CIENCIA DE LA COMPUTACIÓN

TRABAJO DE INVESTIGACIÓN E IMPLEMENTACIÓN DEL ALGORITMO DE EUCLIDES Y ALGORITMO EXTENDIDO DE EUCLIDES

ÁLGEBRA ABSTRACTA

DANIELA CHAVEZ AGUILAR

GIULIA NAVAL FERNANDEZ

PAOLO DELGADO VIDAL

RODRIGO TORRES SOTOMAYOR

PABLO CARAZAS BARRIOS

AÑO 2021

RESUMEN

En el presente trabajo se muestra un análisis y comparación de diversas versiones de los algoritmos de Euclides y Euclides extendido para hallar el mcd de 2 enteros positivos, junto con sus coeficientes de Bézout, en el caso del algoritmo extendido. Entre los algoritmos que fueron examinados, se encuentran:

- Algoritmo de Euclides clásico
- Algoritmo de Euclides binario
- Algoritmo de Euclides con el menor resto
- Algoritmo de Lehmer
- Algoritmo de Euclides extendido clásico (versión iterativa)
- Algoritmo de Euclides extendido clásico (versión recursiva)
- Algoritmo de Euclides extendido binario

Para hacer la comparación de estos algoritmos se utilizó como criterio el tiempo de ejecución, el número de bucles efectuados y el número de variables utilizadas con entradas con diferente número de bits. En dichas pruebas se observó que el algoritmo con menor tiempo de convergencia fue el algoritmo binario para algoritmos de Euclides, de igual manera, la versión binaria se destacó dentro del conjunto de algoritmos extendidos,cuando las entradas eran valores de menos de 1024 bits.

1. INTRODUCCIÓN

En el presente trabajo se resolverá la siguiente **pregunta:** ¿Cuál es el algoritmo de Euclides y Euclides Extendido más eficiente?
Por ende, los **objetivos** serán: analizar y comparar diferentes algoritmos euclidianos, dividiéndolos en 2 categorías: algoritmos de Euclides y algoritmos de Euclides extendido.

2. CONTENIDO TEÓRICO

ALGORITMOS DE EUCLIDES

a. Algoritmo de Euclides

Definición:
Halla el máximo común divisor de dos números a y b de forma iterativa.

Base Matemática:
El algoritmo de euclides emplea como base matemática al teorema de la división, el cual aplicará de forma consecutiva hasta reducir el residuo a 0. El teorema tiene la siguiente forma:

$$a = q \times b + r$$

Donde a será nuestro número mayor entre ambos operandos, q será el resultado entero de la operación a/by $r$será el residuo restante de esta división. Para proceder con el algoritmo, se realizará sustituciones entre a y b continuamente hasta llegar a un residuo igual a 0, viéndose de la siguiente forma:

$$a = q \times b + r \qquad a_0 = b \wedge b_0 = r \qquad a_0 = q \times b_0 + r$$

Seguimiento:
Mcd(240,134) = 2

a	q	b	r
240	1	134	106
134	1	106	28
106	3	28	22
28	1	22	6
22	3	6	4
6	1	4	2
4	2	2	0

Tiempo de ejecución: 0.052 s
Mcd(48314,35128) = 2

a	q	b	r
48314	1	35128	13186
35128	2	13186	8756
13186	1	8756	4430
8756	1	4430	4326
4430	1	4326	104
4326	41	104	62
104	1	62	42
62	1	42	20
42	2	20	2
20	10	2	0

Tiempo de ejecución: 0.050 s
Cantidad variables (type ZZ): 6

Pseudo-Algoritmo:

- Entrada: a y b (números positivos)
- Salida: a (el máximo común divisor)
- Si (b!=0) entonces
- r← a mod b
- a←b
- b←r

Código C++:

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

using namespace std;
using namespace NTL;

ZZ Euclides(ZZ a, ZZ b){
  while(b>0){
  ZZ r=divi(a,b);
  a=b;
  b=r;
  }
  return a;
}
```

