Análisis de los Algoritmos para hallar el MCD y los coeficientes Lineales de Bezout

Becerra Sipiran, Cledy Elizabeth
Oviedo Sivincha, Massiel
Villanueva Borda, Harold Alejandro

Resumen:

Este documento es sobre el maximo comun divisor y sus versiones extendidas, el documento muestra que hay muchos algoritmos, algunos de estos son buenos al momento de hacer pocas iteraciones, y los otros hacen muchas iteraciones con mucho tiempo. Pero como vemos en el análisis de los algoritmos que algunos son más rápidos que los demás en pequeñas cantidades.

Palabras clave:

Algoritmo clásico de Euclides, Bishop para el GCD, Algoritmo de Euclides con el menor resto, Algoritmo binario para el GCD, Algoritmo de Lehmer para el GCD, Algoritmo extendido de Euclides, Algoritmo Binario extendido, Algoritmo de Euclides Extendido recursivo.

1. Introducción

En este documento presentaremos un análisis de los siguientes algoritmos :

- Algoritmos para hallar el GCD:
  - Euclides Clásico
  - Método Bishop
  - Euclides con Menor Resto
  - Binario del MCD
  - Lehmer
- Algoritmos para hallar las combinaciones lineales de Bezout:
  - Extendido de Euclides Clásico
  - Extendido de Euclides Recursivo
  - Binario Extendido

La investigación comenzará sobre el análisis de los algoritmos seleccionados para encontrar el GCD; así como también, el análisis de los algoritmos para hallar los coeficientes lineales de Bezout.El método de Euclides para encontrar GCD a menudo es considerado como el abuelo de todos los algoritmos en teoría de números.

El Máximo Común Divisor(MCD) de dos entero no negativos s A y B diferentes de 0 es el entero más grande que divide a ambos. Es conveniente establecer que el MCD (0,0) = 0.

La identidad de Bézout o Lema de Bezout es un teorema elemental de teorías de números el cual enuncia que si a y b son números enteros diferentes de cero con máximo común divisor d, entonces existen enteros x e y tales que:

$$ax+by=d$$

Dicho de otra manera, para todo a y b, existen un x y un y tales que:

$$ax+by=MCD(a,b)$$

Donde d es el máximo común divisor de (a,b).
Más aún, MCD(a,b) es el elemento mínimo positivo del conjunto de combinaciones lineales enteras {ax + by}.
La identidad fue nombrada en honor del matemático francés Étienne Bézout (1730-1783).

Algunas propiedades del MCD:

1- Cada común divisor de A y B divide al MCD en A y B.

2- Si m es un entero, el MCD(mA,mB) = m MCD(A,B). EX: MCD(2,3) = 1 , MCD(2*2,3*2) = 2* MCD(2,3) = 1*2 = 2 donde MCD(4,6) = 2.

3- Si m es diferente a cero y común divisor de A and B entonces MCD(A/m,B/m) = MCD(A,B)/m. EX: MCD(6,12) =6 , 6 y 12 divide en 2, MCD(6/2,12/2) = MCD(3,6) = 3 equivalente a MCD(6,12) = 6/2 = 3.

4- MCD de tres números MCD(A,B,C) puede expresarse como MCD(MCD(A,B),C) = MCD(A,MCD(B,C)). EX: MCD(2,4,6) = 2 = MCD(MCD(2,4) ,6) = MCD(2,6) = 2 = MCD(2,GCD(4,6))= MCD(2,2) = 2.

5- MCD(A,B) = MCD(B,A). EX:MCD(6,12)= 6 = MCD(12,6).

6- El MCD de A y B está relacionado con su mínimo común múltiplo MCM(A,B): donde MCD(A,B) * MCM(A,B) = AB. EX: MCD(18,12) =6 , MCM(18,12) = 36 , 6 * 36 = 216 = 18*12.

7- MCD(A,B) = MCD(A,B+An) para todos los enteros n . EX: MCD(2,4) = 2 , MCD(2,4+2*2) = MCD(2,8) = 2 , MCD(2,4+2*3) = MCD(2,10) = 2[3].

2. Algoritmo de Euclides

2.1. Algoritmo de Euclides Clásico

Definición:

El algoritmo de Euclides encuentra el máximo común divisor de dos enteros y se basa en el siguiente hecho: Si a y b son números enteros positivos con a > b, entonces mcd (a, b) = mcd (b, a mod b).

Demostración:

Este algoritmo usa divisiones y restas y está basado principalmente en las identidades:

$$\text{mcd}(a, b) = \text{mcd}(b, a - bq), \text{mcd}(r, 0) = r,$$

de tal manera que si $a = bq + r_1$ y $b = r_1 q_1 + r_2$ con $0 \leq r_2 < r_1 < b$,

$$\text{mcd}(a, b) = \text{mcd}(b, r_1) = \text{mcd}(r_1, r_2),$$

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 8.

Formalmente: Sean a y b números naturales, b 6= 0. Aplicando el algoritmo de la división se obtiene una sucesión finita r_1 ,r2,...,r_n definida por:

$$a = bq_1 + r_1, 0 \leq r_1 < r_0$$

$$b = r_1 q_2 + r_2, 0 \leq r_2 < r_1$$

$$r_1 = r_2 q_3 + r_3, 0 \leq r_3 < r_2$$

…

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

$$r_{n-1} = r_n q_{n+1} + 0$$

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

Algoritmo:

INPUT: dos enteros a y b no negativos, donde a ≥ b.
OUTPUT: el máximo común divisor de a y b.
1. While b != 0:
    1.1 Set r←a mod b, a←b, b←r.
2. Return(a).

