Comparación de algoritmos para determinar el Máximo Común Divisor (MCD)

Valeria Calle Rodriguez
Brigham Caceres Gutierrez
Galo de Paula Jimenez Araujo
Fernando Jair Peralta Bustamante

Junio 2021

1 Aporte

Valeria Calle Rodriguez: Diapositivas y algoritmo binario.
Brigham Caceres Gutierrez: Comparacion y algoritmo de lehmer.
Galo de Paula Jimenez Araujo: Testing y algoritmo de Euclides con menor resto.
Fernando Jair Peralta Bustamante: Paper y algoritmo de Euclides clásico.

2 Resumen

En el paper hemos analizado 4 diferentes algoritmos, el algoritmo de Euclides clásico base de todos los algoritmos para hallar el mcd, el algoritmo de Euclides con menor resto, el cual es similar al clásico pero como indica su nombre, busca obtener un resto menor para reducir la cantidad de pasos, el algoritmo binario del MCD donde ya se modifica bastante lo planteado por el algoritmo de Euclides y se da paso a trabajar cada numero de manera independiente teniendo como base la división de 2 en caso el numero sea par y por ultimo el algoritmo de Lehmer en el que trabajamos ya con ordenamiento de los numeros para de esta manera poder obtener el mcd mas eficientemente.

Los algoritmo que mostraron mejor desempeño en las pruebas fueron el algoritmo de menor resto y Euclides clásico pero con la sorpresa dada al momento de evaluar su coste en loops donde el algoritmo de menor resto explica el porque de su buen desempeño en el tiempo y cantidad de bits siendo el completamente descartado el algoritmo binario.

3 Introducción

En este ensayo los investigadores vamos a presentar un análisis de los siguientes algoritmos para determinar el Máximo Común Divisor (MCD):

1. Algoritmo de Euclides clásico.
2. Algoritmo de Euclides con menor resto
3. Algoritmo binario del MCD
4. Algoritmo de Lehmer del MCD

La investigación consistirá de una explicación teórica de cada algoritmo junto con su implementación detallando el proceso de cambio en el contenido de las variables realizando un seguimiento de código con datos de entrada reales (números enteros). Seguido de esto realizaremos una Comparación de los algoritmos de acuerdo a la convergencia (quien tiene el menor tiempo para llegar a los resultados) y eficiencia respaldando esta información con el fundamento matemático que lo comprueba.

4 Discusión de algoritmos seleccionados para MCD

4.1 Algoritmo de Euclides clásico

4.1.1 Definición:

La idea de este algoritmo se basa en que el máximo común divisor de dos números se puede encontrar dividiendo el número mayor por el número menor. Si la división es exacta, el MCD es el número menor. Si la división no es exacta, entonces se toma el residuo, y se divide tantas veces como haga falta para llegar a una división sin residuo ($r = 0$). El MCD es el último número por cuál se puede dividir.

4.1.2 Sustento matemático:

Esta basado principalmente en las identidades:

$$mcd(a, b) = mcd(b, a - bq)$$

$$mcd(r, 0) = r$$

de tal manera que si $q = bq + r_1$ y $b = r_1q_1 + r_2$ con $0 \leq r_2 < r_1 < b$,

$$mcd(a, b) = mcd(b, r_1) = mcd(r_1, r_2)$$

es decir, conforme aplicamos esta relación, cambiamos el cálculo del mcd de dos números a y b por el mcd de dos números más pequeños. El proceso es finito y se detiene cuando encontramos un resto nulo.

Formalmente: Sean a y b números naturales, $b \neq 0$. Aplicando el algoritmo de la división se obtiene una sucesión finita $r_1, r_2, ..., r_n$ definida por:

$$\begin{aligned} a &= bq_1 + r_1, 0 \leq r_1 < r_0 \\ b &= r_1 q_2 + r_2, 0 \leq r_2 < r_1 \\ r_1 &= r_2 q_3 + r_3, 0 \leq r_3 < r_2 \\ &\vdots \\ r_{n-2} &= r_{n-1} q_{n+r_n}, 0 \leq r_n < r_{n-1} \\ r_{n-1} &= r_n q_{n+1} + 0 \end{aligned} \tag{1}$$

$r_n = mcd(a, b)$ pues $mcd(a, b) = mcd(r_1, r_2) = ... = mcd(r_n, 0) = r_n$

4.1.3 Pseudo-algoritmo:

- Ingresa dos enteros positivos m,n donde $m \geq n$
- Inicializa $r = m$ mod n
- Obtendrás el máximo común divisor de m y n