```
int main()
{
   ZZ n_a,n_b;
   cout<<"Ingresa a: "; cin>>n_a;
   cout<<"Ingresa b: ";
cin>>n_b;cout<<endl;
   cout<<"Euclides mcd ("<<n_a<<",
"<<n_b<<") =
"<<Euclides(n_a,n_b)<<endl;
   return 0;
}
```

b. Algoritmo de Euclides Binario

Definición:

Versión del algoritmo de Euclides, que halla de manera iterativa el máximo común divisor de 2 números, junto con los valores a y b que satisfacen *d=ax+by*; en el cual se incrementa la eficiencia al usar un desplazamiento a la derecha de la representación binaria del número.

Base Matemática:

Teorema: si $(a, b) = d$, entonces existe un entero x y y que cumple con: $ax + by = d$ donde $x, y \in Z$.

Para ello empleamos las siguientes reglas, representado la división entre 2 como un right shift binario:

- Si a y b son pares: $2mcd(\frac{a}{2}, \frac{b}{2})$
- Si a es par y b es impar: $mcd(\frac{a}{2}, b)$
- Si a y b son impares: $mcd(\frac{|a-b|}{2}, b)$ **=** $mcd(\frac{|a-b|}{2}, a)$

Pseudo-Algoritmo:

Entrada: 2 enteros positivos *x* y *y*

Salida: *mcd(x, y)*

1. *g ← 1*

2. ***Mientras*** *x* y *y* sean pares: $x \leftarrow x/2,\ \ y \leftarrow y/2,\ \ g \leftarrow 2g$

3. ***Mientras*** *x≠0*:

a. ***Mientras*** *x* sea par: $x \leftarrow x/2$

b. ***Mientras*** *y* sea par: $y \leftarrow y/2$

c. $t \leftarrow |x - y|/2$

d. ***Si*** $x \geq y$***entonces*** $x \leftarrow t$ *;* ***si no,*** $y \leftarrow t$

4. ***return*** *g*y*

Código C++:

```
ZZ BinaryGCD(ZZ x, ZZ y){
  ZZ g=conv<ZZ>(1);
  while(par(x) && par(y)){
```

```
  x>>=1;y>>=1;g<<=1;
 }
 while(x!=0){
  while(par(x)){
  x>>=1;
  }
  while(par(y)){
  y>>=1;
  }
  ZZ t=abs(x-y); t>>=1;
  if (x>=y){
  x=t;
  }
  else{
  y=t;
  }
 }
 return g*y;
}
```

Seguimiento:

Input: a=693, b= 609

Tiempo de ejecución=0.089 s

Cantidad variables (type ZZ): 4

x	y	g
693	609	1
42	609	1
21	609	1
21	294	1
21	147	1
21	63	1
21	21	1
0	21	1

c. Algoritmo de Euclides con menor resto

Definición:

Versión del Algoritmo de Euclides clásico, donde buscaremos emplear el menor resto posible dentro de nuestro algoritmo, con la finalidad de reducir lo máximo posible la cantidad de divisiones.

Base Matemática:

El algoritmo de Euclides clásico emplea un resto que va de 0 a r_i, mientras que el algoritmo de Euclides con menor resto busca reducir este resto de 0 a $r_{i-1}/2$. Para ello, suponiendo un mcd(a,b), donde a>=0 y b>0:

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

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

Donde buscaremos representar r como el mínimo posible, es decir, $r = min\{r_1, r_2\}$,o representado mediante reemplazo, $r = min\{a - b \times ([a/b] + 1), a - b \times [a/b]\}$. Esto nos da como resultado que $r_{min} = a - b \times ([a/b] + 1/2)$.El cual reemplazamos finalmente en el teorema de la división:

$$a = b \times q - (a - b \times ([a/b] + 1/2))$$

Pseudo-Algoritmo:

Entrada: Dos enteros a y b

Salida: mcd(a,b)

Inicializar como ZZ las variables “d”, “c” y “r”

Si a es igual a 0:

- c ← 0

De lo contrario:

- c ← a
- d← b

Mientras el valor de “d” sea diferente a 0:

- r ← “c - d*(c/d+1/2)”
- c← d
- d← r

Si “c” es menor que 0:

- return(-c)

De lo contrario:

- return(c)

Fin de la función

Seguimiento:

Entrada: 48647 y 18519

Salida: mcd(48647,18519) = 1

c	d	q	r
48647	18519	2	11609
18519	11609	1	6910
11609	6910	1	4699
6910	4699	1	2211
4699	2211	2	277
2211	277	7	272
277	272	1	5
272	5	54	2
5	2	2	1
2	1	2	0

Tiempo de ejecución: 1.978 s

Cantidad variables (type ZZ): 5

Código C++:

```
#include<iostream>
#include<string>
#include<time.h>
#include <NTL/ZZ.h>
#include<cstdlib>
using namespace std;
using namespace NTL;
ZZ euclides_menor_resto(ZZ a, ZZ b){
    ZZ d,c,r;
    if(a==0){
```

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

d. Algoritmo de Lehmer

Definición: Versión más veloz del algoritmo de Euclides. Más efectiva cuando los 2 números ingresados son del mismo tamaño.

Base Matemática:

El algoritmo de Lehmer es una variante del algoritmo de Euclides, donde buscaremos reemplazar las divisiones de múltiple precisión por operaciones simples y reducidas. Para ello, extraemos los dígitos de mayor orden de ambos números según una base dada, almacenados en $\sim a$ y $\sim b$ de la forma:

$$base = 10^3 \qquad a = 4562131 \ \wedge \ b = 6723125 \ \rightarrow \ \sim a = 456 \ \wedge \ \sim b = 672$$

Ya con estos dígitos de mayor orden, procedemos a realizar operaciones con 4 variables simples $A = 1,\ B = 0,\ C = 1\ y\ D = 0$, y empleando distintas sustituciones dependiendo de ciertas condiciones:

$$Si\ \sim b + C \neq 0 \wedge \sim b + D \neq 0 \rightarrow q = [(\sim a + A)/(\sim b + C)] \wedge q' = [(\sim a + B)/(\sim b + D)]$$
$$Si\ q = q' \rightarrow t = A - q(C) \wedge A = C \wedge C = t \wedge t = B - q(D) \wedge B = D \wedge D = t$$
$$\wedge\ t = \sim a + q(\sim b) \wedge \sim a = \sim b \wedge \sim b = t$$

$$Si\ B = 0 \rightarrow T = a|b\ \wedge a = b \wedge b = T \qquad sino\ T = A(a) + B(b) \wedge u = C(a) + D(b) \wedge a = T \wedge b =$$
$$v\ =\ mcd(a, b)$$

Pseudo-Algoritmo:

Entrada: 2 enteros positivos *a* y *b*

Salida: *mcd(x, y)*

1. ***Mientras*** $y \leq b$:

a. $\tilde{x} \leftarrow hod(x);\ \tilde{y} \leftarrow hod(y);$

b. $A \leftarrow 1,\ B \leftarrow 0,\ C \leftarrow 0,\ D \leftarrow 1$

c. ***Mientras*** $(\tilde{y} + C) \neq 0\ y\ (\tilde{y} + D) \neq 0$:

i. $q \leftarrow \lfloor(\tilde{x} + A)/(\tilde{y} + C)\rfloor,\ q' \leftarrow \lfloor(\tilde{x} + B)/(\tilde{y} + D)\rfloor$

ii. ***Si*** $q \neq q'$ ***entonces*** *ir al paso d*

iii. $t \leftarrow A - qC,\ A \leftarrow C,\ C \leftarrow t,\ t \leftarrow B - qD,\ B \leftarrow D,\ D \leftarrow t$

iv. $t \leftarrow \tilde{x} - q\tilde{y},\ \tilde{x} - \tilde{y},\ \tilde{y} \leftarrow t$

d. ***Si*** $B = 0$ ***entonces*** $t \leftarrow x\ mod\ y,\ x \leftarrow y,\ y \leftarrow T,$ ***si no***,
$T \leftarrow Ax + By,\ u \leftarrow Cx + Dy,\ x \leftarrow T,\ y \leftarrow u$

2. $v = gcd(x, y)$

3. ***return*** v

Código C++:

```
}ZZ LehmerGCD(ZZ x, ZZ y){

  int b=10;

  while (y>=b){

    ZZ xn=Str2Num(Num2Str(x)[0]);

    ZZ yn=Str2Num(Completar0s(y,
Num2Str(x).size())[0]);