Seguimiento numérico:

EJEMPLO:
(a=12378, b=3054).....12378 = 4(3054) + 162
(a=3054, b=162).....3054 = 18(162) + 138
(a=162, b=138)..... 162 = 1(138) + 24
(a=138, b=24)..... 138 = 5(24) + 18
(a=24, b=18)..... 24 = 1(18) + 6
(a=18, b=6)..... 18 = 3(6) + 0

MCD(12378,3054) = 6

Implementación:

```
ZZ modulo(ZZ a, ZZ b){
        ZZ q = a/b;
        if (b < ZZ(0)) { q++; }
        return  a - b * q;
}
ZZ euclides(ZZ a, ZZ b){

        ZZ r;

        if (a == ZZ(0)) { return b; }

        else

        {

                while(b != ZZ(0))

                {

                        r = modulo(a,b);

                        a = b;

                        b = r;

                }

        }

        return a;

}
```

Pruebas:
La siguiente tabla muestra las pruebas en diferentes sistemas operativos.

Procesador/Bits	AMD Ryzen 7 4800H 2.90 GHz	Intel(R) i5-6200U 2.30GHz	Intel(R) i3-3110M 2.40GHz
16	0.028 s	0.070 s	0.285 s
32	0.042 s	0.155 s	0.355 s
64	0.102 s	0.267 s	0.388 s
128	0.384 s	0.617 s	0.791 s
256	1.257 s	2.075 s	2.329 s
512	5.080 s	8.130 s	8.358 s
1024	18.732 s	29.323 s	30.392 s
2048	81.118 s	114.072 s	121.985 s

2.2. Método Bishop's

Definición:

Si un número grande y un número pequeño son múltiplos de K, entonces grande - pequeño es un múltiplo de K. tenga en cuenta que grande-pequeño es menor que grande, por lo que hemos reducido el problema a uno más fácil de resolver. Entonces necesitamos el mayor múltiplo de grande: pequeño y pequeño... y así sucesivamente.

Demostración:

Si el mcd(A,B) = K, donde A > B, entonces:

A = mK y B = nK, de ello podemos deducir que, A - B = (m - n)K.

Por lo tanto, K es también un divisor de A-B.

Algoritmo:

```
1- input two positive integers X,Y
2- while (x != y) do
        if (x>y) then
                x=x-y
        else
        {
                temp = y
                y=x
                x=temp
        }
return(x)
```

Implementación:

```
ZZ bishop(ZZ x, ZZ y){
        ZZ temp;
        while (x != y){
                if(x > y)
                        x = x - y;
                else{
                        temp = y;
```

```
                    y = x;
                    x = temp;
                }
        }
return x;
}
```

Pruebas:

La siguiente tabla muestra las pruebas en diferentes sistemas operativos.

Procesador/Bits	AMD Ryzen 7 4800H 2.90 GHz	Intel(R) i5-6200U 2.30GHz	Intel(R) i3-3110M 2.40GHz
16	0.065 s	0.240 s	0.331 s
32	0.280 s	0.573 s	0.675 s
64	0.737 s	1.212 s	1.548 s
128	5.349 s	8.633 s	8.706 s
256	9.317 s	15.020 s	14.633 s
512	79.031 s	125.997 s	106.659 s
1024	242.531 s	393.299 s	335.152 s
2048	955.700 s	1440.019 s	1010.608 s

2.3. Algoritmo de Euclides con menor resto

Definición:

Fue establecido por Kronecker . Basada en la versión "clásica" del algoritmo de euclides pero con una pequeña variación en cuanto a la reducción de pasos o divisiones en cada nuevo resto ; a lo que se definió como "menor resto" teniendo en cuenta que el número divisiones serán menores o iguales que el algoritmo clásico de Euclides.

Sean a, b $\in$ Z con b $\neq$ 0. Sea q $\in$ Z definido como:

a= b (||a/b||) +r_1 con $0 \leq r_1 <$ |b|

a= b (||a/b|| + 1) - r_2 con $0 \leq r_2 <$ |b|

Donde r = mín{r1,r2} . Entonces , evaluamos si r_1 o r_2 par cumplir que $0 \leq r \leq |b|/2$.

Demostración:

A Través del anterior teorema sacamos ecuaciones del resto :

r=Min{r_1,r_2}=Min{$|a - b\,.\,||a/b||\,|$, $|a - b\,.\,||a/b|| + 1\,|$}

En efecto se cumple:

$$a - b||a/b|| \leq b\,.\,(||a/b|| + 1) - a$$

$$2a/b \leq 2.\,||a/b|| + 1$$

como: $a/b \leq ||a/b|| + pfrac(a/b)$

$$pfrac(a/b) \leq 1/2$$

Entonces:

En el caso en que a ≥ 0 y b > 0, se afirma :

$||a/b|| + 1\,|$ si pfrac(a/b) $\leq 1/2$

$||a/b|| + 1\,|$ si pfrac(a/b) $\leq 1/2$

El menor resto es :

r=|a-b . ||a/b + ½|| |

Algoritmo:

1. Ingresar dos enteros a, b.

2. If a← 0

    2.1.Then c ← b

3. Else c← a d← b

    3.1. While d<> 0

        3.1.1. r ← c-d*Int(c/d+1/2) //'rem q= Int(c/d+1/2)

        3.1.2. c ← d , d ← r

        3.1.3. Wend

3.2. End If

Seguimiento numérico:

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

Implementación:

```
ZZ mcdMenorResto(ZZ a,ZZ b){
  ZZ c,d, r;
  if(a==0){
    c=b;
  }
  else{
    c=a;d=b;
    while(d!=0){
      float aux1=0,aux2=0;
      conv(aux1, c);conv(aux2, d);
      float aux=aux1/aux2+0.5;
      r=c-(d*ZZ(aux));

coût<<c<<"="<<d<<"*"<<ZZ(aux)<<"+("<<r<<")"<<
endl;

      c=d;
      d=abs(r);
    }
  }
  return abs(c);
}
```

Pruebas:

La siguiente tabla muestra las pruebas en diferentes sistemas operativos.