1. mientras $n \neq 0$ has lo siguiente:
   (a) Coloca $r \longleftarrow m$ mod n, $m \longleftarrow n$, $n \longleftarrow r$
2. retorna (m)

4.1.4 Seguimiento numérico:

1. $m = 15$, $n = 8$
2. $r = 15$ mod 8 (7)
3. como $n \neq 0$, entonces
   - $m = 8$
   - $n = 7$
   - $r = 8$ mod 7 (1)
4. Comparáramos de nuevo $n \neq 0$, entonces
   - $m = 7$
   - $n = 1$
   - $r = 7$ mod 1 (0)
5. Una vez mas $n \neq 0$, entonces

- $m = 1$
- $n = 0$
- $r = 1 \bmod 0$ (0)

6. Tenemos que $n = 0$, por lo tanto retornamos el valor de m
7. **MCD por Euclides clásico es: 1**

4.1.5 Rastreo:

Steps	m	n	r
1	15	8	7
2	8	7	1
3	7	1	0

Table 1: Valores de las variables en el programa

4.1.6 Implementación en c++:

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

using namespace std;
using namespace NTL;

ZZ euclides_clasico(ZZ a, ZZ b)
{
    // r -> rest
    while(b != 0){
        ZZ r = modulo(a, b);
        a = b;
        b = r;
    }
    return a;
}
```

4.2 Algoritmo de Euclides con menor resto

4.2.1 Definición

Este algoritmo tiene menos pasos que el algoritmo de Euclides común dado que opta por usar números mas pequeños. Esto tiene dos ventajas, por la parte computacional tratando con números tan granes siempre es bienvenido usar números mas pequeños para reducir el uso de memoria y de poder de

procesamiento. Por otro lado, como se busca usar números mas pequeños se llega a la respuesta más rápido y se tiene que hacer menos iteraciones de el algoritmo para llegar al mcd.

4.2.2 Sustento matemático

En la versión clásica del algoritmo de Euclides, el resto r_i está entre 0 y r_{i-1}. Podemos hacer una pequeña variación para que cada nuevos resto r_i esté entre 0 y $r_{i1}/2$ con lo que, en general, podría haber una reducción en el número de divisiones

Como $mcd(a,b) = mcd(\mid a \mid, \mid b \mid)$ vamos a suponer que $a \geq 0$ y $b > 0$. Recordemos que :

$$a = b.[[a/b]] + r_2.0 \leq r_2 < b$$

$$a = b.([[a/b]] + 1) - r_1.0 \leq r_1 < b$$

Para mejorar un poco el desempeño del algoritmo de Euclides, escogemos en cada paso el menor resto, es decir, $r = Min\{r1, r2\} = Min\{\mid a - b * [[a/b]] \mid, \mid a - b * ([[a/b]] + 1) \mid\}$. De esta manera r$\leq b/2$.

El algoritmo de Euclides sigue siendo válido pues si tomamos el menor resto r en cada paso, $mcd(a,b) = mcd(b,r)$ y de nuevo obtenemos una sucesión decreciente de restos, el último resto no nulo $\mid r_n \mid$ $eselmcddeayb.Porsupuesto$,

$$r = Min\{r_1, r_2\} = a - b * [[a/b + 1/2]]$$

4.2.3 Pseudo-algoritmo

- Ingresa dos enteros positivos a,b donde
- Inicializa c, d y r
- Obtendrás el máximo común divisor de m y n

1. Si a=0 entonces c=b
2. Si no se cumple c=a y d=b
3. Mientras $d \neq 0$
   - $r = c - d * int(c/d + 1/2)$
   - c=d
   - d=r
4. mcdMenorResto = Abs(c)

4.2.4 Seguimiento numérico

$$
\begin{aligned}
& mcd(89, 144) \\
144 = 89 * 2 - 34 \Rightarrow & mcd(89, 144) = mcd(89, 34) \\
89 = 34 * 3 - 13 = & mcd(34, 13) \\
34 = 13 * 3 - 5 = & mcd(13, 5) \\
13 = 5 * 3 + 2 = & mcd(5, 2) \\
5 = 2 * 2 + 1 = & mcd(2, 1) \\
2 = 1 * 2 + 0 = & mcd(1, 0) = 1
\end{aligned} \tag{2}
$$

mcd (89,144) = 1

4.2.5 Rastreo

Steps	a	b	r
1	144	89	-34
2	89	34	-13
3	34	13	-5
4	13	5	2
5	5	2	1
6	2	1	0

Table 2: Valores de las variables en el programa

4.2.6 Implementación en c++

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

using namespace std;
using namespace NTL;

ZZ euclides_menor_resto(ZZ a, ZZ b){

    ZZ c, d, r;
    if (a == 0)
        c = b;
    else
    {
        c = a;
        d = b;
        while(d != 0)
```

```
        {
            r = c - d * (c/d + 1/2);
            c = d;
            d = r;
        }
    }
    return abs(c);
}
```