    ZZ
A=conv<ZZ>(1),B=conv<ZZ>(0),C=co
nv<ZZ>(0),D=conv<ZZ>(1),t,T,u;

    while (yn+C!=0 && yn+D!=0){
```

```
    ZZ q=(xn+A)/(yn+C);
    ZZ qp=(xn+B)/(yn+D);
    if (q==qp){
      t=A-q*C; A=C; C=t; t=B-q*D;
B=D; D=t;
      t=xn-q*yn; xn=yn; yn=t;
    }
    else{
      break;
     }
    }
    if (B==0){
          T=divi(x,y);
          x=y; y=T;
     }
    else{
          T=A*x+B*y; u=C*x+D*y;
          x=T; y=u;
     }
 }
 ZZ v=Euclides(x,y);
 return v;
```

A	B	C	D	x	y
1	0	0	1	412	303
0	1	1	-1	412	303
0	1	1	-1	303	109
1	0	0	1	303	109
1	0	0	1	109	85
1	0	0	1	85	24
1	0	0	1	24	13
1	0	0	1	13	11
1	0	0	1	11	2

Seguimiento:

Input: a=412, b= 303

ALGORITMOS DE EUCLIDES EXTENDIDOS

a. Algoritmo Extendido de Euclides

Definición:
Se halla el máximo común divisor de “a” y “b” (mcd(a,b)=d), junto con los valores de “x” y “y” que satisfacen la ecuación ax+by=d

Base Matemática:

El algoritmo extendido de euclides emplea como base matemática la inversa multiplicativa ya que junto a la obtención del $mcd(a\ y\ b) = d$ encuentra la solución a:

$$ax + by = d$$

En el cual, si $(a, b) = d$, entonces existe un entero x y y que cumple con: $ax + by = d$ donde $x, y \in Z$. Hallando dicha solución como una combinación lineal de $a\ y\ b$en la cual, si $a, b\ y\ c$son enteros positivos tal que $mcd(a\ y\ b) = 1$ y $a\ |bc$ entonces $a|c$.

Seguimiento:

q	r	x	y	a	b	x2	x1	y2	y1
				240	134	1	0	0	1
1	106	1	-1	134	106	0	1	1	-1
1	28	-1	2	106	28	1	-1	-1	2
3	22	4	-7	28	22	-1	4	2	-7
1	6	-5	9	22	6	4	-5	-7	9
3	4	19	-34	6	4	-5	19	9	-34
1	2	-24	43	4	2	19	-24	-34	43
2	0	67	-120	2	0	-24	67	43	-120

ax + by = d => 240(-24) + 134(43) = 2
Tiempo ejecución: 0.031 s

q	r	x	y	a	b	x2	x1	y2	y1
				48314	35128	1	0	0	1
1	13186	1	-1	35128	13186	0	1	1	-1
2	8756	-2	3	13186	8756	1	-2	-1	3
1	4430	3	-4	8756	4430	-2	3	3	-4
1	4326	-5	7	4430	4326	3	-5	-4	7
1	104	8	-11	4326	104	-5	8	7	-11
41	62	-333	458	104	62	8	-333	-11	458
1	42	341	-469	62	42	-333	341	458	-469
1	20	-674	927	42	20	341	-674	-469	927
2	2	1689	-2323	20	2	-674	1689	927	-2323
10	0	-17564	24157	2	0	1689	-17564	-2323	24157
	mcd(5638,3517)	2							

ax + by = d => 48314(1689) + 35128(-2323) = 2

Tiempo ejecución: 0.334 s
Cantidad variables (type ZZ): 14

Pseudo-Algoritmo:

- Entrada: a y b(números enteros positivos)
- Salida: d, x, y
- Si b es igual a 0 (b=0) colocar d←a, x←1, y←0, y retorna los valores(d,x,y)
- X2 ←1, x1←0, y2←0, y1←1

- Mientras b>0:
- q←[a/b], r←a-qb, x←x2-qx1, y ←y2-qy1
- a←b, b←r, x2←x1, x1←x, y2←y1, y1←y

Código C++