Procesador/Bits	AMD Ryzen 7 4800H 2.90 GHz	Intel(R) i5-6200U 2.30GHz	Intel(R) i3-3110M 2.40GHz
16	0.024 s	0.077 s	0.257 s
32	0.037 s	0.109 s	0.268 s
64	0.075 s	0.389 s	0.343 s
128	0.245 s	0.395 s	0.638 s
256	---	---	---
512	---	---	---
1024	---	---	---
2048	---	---	---

2.4. Algoritmo Binario del MCD

Definición:

Este algoritmo fue descubierto por el físico Israelí J. Stein .Este último algoritmo solo usa restas, prueba de paridad y divisiones por dos (mucho menos costosas que las divisiones que requiere el algoritmo de Euclides). Si los números vienen codificados en binario, teóricamente habría una mejora del 60%[4] en la eficiencia, además de que es el más popular; algunos autores le llaman “algoritmo binario de Euclides”. Desde el punto de vista del computador la división por dos (y también la multiplicación por 2) se hace en representación binaria, así que solo se requiere un desplazamiento de bits

Enfoque matemático sobre el que se sustenta ([7] pág 163):

El algoritmo opera con los siguientes teoremas:

a) Si a, b son pares:

$$mcd(a, b) = 2\ mcd(a/2, b/2)$$
Regla 1

b) Si a es par y b impar:

$$mcd(a, b) = mcd(a/2, b)$$
Regla 2

c) Si a, b son impares:

$$mcd(a, b) = mcd(|a - b|/2, b) = mcd(|a - b|/2,$$
Regla 3

La Regla 3 se aplica como :

$$mcd(a, b) = mcd(|a - b|/2, Mín\{a, b\}).$$

Demostración de las reglas ([7] pág 167):

Supongamos que $a, b \in Z$, $a \geq 0$, $b > 0$. Si a y b son pares, aplicamos la regla 1, digamos s veces, hasta que alguno de los dos sea impar. Al final hay que multiplicar por 2^s como compensación por haber usado la regla 1, s veces. Si todavía a o b es par, aplicamos la regla 2 hasta que ambos queden impares. Siendo los dos impares, aplicamos la regla 3 y luego alternamos las reglas 2 y 3 conforme el cociente $|a - b|/2$ sea par o impar.

Prueba de la regla 1:

Sean $d = mcd(a, b)$ y $d' = mcd(a/2, b/2)$. Por el teorema de Bezout, d es la mínima combinación lineal positiva de a y b. Si $d = ax + by > 0$, como a y b son pares, d es par y podemos dividir a ambos lados por 2, $d/2 = a/2\ x + b/2 \geq d'$ por ser d' es la mínima combinación lineal positiva de a/2 y b/2. Por tanto $d \geq 2d'$. De manera análoga se prueba que $2d' \geq d$, con lo cual se concluye $d = 2d'$.

Prueba de la regla 2:

Sean $d = mcd(a, b)$ y $d' = mcd(a/2, b)$. Como d|b y b es impar, d es impar. Si $a = kd$, como a es par y d impar, tenemos que k es par, entonces $a/2 = (k/2)d$, por tanto d|(a/2) y d|b, es decir, $d \leq d'$. Ahora como d' |(a/2), $a/2 = k\ 'd\ '$, es decir $a = 2k\ 'd\ '$, por tanto d'|a, entonces d'|a y d'|b, así $d\ ' \leq d$. $\therefore\ d = d'$.

Prueba de la regla 3:

Sean $d = mcd(a, b)$ y $d' = mcd(|a - b|/2, b)$. Como d|a y d|b entonces d||a − b|. Como a es impar, d es impar pero |a − b| es par (a y b son impares), luego si |a − b| = kd, k debe ser par, así que podemos dividir por dos a ambos lados, |a − b|/2 =(k/2)d. Por tanto d|(|a − b|/2) y d|b, entonces $d \leq d'$. De manera similar, si $b = k'd\ '$ y |a − b| = 2k"d ' , sustituyendo b en la última ecuación obtenemos que d'|a. Por tanto $d' \leq d$ y entonces $d = d'$.

Algoritmo ([5]pág 606):

INPUT: two positive integers x and y with $x \geq y$.
OUTPUT: gcd(x, y).
1. g←1.
2. While both x and y are even do the following:
x←x/2, y←y/2, g←2g.
3. While x 6= 0 do the following:
   3.1 While x is even do: x←x/2.
   3.2 While y is even do: y←y/2.
   3.3 t←|x − y|/2.
   3.4 If $x \geq y$ then x←t; otherwise, y←t.
4. Return(g · y).

Seguimiento Numérico:

La siguiente tabla muestra los pasos realizados por el algoritmo binario para calcular mcd(89, 44) = 1 y mcd(8,48)=8.

mcd (89,44)	=	mcd (22,89),	por Regla 2
	=	mcd (11,89),	por Regla 2
	=	mcd (39,11),	por Regla 3
	=	mcd (14,11),	por Regla 3
	=	mcd (7,11),	por Regla 2
	=	mcd (2,7),	por Regla 3
	=	mcd (1,7),	por Regla 2
	=	mcd (3,1),	por Regla 2
	=	mcd (1,1),	por Regla 3
	=	mcd (0,1),	por Regla 3
	=	1	

mcd (8,48)	=	2 · mcd (4,24),	por Regla 1
	=	4 · mcd (2,12),	por Regla 1
	=	8 · mcd (1,6)	por Regla 1
	=	8 · mcd (1,3),	por Regla 2
	=	8 · mcd (1,1),	por Regla 3
	=	8 · mcd (0,1),	por Regla 3
	=	8	

Por supuesto, en cálculo manual terminamos cuando obtenemos mcd (0, d) = d o mcd (1, d) = 1

Implementación:

```
ZZ mod(ZZ a,ZZ b){
   ZZ r=a-(b*(a/b));
   if(r<0)r=b-r;
   return r;
}
bool even(ZZ a){
   if(mod(a,ZZ(2))==0) return 1;
   return 0;
}
ZZ ValAbs(ZZ a){
   if (a<0) return (a*-1);
   return a;
}
ZZ Binary_GCD1(ZZ x,ZZ y){
   ZZ g=ZZ(1);
   while(even(x)&&even(y)){
      x=x/2;
      y=y/2;
      g=2*g;
      cout<<"x: "<<x<<" y: "<<y<<" g: "<<g<<endl;
   }

while(x!=0){
      while(even(x)){
         x/=2;
         cout<<"x: "<<x<<endl;
      }
      while(even(y)){
         y/=2;
         cout<<"y: "<<y<<endl;
      }
      ZZ t=ValAbs((x-y))/2;
      cout<<"t: "<<t<<endl;
      if(x>=y) x=t;
      else y=t;
      cout<<"x: "<<x<<" y: "<<y<<endl;
   }
   return (g*y);
}
```

Pruebas:

La siguiente tabla muestra las pruebas en diferentes sistemas operativos.