4.3 Algoritmo binario del MCD

4.3.1 Definición:

Es una sucesión de pasos para encontrar el mcd de dos números enteros positivos a y b donde a es mayor o igual a b. La solución que plantea este algoritmo es inicializar una variable g en 1. Luego, en caso los dos números fuesen pares, se dividen en simultáneo entre 2 y a g se le multiplica por 2; si no es el caso, se divide a entre 2 hasta que a sea impar y seguido se realiza en b la misma operación. Una vez a y b sean impares se realiza el valor absoluto de la resta de ambos números y si el resultado fuese par se divide entre dos, este resultado se le asigna al número mayor, y así hasta que a sea 0. Finalizando con el algoritmo, se multiplica g y b, con esto obtendría el mcd de a y b.

4.3.2 Sustento matemático:

En este algoritmo se llega a la solución más que todo restando y se basan en tres pasos principales:

Si a y b son pares $\Rightarrow mcd(a,b) = 2mcd(\frac{a}{2}, \frac{b}{2})$.
Si a es par y $b \Rightarrow mcd(a,b) = 2mcd(\frac{a}{2}, b)$.
Si a y b son impares y $a < b \Rightarrow mcd(a,b) = mcd(b-a, a)$
Si a y b son impares y $a > b \Rightarrow mcd(a,b) = mcd(a-b, b)$.

Se aplican los pasos según correspondan hasta que a sea igual a 0. Por último se multiplica g por b.

$\frac{a-b}{2} = 0 \Rightarrow mcd = g \times b$

4.3.3 Pseudo-algoritmo

- Se ingresan dos números enteros positivos a y b donde a
- Entero $g = 1$
- Mientras a módulo 2 = 0 y b módulo 2 = 0
  entonces $a = a/2, b = b/2, g = g \times 2$
- Mientras a sea diferente de 0
  entonces
  Mientras $amodulo2 = 0$

Entonces $a = a/2$
Mientras $bmodulo2 = 0$
Entonces $b = b/2$
Entero t es igual a la mitad del valor absoluto de $a - b$
Si $a \geq b$
Entonces $a = t$

4.3.4 Seguimiento numérico

Valores iniciales de las operaciones:
x=1764, y=868
g=1
Al ser ambos pares lo que sigue es dividir ambos entre dos y multiplicar la g por el mismo:
x/2=882 , y/2=434
g*2=2
Nuevamente ambos son pares, por ende, volvemos a dividir ambos entre dos y multiplicar la g por 2:
x/2=441 , y/2=217
g*2=4
Al ser ambos impares lo que hacemos es restarle al mayor el menor numero, en este caso la x es mayor por ende x=x-y y almacenamos el valor en x, g se mantiene igual:
x=x-y=224 , y=217
g=4
En este caso tenemos que solo 1 de ellos es par, el numero x, por ende este numero lo dividimos entre 2:
x/2=112 , y=217
g=4
Al igual que antes tenemos que x es par, lo que haremos es dividirlo entre 2 hasta que obtengamos un numero impar:
.
.
.
x=7 , y=217
g=4
Ahora tenemos dos números impares, por ende, se le resta el menor al mayor y se almacena en la variable del mayor, en este caso, y
x=7, y=y-x=210
g=4
Seguido de eso nos encontramos con un numero par y, procedemos a dividirlo entre 2 hasta obtener un numero impar
x=7, y=105
g=4
Procedemos al encontrarnos con dos números impares restándole el menor al

mayor:
x=7, y=y-x=98
g=4
Y obtenemos un y par, por ende dividimos entre dos hasta obtener un numero impar:
x=7, y/2=49
g=4
Nos encontramos con 2 impares por ende restamos el menor al mayor x=7, y=y-x=42
g=4
Y tenemos un número par, lo dividimos entre 2: x=7, y/2=21
g=4
Luego tenemos dos impares, restamos el menor al mayor: x=7, y=y-x=14
g=4
Ahora tenemos un par, lo dividiremos entre 2: x=7, y=7
g=4
Finalizando y al tener en ambos 7 debemos tener en cuenta que el algoritmo termina cuando x es 0 por ende al ser ambos impares lo que haremos es restarle a x:
x=x-y=0, y=7
g=4
Una vez conseguido el x=0 lo que haremos sera calcular el mdc multiplicando el "y" restante con el "g":
mcd=y*g mcd=7*4=28

mcd(1764,868)=28

4.3.5 Rastreo

4.3.6 Implementación en c++

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

using namespace std;
using namespace NTL;

