141, 96, x, y)

• Aplicamos el Algoritmo de Euclides

141 = 96 * 1 + 45
96 = 45 * 2 + 6
45 = 6 * 7 + 3
6 = 3 * 2 + 0

=> d = 3

• Calculamos x e y

141 = 141 * 1 + 96 * 0
96 = 141 * 0 + 96 * 1

45 = 141 - 96 = (141 * 1 + 96 * 0) - (141 * 0 + 96 * 1) = 141 * 1 + 96 (-1)
6 = 96 - 45 * 2 = (141 * 0 + 96 * 1) - (141 * 1 + 96 (-1)) * 2 = 141 * 2 + 96 * 3
3 = 45 - 6 * 7 = (141 * 1 + 96 (-1)) - (141 (-2) + 96 * 3) * 7 = 141 * 15 + 96 (-22)

=> x = 15 ∧ y = -22

141 * 15 + 96 (-22) = 2115 - 2112 = 3 ✓

4)*Implemetacion en c++:*

```cpp
#include <iostream>
#include <NTL/ZZ.h>
using namespace std;
using namespace NTL;
void eucExt(ZZ a, ZZ b, ZZ& x, ZZ& y)
{
   x = ZZ(1), y = ZZ(0);
```

```
    ZZ x1(0), y1(1), a1(a), b1(b);
    while (b1 != 0)
    {
        ZZ q = a1 / b1;
        tie(x, x1) = make_tuple(x1, x - q * x1);
        tie(y, y1) = make_tuple(y1, y - q * y1);
        tie(a1, b1) = make_tuple(b1, a1 - q * b1);
    }
}
int main()
{
    ZZ x, y, a, b;
    cin>>a;
    cin>>b;
    eucExt(a, b, x, y);
    cout << "MCD("<< a << ", " << b<< " ) = " << eucExt(a, b, x, y)<< endl;
    return 0;
}
```

B. Algoritmo extendido de Euclides(RECURSIVO)

1)Definición:

Devuelve un valor GCD de los números a y b , pero aparte de eso, como coeficientes requeridos x e y en función de los parámetros pasados por referencia.

2)Enfoque matemático:

La base de recursividad es el caso a = 0. Entonces MCD es igual b y, obviamente, los coeficientes requeridos x e y son iguales 0 y, 1 respectivamente.

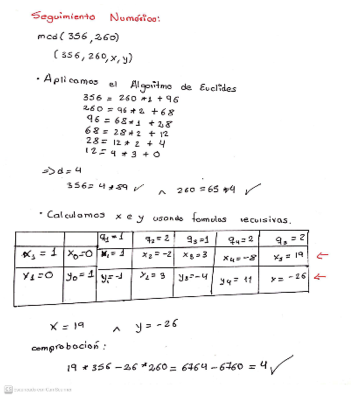
3)Pseudocódigo:

```
Function gcdExtended(a,b,x,y)
    Si (a=0) entonces
        x=0
        y=1
        muestra b
    endSi

    long x1 , y1
    long gcd=gcdExtended(b % a, a, x1, y1)
    x = y1 - (b / a) * x1
    y = x1

    muestra gcd
endFunction
```

4)Seguimiento Numérico:



5)Implementación en c++:

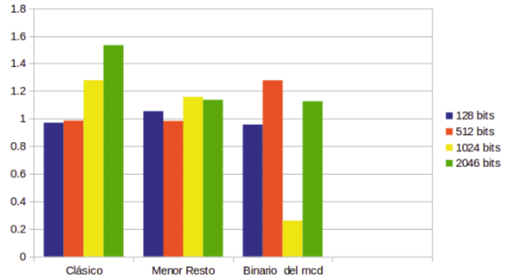
```
#include <iostream>
#include <NTL/ZZ.h>
using namespace std;
using namespace NTL;
ZZ gcdExtended(ZZ a, ZZ b, ZZ* x, ZZ* y)
{
    if (a == 0)
    {
        *x = 0;
        *y = 1;
        return b;
    }
    ZZ x1, y1;
    gcdExtended(b % a, a, &x1, &y1);
    *x = y1 - (b / a) * x1;
    *y = x1;
    return gcdExtended(b % a, a, &x1, &y1);
}
int main()
{
    ZZ x, y, a , b ;
    cin>>a;
    cin>>b;
    gcdExtended(a, b, &x, &y);
    cout << "MCD(" << a << ", " << b << " ) = " << gcdExtended(a, b, &x, &y) <<
endl;
    return 0;
}
```

III. Análisis de algorimos

Comparación del tiempo de ejecución según el N° de bits (128 – 512 – 1024 – 2046):

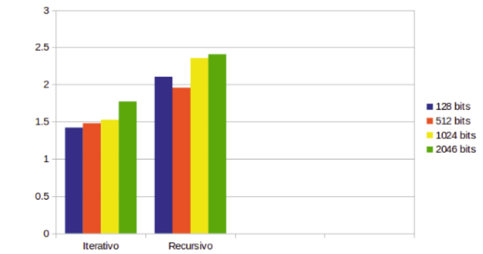
A) Algoritmo de Euclides:

	128 bits	512 bits	1024 bits	2046 bits
Clásico	0.970	0.986	1.279	1.533
Menor Resto	1.054	0.983	1.158	1.137
Binario	0.957	1.278	1.260	1.126



B) Algoritmo Extendido de Euclides:

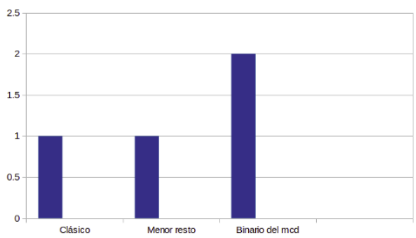
	128 bits	512 bits	1024 bits	2046 bits
Iterativo	1.420	1.479	1.526	1.771
Recursivo	2.105	1.957	2.355	2.407



Comparación del N° de loops:

A)Algoritmo de Euclides:

	Clásico	Menor Resto	Binario
N° de loops	1	1	2



B) Algoritmo Extendido de Euclides:

	Iterativo	Recursivo
N° de loops	1	0

IV. CONCLUSIÓN

Evaluando todos los algoritmos llegamos a la conclusión que el mejor algoritmo de euclides es el Algoritmo de euclides con menor resto ya que es el mas eficaz para el N° de bits.

En el algoritmo extendido de euclides se llegó a al conclusion que el mejor algoritmo es el iterativo ya que no presenta demora al ejecutar el programa y es eficaz en N° de bits

REFERENCES

[1] R. Johnsonbaugh, Matemáticas Discretas, Chicago: Pearson Prentice Hall, 2013.

[2] W. Mora, Introducción a la Teoría de Números: Ejemplos y Algoritmos, Cartago, Costa Rica: Revista digital Matemática Educación e Internet, 2010.