Procesador/Bits	AMD Ryzen 7 4800H 2.90 GHz	Intel(R) i5-6200U 2.30GHz	Intel(R) i3-3110M 2.40GHz
16	0.023 s	0.038 s	0.267 s
32	0.037 s	0.076 s	0.367 s
64	0.086 s	0.420 s	0.422 s
256	1.109 s	1.725 s	2.059 s
512	3.874 s	6.070 s	6.775 s
1024	16.254 s	25.416 s	25.798 s
2048	65.421 s	96.078 s	86.125 s

2.5. Algoritmo de Lehmer

Definición:

Un enfoque alternativo para acelerar el Algoritmo de Euclides se debe a Lehmer. Uno nota que cuando a y b tienen el mismo tamaño, la parte entera w de a / b suele ser de un solo dígito.

Suponga que a, b son números muy grandes, ^ a, ^ b son números pequeños tal que: a / b = ^ a / ^ b. Luego la secuencia de cocientes producida por EA (a, b) y por EA (^ a, ^ b) será la misma al principio, hasta que uno pueda calcular EA (^ a, b) en lugar de EA (a, b) que es mucho más económico [7] .

Demostración:

Teorema de la división: A = B*q +r

Identidades de Bezout:

Si A, B son dos enteros no ambos cero, existen x,y ∈ Z (posiblemente no únicos) tales que A(x) + B(y) = gcd(A, B).

Sean 0 < B < A enteros. El algoritmo de Euclides consiste en computarizar (A, B): (A0, A1,..., An , An+i) definido por las relaciones: [9]

A0 = A, A1 = B, Ai+2 = A; mod Ai+1, An+1 = 0. Es sabido que An = gcd(A, B).

En el extendido, las primeras y segundas secuencias simultáneas (U0, U1, ... , Un+1) y (V0, V1, ... , Vn+1), donde (Uo, Vo) = (1,0), (U1, V1)= (0, 1),

Congruencia: x mod n ≅ y mod n

Seguimiento numérico:

X	Y	q1	q2	A	B	C	D
768454923	542167814	1	1	1	0	0	1
542167814	226287109	2	2	0	1	1	-1
226287109	89593596	2	2	1	-1	-2	3
89593596	47099917	1	1	-2	3	5	-7
47099917	42493679	1	1	5	-7	-7	10
42493679	4606238	9	9	-7	10	12	-17
4606238	1037537	4	4	12	-17	-115	163
1037537	456090	2	2	-115	163	472	-669
456090	125357	3	3	472	-669	-1059	1501
125357	80019	1	1	-1059	1501	3649	-5172
80019	45338	2	1	3649	-5172	-4708	6673

Algoritmo:

1- INPUT: two positive integers x and y in radix b representation, with x >= y.

2- OUTPUT: gcd(x; y).

1. While y >= b do the following:

    1.1 Set x^, y^ to be the high-order digit of x, y, respectively (y^ could be 0).

    1.2 A=1, B=0, C=0,=D=1.

    1.3 While (y^ + C) != 0 and (y^ + D) != 0 do the following:

        q=floor((x^ + A)=(y^ + C)), q^=(x^ + B)=(y^ + D)

        If q != q^ then go to step 1.4.

        t=A - qC, A=C, C=t, t=B - qD, B=D, D=t.

        t=x^ - qy^, x^=y^, y^=t.

    1.4 If B = 0, then t=x mod y, x=y, y=t ; otherwise, t= Ax + By, u=Cx + Dy, x=t , y=u.

2. Compute v = gcd(x; y) using Euclid's Algorithm:

    input x,y where x > y

    while b != 0 do the following:

    r = a mod b , a=b , b=r

3. Return(v)

Implementación:

```
ZZ modulo(ZZ a, ZZ b)

{

    ZZ q = a/b;

    if (b < ZZ(0)) { q++; }

    return  a - b * q;

}

ZZ euclides(ZZ a, ZZ b)

{

    ZZ r;

    if (a == ZZ(0)) { return b; }

    else{

        while(b != ZZ(0))

        {

            r = modulo(a,b);

            a = b;
            b = r;

        }

    }

    return a;
```

```
}

ZZ lehmer(ZZ x, ZZ y){
        ZZ A = ZZ(1), B = ZZ(0), C = ZZ(0), D = ZZ(1);
        ZZ q;
        while(y+C != ZZ(0) and y+D != ZZ(0))
        {
                q = (x+A) / (y+C);
                if(q != (x+B) / (y+D)) break;
                ZZ t = A - q*C;

                A = C;
                C = t;
                t = B-q*D;
                B = D;
                D = t;
                t = x-q*y;
                x = y;
                y = t;

                cout << "[" << A << "," << B << "," << C << "," << D << "]" << endl;
        }
        if(B == ZZ(0)){
                ZZ t = modulo(x,y);
                x = y;
                y = t;
        }
        else{
                ZZ t = A*x + B*y;
                ZZ u = C*x + D*y;
                x = t;
                y = u;
        }
        return euclides(x,y);
}
```

Pruebas:

La siguiente tabla muestra las pruebas en diferentes sistemas operativos.