ZZ binary_gcd(ZZ a, ZZ b){

    ZZ g(1);

    while (modulo(a, to_ZZ(2)) == 0 && modulo(b, to_ZZ(2)) == 0)
    {
        a /=2;
```

Steps	a	b	q
1	1764	868	1
2	882	434	2
3	441	217	4
4	224	217	4
5	112	217	4
6	7	217	4
7	7	210	4
8	7	105	4
9	7	98	4
10	7	49	4
11	7	42	4
12	7	21	4
13	7	14	4
14	7	7	4
12	7	0	4

Table 3: Valores de las variables en el programa

```
            b /=2;
            g *=2;
        }
        while(a!=0)
        {
            while(modulo(a,to_ZZ(2))==0)
                a/=2;
            while(modulo(b,to_ZZ(2))==0)
                b/=2;
            ZZ t = abs(a - b)/2;
            if(a >= b)
                a = t;
            else
                b = t;
        }

        return g*b;
}
```

4.4 Algoritmo de Lehmer del MCD

4.4.1 Definición

Es una variación del algoritmo de Euclides que se basa en el análisis del tamaño de las variables.

Cuando a y b tienen el mismo tamaño, la parte entera w de a/b suele ser de un solo dígito.

Entonces, supongamos que a, b son números enteros muy grandes, 'a,'b son números pequeños tales que: $a/b =' a/'b$ entonces la secuencia de cocientes producida por EA (a,b) y por EA ($'a,'b$) será el mismo al principio, siempre que esto sea así, se puede calcular EA ('a,b) en lugar de EA (a,b), que es mucho mas económico.

4.4.2 Sustento matemático

4.4.3 Pseudo-algoritmo

- Ingresa dos enteros m y n en representación en radix b, con x¿=y.
- Obtendrás el máximo común divisor de m y n

1. mientras $n \geq b$ has lo siguiente:
   (a) Coloca 'm, 'n para que sea el "high-order" digito de m, n, respectivamente ('n no puede ser 0)
   (b) $A = 1, B = 0, C = 0, D = 1$.
   (c) Mientras $('n + C) \neq 0$ y $('y + D) \neq 0$ has lo siguiente
      i. $q = floor((('x + A) = ('y + C)),' q = ('x + B) = ('y + D)$
      ii. Si $q \neq' q$ entonces corre al paso (d)
      iii. t=A-qC, A=C, C=t, t=B-qD, B=D, D=t.
      iv. t='x-q'y, 'x='y, 'y=t
   (d) Si B=0, entonces t=x mod y, x=y y=t;
   (e) De lo contrario, t=Ax +By, u=Cx +Dy, x=t, y=u.
2. Calcula $v = mcd(m; n$ usando el algoritmo de Euclides:
   - Ingresa x, y donde x ¿ y
   - mientras $b \neq 0$ has lo siguiente:
     - r= a mod b, a=b, b=r
3. Retorna(v)

4.4.4 Seguimiento numérico

4.4.5 Rastreo

4.4.6 Implementación en c++

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

using namespace std;
```

```
using namespace NTL;

unsigned length(ZZ n){

    int d=1;
    if(n<=9){
        return d;
    }
    else{
        return d+length(n/10);
    }
}

ZZ radix(ZZ B, ZZ c){

    long i=0;
    ZZ a[length(c)];
    ZZ za=c;
    ZZ zq=za/B;
    a[i]=za-zq*B;
    while(zq>0l){
        i++;
        za=zq;
        zq=za/B;
        a[i]=za-zq*B;
    }
    c=a[i];
    return c;
}

ZZ lehmer_gcd(ZZ a, ZZ b, ZZ base)
{
    ZZ xh, yh, q, qh, A, B, C, D;

    while(b >= base)
    {
        xh = radix(base,a);
        yh = radix(base,b);
        A = 1;
        B = 0;
        C = 0;
        D = 1;
        while(yh + C != 0 && yh + D != 0)
        {
            q = (xh + A)/(yh + C);
            qh = (xh + B)/(yh +D);
```