Procesador/Bits	AMD Ryzen 7 4800H 2.90 GHz	Intel(R) i5-6200U 2.30GHz	Intel(R) i3-3110M 2.40GHz
16	0.020 s	0.036 s	0.237 s
32	0.024 s	0.048 s	0.307 s
64	0.038 s	0.065s	0.351 s
128	0.129 s	0.215 s	0.504 s
256	0.489 s	0.741 s	1.200 s
512	1.796 s	2.804 s	3.292 s
1024	6.452 s	10.241 s	10.333 s
2048	27.226 s	41.090 s	36.710

3.Algoritmo Extendido de Euclides

3.1. Algoritmo Extendido de Euclides Clásico:

Definición:

Es una versión del algoritmo euclidiano, la entrada de este algoritmo son dos enteros positivos m, n, el algoritmo calcula el MCD así como los enteros A y B tales que Am + Bn = MCD (m, n) [6].

El enfoque matemático sobre el que se sustenta([7]pág. 32):

El siguiente teorema establece la llamada “Identidad de Etienne Bezout” aunque el resultado lo descubrió primero el francés Claude Gaspard Bachet de Méziriac (1581-1638).

Teorema (Identidad de Bézout):

Si a, b son dos enteros no ambos cero, existen s,t ∈ Z (posiblemente no únicos) tales que

sa + tb = mcd(a, b)

Prueba: Sea A el conjunto de combinaciones lineales enteras de a y b, A = {u a + v b : u,v ∈ Z}. Este conjunto tiene números positivos, negativos y el cero. Sea m = a x + b y el más pequeño entero positivo en A.

supongamos que a = mq + r con 0 ≤ r < m. Entonces:

$0 \le r = a - mq = a - (a\,x + b\,y)q = (1 - qx)a + (-qy)b$

Así, r es una combinación lineal de a y b, es decir, r ∈ A. Pero 0 ≤ r < m, así que la única posibilidad es que r = 0 por ser m el mínimo entero positivo en A.

Luego, a = mq y m|a. De manera análoga podemos establecer que m|b. Sea d = mcd(a, b). Como m es común divisor de a y b, entonces d ≥ m. Pero, como a = k1d y b = k2d entonces m = ax + by = (xk1 + yk2)d > 0, por lo tanto m ≥ d. Así que m = d.

COROLARIO:

a. El mcd(a, b) es el más pequeño entero positivo de la forma sa + tb; s,t ∈ Z. En particular, mcd(a, b) = 1 si y sólo si existen x,y ∈ Z tal que ax + by = 1.

b. Si a|bc y mcd(a, b) = 1 entonces a|c.

Prueba para el corolario b: Como mcd(a, b) = 1, existen x,y ∈ Z tal que xa + by = 1. Multiplicando por c a ambos lados, acx + bcy = c.

Como a|ac y a|bc entonces a|(acx + bcy). ∴ a|c

Calcular t y s: La ecuación sa + tb = mcd(a, b) no tiene solución única para s,t enteros. Se puede obtener una solución despejando los residuos, en el algoritmo de Euclides, y haciendo una sustitución hacia atrás. Consideremos la sucesión r1,r2,...,rn del algoritmo de Euclides. Todos estos residuos son una combinación lineal entera de a y b : En efecto, como r1 = a − bq0 y b = r1q1 + r2 entonces r2 es combinación lineal de a y b. Como r1 y r2 son combinaciones lineales de a y b y como r1 = r2q2 + r3 entonces r3 es combinación lineal de a y b. Continuando de esta manera, como ri−1 y ri−2 son combinaciones lineales de a y b y como ri−2 = ri−1q2 + ri (i = 2,...,n), entonces rn es combinación lineal (mínima) de a y b.

Algoritmo:

INPUT: two non-negative integers a and b with a ≥ b.
OUTPUT: d = gcd(a, b) and integers x, y satisfying ax + by = d.
1. If b = 0 then set d←a, x←1, y←0, and return(d,x,y).
2. Set x2←1, x1←0, y2←0, y1←1.
3. While b > 0 do the following:
3.1 q←⌊a/b⌋, r←a − qb, x←x2 − qx1, y←y2 − qy1.
3.2 a←b, b←r, x2←x1, x1←x, y2←y1, and y1←y.
4. Set d←a, x←x2, y←y2, and return(d,x,y).
Este algoritmo tiene un tiempo de ejecución de $O(\log(n)^2)$ operaciones de bit [3].

Seguimiento numérico:

q	r	x	y	a	b	x_2	x_1	y_2	y_1
–	–	–	–	4864	3458	1	0	0	1
1	1406	1	−1	3458	1406	0	1	1	−1
2	646	−2	3	1406	646	1	−2	−1	3
2	114	5	−7	646	114	−2	5	3	−7
5	76	−27	38	114	76	5	−27	−7	38
1	38	32	−45	76	38	−27	32	38	−45
2	0	−91	128	38	0	32	−91	−45	128

Implementación:

```
ZZ Euclides_ext(ZZ a,ZZ b,ZZ &x,ZZ &y){

        if(b == 0) {

    ZZ d=a;

     x = ZZ(1);

     y = ZZ(0);

    return d;

}

   ZZ q=a/b;
```

```
        ZZ x1=ZZ(1);ZZ x2=ZZ(0);ZZ y1=ZZ(0);ZZ y2=ZZ(1);

    ZZ r=a-(q*b);x=x1-(q*x2);

        y=y1-(q*y2);

        while(r>0){

                q=a/b;

                r=a-(q*b);
                x=x1-(q*x2);
                y=y1-(q*y2);
                a=b;
                b=r;
                x1=x2;
                x2=x;
                y1=y2;
                y2=y;

        }

        ZZ d=a;x=x1;y=y1;return d;

}
```

Pruebas:

La siguiente tabla muestra las pruebas en diferentes sistemas operativos.

Procesador/Bits	AMD Ryzen 7 4800H 2.90 GHz	Intel(R) i5-6200U 2.30GHz	Intel(R) i3-3110M 2.40GHz
16	0.026 s	0.057 s	0.243 s
32	0.034 s	0.060 s	0.450 s
64	0.092 s	0.153 s	0.350 s
128	0.278 s	0.461 s	0.651 s
256	0.818 s	1.263 s	1.550 s
512	3.468 s	5.414 s	5.755 s
1024	13.187 s	20.737 s	20.079 s
2048	53.548 s	80.734 s	80.055 s

3.2. Algoritmo Extendido de Euclides Recursivo:

Definición:

Es una versión del algoritmo euclidiano, la entrada de este algoritmo son dos enteros positivos m, n, el algoritmo calcula el MCD así como los enteros A y B tales que Am + Bn = MCD (m, n) [6].

El enfoque matemático sobre el que se sustenta:

Si mcd(a,n) = 1 entonces podemos hallar:

$$x,y \in Z \,/\, xa + yn = 1$$

Demostración:

En esta última ecuación tomamos congruencia módulo n:

Tenemos: 1 = xa + yn

Y también xa + yn ≡ xa(mod n)

Por definición de congruencia:

1) a ≡ b(mod n) si a y b dejan el mismo resto en la división por n

2) a ≡ b(mod n) si a−b es múltiplo de n

Aplicando la segunda definición tenemos que:

$$(xa + yn) - xa = yn$$

es múltiplo de n, por lo tanto:

$$xa + yn \equiv xa(\text{mod } n)$$

Pero como xa + yn = 1, entonces tenemos que :

$$1 \equiv xa(\text{mod } n).$$

En conclusión, 1 = xa + yn ≡ xa(mod n), pues yn es múltiplo de n, por lo tanto congruente con 0.

Obtuvimos xa ≡ 1(mod n), es decir, x es un inverso modular de a. [6]

Algoritmo:

INPUT:dos enteros , cumple a,b>0 y x,y para guardar las variables
OUTPUT: d = mcd(a, b) y valores x,y
1. If b = 0 then set x←1, y←0, and return(a).

2. Set gcd, x1, y1 ← xGCD(b, a % b, x1, y1)
3. Set x←y1, y←x1 - (a/b) * y1, and return gcd

Seguimiento numérico:

r	q	x	y
99		1	0
78	1	0	1
21	3	1	-1
15	1	-3	4
6	2	4	-5
3	2	-11	14
0			

Implementación:

```
ZZ mod(ZZ a,ZZ b){

   ZZ r=a-(b*(a/b));
   if(r<0)r=b-r;
   return r;
}
ZZ  xGCD(ZZ a, ZZ b, ZZ &x, ZZ &y) {

   if(b == 0) {

          x = ZZ(1);

           y = ZZ(0);
          return a;

   }

            ZZ x1, y1, gcd = xGCD(b, mod(a, b), x1, y1);
              x = y1;
              y = x1 - (a/b) * y1;
          return gcd;

}

int main() {

   ZZ x,y;

   cout<<xGCD(ZZ(78),ZZ(99) ,x, y)<<endl;

   cout<<"x:"<<x<<endl<<"y:"<<y<<endl;

   return 0;}
```

Pruebas:
La siguiente tabla muestra las pruebas en diferentes sistemas operativos.

Procesador/Bits	AMD Ryzen 7 4800H 2.90 GHz	Intel(R) i5-6200U 2.30GHz	Intel(R) i3-3110M 2.40GHz
16	0.049 s	0.060 s	0.292 s
32	0.077 s	0.170 s	0.348 s
64	0.187 s	0.190 s	0.422 s
128	0.775 s	0.820 s	1.076 s
256	2.580 s	2.575 s	3.150 s
512	10.469 s	10.595 s	10.865 s
1024	38.531 s	38.723 s	36.555 s
2048	118.001 s	152.148 s	140.999 s

3.3. Algoritmo Binario Extendido del GCD

Definición:

Dados los números enteros x e y, el algoritmo extendido de Euclides calcula los números enteros a y b tales que ax + by = v, donde v = mcd (x, y). Tiene el inconveniente de requerir divisiones de precisión múltiple relativamente costosas cuando x e y son números enteros de precisión múltiple. El algoritmo binario extendido elimina este requisito a expensas de más iteraciones.

Demostración:

Enfoque matemático sobre el que se sustenta:

Para su desarrollo, se basa en el mismo enfoque matemático que para su contraparte par hallar el gcd y a su vez, para hallar los coeficientes emplea la estrategia arriba-abajo de las identidades de Bézout.

Donde la identidad de Bézout usa el siguiente teorema:

Sean a y b números enteros con MCD (a, b) = d. Entonces existen enteros x e y tales que ax + by = d.

inicialice las dos primeras filas como se ilustra en la siguiente tabla. Es claro ver que en las dos primeras filas a · x + b · y = r [8].

$$q_{n+1} = r_{n-1}/r_n$$
$$r_{n+1} = r_{n-1} - r_n \cdot q_{n+1}$$
$$x_{n+1} = x_{n-1} - x_n \cdot q_{n+1}$$
$$y_{n+1} = y_{n-1} - y_n \cdot q_{n+1}$$

Algoritmo:

INPUT: two positive integers x and y.
OUTPUT: integers a, b, and v such that ax + by = v, where v = gcd(x, y).
1. g←1.
2. While x and y are both even, do the following: x←x/2, y←y/2, g←2g.
3. u←x, v←y, A←1, B←0, C←0, D←1.
4. While u is even do the following:
    4.1 u←u/2.
    4.2 If A ≡ B ≡ 0 (mod 2) then A←A/2, B←B/2; otherwise, A←(A + y)/2, B←(B − x)/2.
5. While v is even do the following:
    5.1 v←v/2.
    5.2 If C ≡ D ≡ 0 (mod 2) then C←C/2, D←D/2; otherwise, C←(C + y)/2, D←(D − x)/2.
6. If u ≥ v then u←u − v, A←A − C, B←B − D; otherwise, v←v − u, C←C − A, D←D − B.
7. If u =0, then a←C, b←D, and return (a, b, g · v); otherwise, go to step 4.

Eficiencia computacional del algoritmo binario extendido ([5]pág. 610):

(i) Las únicas operaciones de precisión múltiple necesarias para el algoritmo binario extendido son la suma y la resta. La división por 2 es simplemente un desplazamiento a la derecha de la representación binaria.