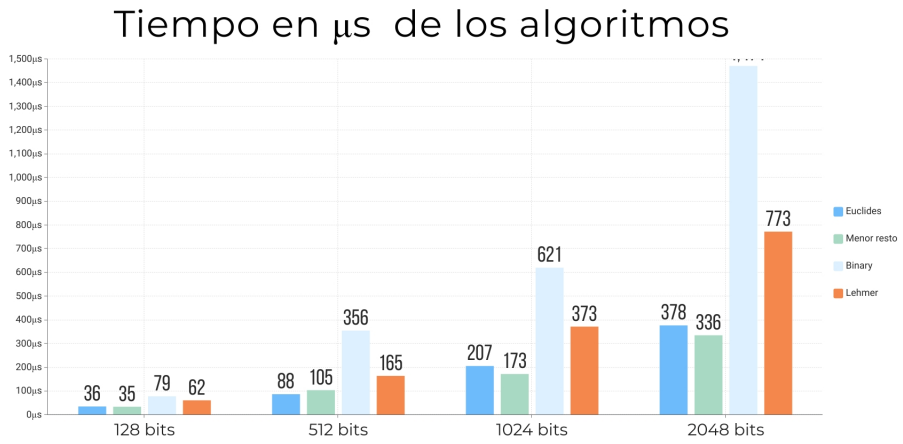
```
            if (q!=qh)
                break;
            else
            {   // swap
                ZZ t = A - q*C; A = C; C =t;
                t = B -q*D; B = D; D = t;
                t = xh - q*yh; xh = yh; yh = t;
            }
        }

        if (B == 0)
        {   // swap
            ZZ t = modulo(a, b);
            a = b;
            b = t;
        }
        else
        {   // swap
            ZZ t = A*a + B*b;
            ZZ u = C*a + D*b;
            a = t;
            b = u;
        }
    }

    return euclides_clasico(a, b);
}
```

4.5 Análisis de los algoritmos

4.5.1 Computadora donde se compararon

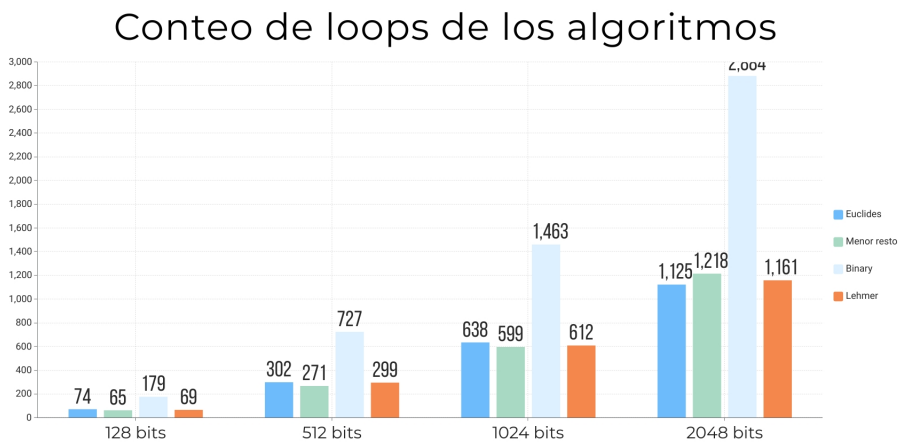
Intel(R) Core(TM) i7-4770 CPU @ 3.40GHz — Linux pop-os 5.11.0-7614-generic Ubuntu SMP $x86_64 GNU/Linux$

4.5.2 Comparación de los algoritmos por tiempo de ejecución vs Nro. de bits



Como observamos en el gráfico en todas las comprobaciones con los números en diferentes cantidades de bits se aprecia que el ganador es el algoritmo de menor resto siendo el segundo mas eficaz en el caso del tiempo, el algoritmo de Euclides clasico.

4.5.3 Comparación de los algoritmos por número de loops o bucles



En el caso de la comprobación por loops observamos de manera sorprendente que el algoritmo que tiene mas cantidad de loops es el binario, mientras que los demás algoritmos contienen una cantidad de loops similares e incluso menores a la mitad de la cantidad del algoritmo binario.

4.6 Conclusiones

-En promedio con el mejor resultado en cuanto a tiempo y cantidad de loops lo obtuvo el algoritmo de Euclides de menor resto, obteniendo un puntaje, si bien no tan drástico en cuanto al resto, suficiente para ser considerado para el uso, además de su simpleza matemática.

-El algoritmo binario para hallar el mcd, obtiene uno de los peores resultados, tanto en tiempo como cantidad de loops, el cual indica una pobre eficiencia, y posiblemente un descarte de elección para su uso.

-Los algoritmos para obtener el mcd de dos números, tienen eficiencia si es que se aplican correctamente con los números que fueron pensados, esto implica que habrá ciertas situaciones en los que podrían destacar.