(ii) El número de bits necesarios para representar u o v disminuye en (al menos) 1, después de como máximo dos iteraciones de los pasos 4 - 7; por lo tanto, el algoritmo toma como máximo 2([lg x] + [lg y] + 2) de tales iteraciones.

Seguimiento numérico:

Sea x = 693 e y = 609; muestra los pasos para calcular los números enteros a, b, v tales que 693a + 609b = v, donde v = mcd (693, 609). El algoritmo devuelve v = 21, a = −181 y b = 206 ([5]pág. 610).

u	v	A	B	C	D
693	609	1	0	0	1
84	609	1	−1	0	1
42	609	305	−347	0	1
21	609	457	−520	0	1
21	588	457	−520	−457	521
21	294	457	−520	76	−86
21	147	457	−520	38	−43
21	126	457	−520	−419	477
21	63	457	−520	95	−108
21	42	457	−520	−362	412
21	21	457	−520	−181	206
0	21	638	−726	−181	206

Implementación:

```
ZZ mod(ZZ a,ZZ b){
   ZZ r=a-(b*(a/b));
   if(r<0)r=b-r;
   return r;
}
bool even(ZZ a){
   if(mod(a,ZZ(2))==0) return 1;
   return 0;
}
ZZ Binary_Extend(ZZ x,ZZ y){
   ZZ g=ZZ(1); ZZ a, b;
   while(x == y){
         x = x/2;
         y = y/2;
         g = g*2;
         }
         ZZ u=x, v=y,A=ZZ(1), B=ZZ(0), C=ZZ(0),
D=ZZ(1);
   while(u!=0){
         while(even(u)){
         u = u/2;
         if(even(A) and even(B)){
                  A = A/2;
                  B = B/2;
         }else{
```

```
            A = (A+y)/2;
            B = (B-x)/2;
        }
    }
    while(even(v)){
            v = v/2;
            if(even(C) and even(D)){
                C = C/2;
                D = D/2;

            }else{
                C = (C+y)/2;
                D = (D-x)/2;
            }
    }
    if(u >= v){
            u=u-v;
            A = A-C;
            B = B-D;
    }else{
            v= v-u;
            C = C-A;
            D = D-B;
    }
    if(u==0){
        a = C;
        b = D;
    }
  }
  cout<<"\n\t"<<x<<"("<<a<<")"<<"+"<<y<<"("
<<b<<")"<<"="<<v<<endl;
  cout<<"\tGCD: "<< v <<endl;
  cout<<"\tinversa: "<< a <<endl;
}
```

Pruebas:

La siguiente tabla muestra las pruebas en diferentes sistemas operativos.

Procesador/Bits	AMD Ryzen 7 4800H 2.90 GHz	Intel(R) i5-6200U 2.30GHz	Intel(R) i3-3110M 2.40GHz
16	0.522 s	1.696 s	1.291 s
32	0.914 s	2.289 s	2.182 s
64	2.089 s	3.157 s	3.139 s
128	6.660 s	8.081 s	8.599 s
256	23.681 s	25.789 s	26.717 s
512	79.889 s	93.421 s	99.795 s
1024	258.723 s	353.201 s	257.774 s
2048	0	0	0

4. Análisis of Algoritmos

Los análisis de los algoritmos se realizaron en tres diferentes procesadores (podrá ver los diferentes resultados en las pruebas realizadas al final de cada descripción del algoritmo) ,pero mismo sistema operativo:

- AMD Ryzen 7 4800H with Radeon Graphics 2.90 GHz
- Intel(R) core(TM) i5-6200U CPU @ 2.30GHz 2.40GHz
- Intel(R) Core(TM) i3-3110M CPU @ 2.40GHz 2.40 GHz

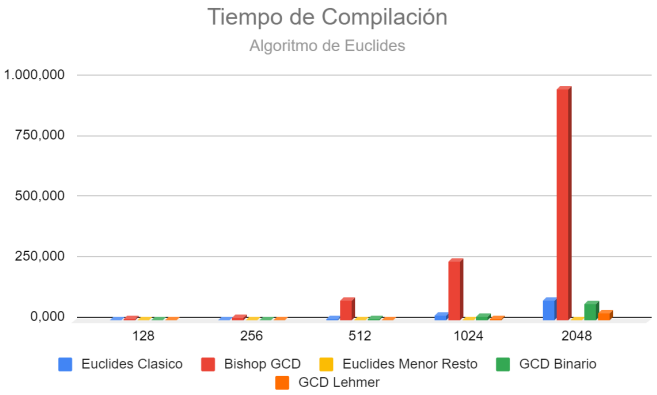
Teniendo en cuenta que para el análisis tomamos el procesador más rápido de los tres; es decir , AMD Ryzen 7 4800H con Radeon Graphics 2.90 GHz.

4.1. Algoritmo de Euclides

4.1.1. Tiempo de Ejecución

En la siguiente tabla y gráfico mostrará el tiempo de compilación en (s) dependiendo de los bits implementados.

Algoritmos /Bits	Euclides Clásico	Bishop GCD	Euclides Menor Resto	GCD Binario	GCD Lehmer
128	0,384	5,349	0,250	0,300	0,129
256	1,257	9,317	0,000	1,110	0,489
512	5,080	79,031	0,000	3,870	1,796
1024	18,732	242,531	0,000	16,250	6,452
2048	81,118	955,700	0,000	65,420	27,226



Cómo se logra ver en la gráfica el Bishop GCD es el algoritmo que destaca por su larga duración al compilar dependiendo de los bits llegando hasta los 955 s . También resalta los algoritmos: GCD Binario y GCD Lehmer por su menor tiempo de compilación. Tener en cuenta que el Euclides de menor resto no registra valores desde los 256 bits debido a su larga compilación interminable .

4.1.2. Número de Loops

En la siguiente tabla y gráfico la cantidad de loops o ciclos que se realizaron en los algoritmos dependiendo de los bits implementados.

Bits	Euclides Clasico	Bishop GCD	Euclides Menor Resto	GCD Binario	GCD Lehmer
128	79	965	50	81	38
256	140	1362	0	180	74
512	309	4814	0	356	157
1024	600	6898	0	753	292
2048	1181	15383	0	1458	594

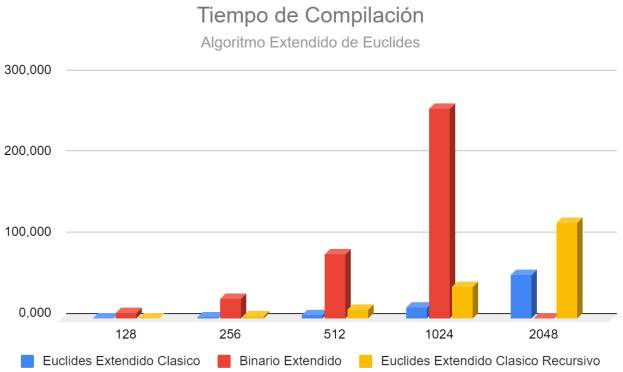
Cómo se logra ver en la gráfica , el Bishop GCD nuevamente resalta por su alta cantidad de loops realizados en todos los bits vistos. También resaltan nuevamente el GCD Binario y GCD Lehmer como los que realizan menos loops. Cabe destacar que el algoritmo Euclides Menor Resto no soporto desde los 256 bits.

4.2. Algoritmo Extendido de Euclides

4.2.1. Tiempo de Ejecución

En la siguiente tabla y gráfico mostrará el tiempo de compilación en (s) dependiendo de los bits implementados.

Algoritmos/ Bits	Euclides Extendido Clásico	Binario Extendido	Euclides Extendido Clásico Recursivo
128	0,278	6,660	0,775
256	0,818	23,681	2,580
512	3,468	79,889	10,469
1024	13,187	258,723	38,531
2048	53,548	0,000	118,001

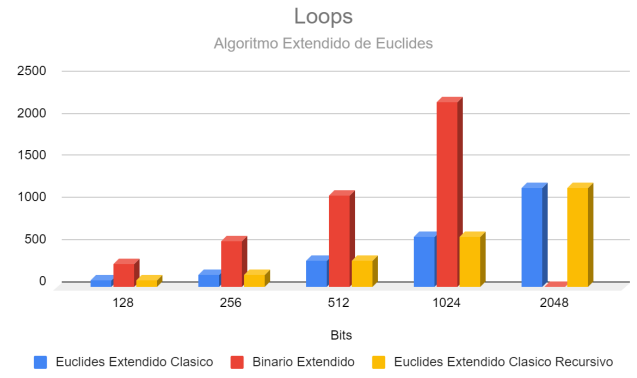


Cómo se logra ver en la gráfica , el Binario extendido resalta sobre los demás por su larga duración de compilación , a diferencia de Euclides Extendido Clásico el cual destaca por su menor tiempo de compilación. Tener en cuenta que binario no se pudo soportar desde los 2048 bits.

4.2.2. Número de Loops

En la siguiente tabla y gráfico la cantidad de loops o ciclos que se realizaron en los algoritmos dependiendo de los bits implementados.

Algoritmos/ Bits	Euclides Extendido Clásico	Binario Extendido	Euclides Extendido Clásico Recursivo
128	79	277	79
256	140	545	140
512	309	1079	309
1024	600	2194	600
2048	1181	0	1181



Cómo se logra ver en la gráfica , el Binario extendido nuevamente resalta por su alta cantidad de loops realizados . También resalta nuevamente el Euclides Extendido Clásico que empata con el euclides recursivo como los que realizan menos loops. Cabe destacar que el algoritmo Binario Extendido no soporto desde los 2046 bits.

5. Conclusiones:

Para finalizar este análisis, se puede observar el comportamiento de diferentes algoritmos para el mismo resultado u objetivo, donde comparamos su eficacia. En primer lugar, se tiene 5 algoritmos referidos al Algoritmo de Euclides; después de evaluar tanto su tiempo de ejecución y cantidad de bucles, afirmamos en la alta eficacia del Algoritmo Binario GCD para pequeños números y el Algoritmo de Lehmer por su alta eficacia en numeros grandes.
Por otro lado, en cuanto a los 3 algoritmos extendidos evaluados; se concluye que el Algoritmo extendido de Euclides clásico es mucho más eficiente. por su poca duración de ejecución.

6. Referencias bibliográficas

[1] El algoritmo de Euclides con residuos de menor valor absoluto. https://miscelaneamatematica.org/download/tbl_articulos.pdf2.82b18c3ae44d6c24.363130312e706466.pdf

[2] Chapter 10. Number theory and Cryptography. https://silo.tips/download/chapter-numbertheory-and-cryptography-contents

[3] Handbook of Applied Cryptography, Menezes, Oorschot, Vanstone. CRC Press, New York, fifth edition (2001). http://www.cacr.math.uwaterloo.ca/hac/

[4] A. Stepanov, P. McJones (2009). Elements of Programming. Addison-Wesley

[5] Handbook of Applied Cryptography, Menezes, Oorschot, Vanstone. CRC Press, New York, fifth edition (2001). http://cacr.uwaterloo.ca/hac/about/chap14.pdf

[6]https://foro.rinconmatematico.com/index.php?topic=7039.0

[7] Introducción a la Teoría de Números. Ejemplos y algoritmos. Walter Mora https://repositoriotec.tec.ac.cr/bitstream/handle/2238/6299/introducci%C3%B3nteor%C3%ADa-n%C3%BAmeros.pdf?sequence=1&isAllowed=y

[8] Extending Stein's GCD algorithm

[9]https://reader.elsevier.com/reader/sd/pii/S0747717185710097?token=F7D806E1C99AF94B8D740105A575F997682CE62D315F13BE3EA81179E1DBD4613AE2847084550A99B0EDD8D56CE41672&originRegion=us-east-1&originCreation=20210604022156