```
vector<ZZ> Euclides_ext(ZZ a, ZZ b){
  ZZ x0=conv<ZZ>(1),y0=conv<ZZ>(0),x=conv<ZZ>(0),y=conv<ZZ>(1);
  while(b>0){
  ZZ aux=x;
  ZZ q=a/b;
  ZZ r=divi(a,b);
  x=x0-q*x;
  x0=aux;
  aux=y;
  y=y0-q*y;
  y0=aux;
  a=b;
  b=r;
  }
  ZZ result[]={a,x0,y0};
  vector<ZZ>p(result,result+3);
  return p;
}
```

b. Algoritmo Extendido de Euclides Recursivo

Definición: Versión del algoritmo extendido de Euclides, que halla de manera recursiva el máximo común divisor de 2 números, junto con los valores a y b que satisfacen *d=ax+by*.

Base Matemática:

Como, en su versión iterativa, el algoritmo extendido de euclides emplea como base matemática la inversa multiplicativa ya que junto a la obtención del $mcd(a\ y\ b) = d$ encuentra la solución a:

$$ax + by = d$$

En el cual, si $(a, b) = d$, entonces existe un entero x y y que cumple con: $ax + by = d$ donde $x, y \in Z$. Hallando dicha solución como una combinación lineal de $a\ y\ b$en la cual, si $a, b\ y\ c$son enteros positivos tal que $mcd(a\ y\ b) = 1$ y $a\ |bc$ entonces $a|c$.

Pseudo-Algoritmo:

- ***Entrada:*** Enteros positivos a y b
- ***Salida:*** Arreglo de enteros *{d,i,j}* que satisfacen *d = gcd(a,b) = ia+jb*
- ***si*** *b*=0 ***entonces***
- ***return*** *{a,1,0}*
- *q ← a mod b*
- r es el entero que satisface *a = rb + q*
- *{d,k,l}* ← AlgoritmoExtendido*(b,q)*
- ***return*** *{d,l,k-lr}*

Código C++:

```
vector<ZZ> ExtendedEuclidGCD(ZZ a, ZZ b){
 if (b==0){
vector<ZZ>p{a,conv<ZZ>(1),conv<ZZ>(0)};
return p;
 }
 ZZ q=divi(a,b);
 ZZ r=(a-q)/b;
 vector<ZZ>p=ExtendedEuclidGCD(b, q);
 ZZ k=p[1], l=p[2];
 p[1]=l;
 p[2]=k-l*r;
 return p;
}
```

Seguimiento:

Input: a=393, b= 267

Tiempo de ejecución=0.077 s

Cantidad variables (type ZZ): 7

a	b	r	I (p[1])	J (p[2])
393	267	1	-36	53
267	126	2	17	-36
126	15	8	-2	17
15	6	2	1	-2
6	3	2	0	1
3	0		1	0

c. Algoritmo Extendido de Euclides Binario

Definición:

Versión del algoritmo extendido de Euclides, que halla de manera iterativa el máximo común divisor de 2 números, junto con los valores a y b que satisfacen *d=ax+by*; en el cual se incrementa la eficiencia al usar un desplazamiento a la derecha de la representación binaria del número.

Base Matemática:

El algoritmo extendido de euclides emplea como base matemática la inversa multiplicativa ya que junto a la obtención del mcd(a y b)=d encuentra la solución a:

ax+by=d

En el cual, si (a,b)=d, entonces existe un entero x y y que cumple con: ax+by=d donde x,yZ. Hallando dicha solución como una combinación lineal de a y b en la cual, si a,b y c son enteros positivos tal que mcd(a y b)=1 y a |bc entonces a|c.

Para ello reduce el número de operaciones de precisión múltiple representando la división entre 2 como un desplazamiento a la derecha y las divisiones como restas, aunque esta reducción suponga más iteraciones.

Pseudo-Algoritmo:

Entrada: 2 enteros positivos *x* y *y*

Salida: Enteros a y b que satisfacen *v = ax+by*, donde *v=mcd(x, y)*

1. $g \leftarrow 1$
2. ***Mientras*** *x* y *y* sean pares: $x \leftarrow x/2,\ y \leftarrow y/2,\ \ g \leftarrow 2g,$
3. $u \leftarrow x,\ v \leftarrow y,\ A \leftarrow 1,\ B \leftarrow 0,\ C \leftarrow 0,\ D \leftarrow 1$
4. ***Mientras*** *u* sea par:

a. $u \leftarrow u/2$

b. ***Si*** $A \equiv B \equiv 0 (mod\ 2)$ ***entonces*** $A \leftarrow A/2, B \leftarrow B/2$; ***si no,*** $A \leftarrow (A + y)/2, B \leftarrow (B - x)/2$

5. ***Mientras*** *v* sea par:

a. $v \leftarrow v/2$

b. ***Si*** $C \equiv D \equiv 0 (mod\ 2)$ ***entonces*** $C \leftarrow C/2, D \leftarrow D/2$; ***si no,*** $C \leftarrow (C + y)/2, D \leftarrow (D - x)/2$

6. ***Si*** $u \geq v$ ***entonces*** $u \leftarrow u - v, A \leftarrow A - C, B \leftarrow B - D$; ***si no,*** $v \leftarrow v - u, C \leftarrow C - A, D \leftarrow D - B$

7. ***Si*** $u = 0$ ***entonces*** $a \leftarrow C, b \leftarrow D$, ***return*** *(a, b, g*v) ;* ***si no,*** *volver al paso 4*

Código C++:

```
vector<ZZ> ExtendedBinaryGCD(ZZ x, ZZ y){
 ZZ g=conv<ZZ>(1);
 while(par(x) && par(y)){
x>>=1;y>>=1;g<<=1;
 }
 ZZ u=x,v=y,A=conv<ZZ>(1),B=conv<ZZ>(0),C=conv<ZZ>(0),D=conv<ZZ>(1);
 while(u!=0){
while(par(u)){
        u>>=1;
        if (par(A) && par(B)){
        A>>=1;B>>=1;
        }
        else{
        A+=y;A>>=1;
        B-=x;B>>=1;
        }
}
while(par(v)){
        v>>=1;
        if (par(C) && par(D)){
        C>>=1;D>>=1;
        }
        else{
        C+=y;C>>=1;
        D-=x;D>>=1;
        }
}
if (u>=v){
```

```
            u-=v;A-=C;B-=D;
    }
    else{
            v-=u;C-=A;D-=B;
    }
     }
     vector<ZZ> result{g*v,C,D};
     return result;
    }
```

Seguimiento:

Input: a=412, b= 303

Tiempo de ejecución: 0.083 s

Cantidad variables (type ZZ): 10

u	v	A	B	C	D
412	303	1	0	0	1
206	303	152	-206	0	1
103	303	76	-103	0	1
103	200	76	-103	-76	104
103	100	76	-103	-38	52
103	50	76	-103	-19	26
103	25	76	-103	142	142
78	25	-66	90	142	-193
39	25	-33	45	142	-193
14	25	-175	238	142	-193
7	25	64	-87	142	-193
7	18	64	-87	78	-106
7	9	64	-87	39	-53
7	2	64	-87	-25	34
7	1	64	-87	139	-189
6	1	-75	102	139	-189
3	1	114	-155	139	-189
2	1	-25	34	139	-189
1	1	139	-189	139	-189
0	1	0	0	139	-189

3. ANÁLISIS DE LOS ALGORITMOS

Para los algoritmos utilizamos: Codeblocks usando la librería NTL para probar con los distintos números de bits.

Sistema Operativo: Windows 10

Resultados ejecución y N° de Bits:

Probaremos con los siguientes números:
De 8 bits: 152 y 214

De 16 bits:46551 y 57688

De 32 bits: 3983912617 y 2555456905

De 64 bits: 14411084638258398830 y 54485576096155850184

De 128 bits: 172027756592610196758328726200628638229 y 254971332396674192098953250218999913763

De 256 bits:
110712138368495899129846235571176642817949412817682077248125557883511188904693 y
74593852437691348211017268482459664548683965409297776974415928312947489229807

De 1024 bits:

-> Número 1:
13016760040412969108405797058553182772027087709542651468056849457098148842587840533438455348805435097530343192654618401705138787087519643598611351569490064365014630419137265402430803585208699683083190769899387818095445192880558845975361399905307764797792402328547571541790032511237039039229781557943778891763 2

-> Número 2:
91860333006720436715282712271443961187684739190574558930174331118210215020152136640988802969024973554176698739910772696024215357822488289497013380701226114548393963056477558 7

De 2048 bits

-> Número 1:

2356419731499597662687589404844765655019061678826845066289807003421034894802429680380527815409851024512900236980236763220278177572155837132973024126200521057591163256464454644454644516435800047685781200551336940836742568862331970340982077110474631587687535650653824796760372859940018343934768520179217791185988580701232282034633113054141336354300145043601374595687477993859996840522400171241798577756589055365431535362144160173117683592030542534686073177456395175270115592758981371 48

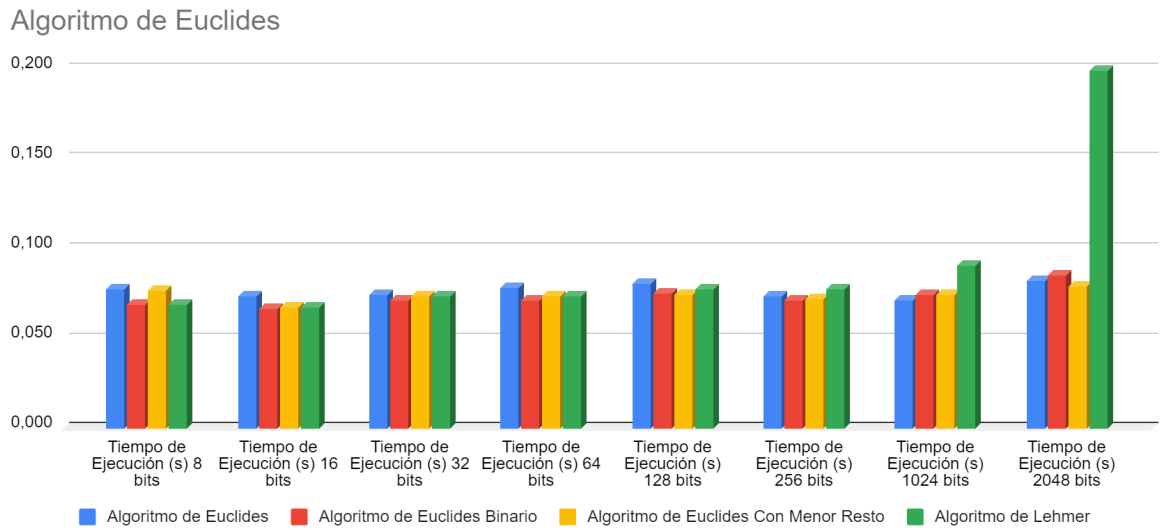
-> Número 2:

2847173816560474913734881490052141685926634970444589951234211628391154852620438067350147348396250578683470764435282022603509066099768094519352194791252154881725904273031445991910447906618620811884254685484879563690812623339219662831776313498549572794896817085543708660511847286259924710139980104306672560700090034759834921825896466454191759072981167896822453093928584716175233549132918281716173636950714804484982579347098458346954797043203166055909026424915213506488226054044259491402147550397686668 2

ALGORITMO DE EUCLIDES:

	Algoritmo de Euclides	Algoritmo de Euclides Binario	Algoritmo de Euclides Con Menor Resto	Algoritmo de Lehmer
Tiempo de Ejecución (s) 8 bits	0,077	0,069	0,076	0,069

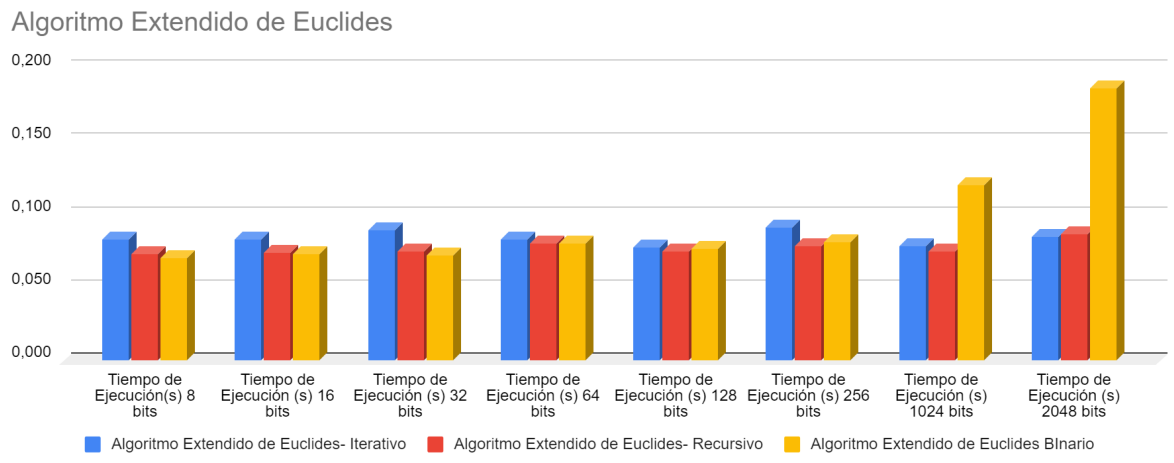
Tiempo de Ejecución (s) 16 bits	0,073	0,066	0,067	0,067
Tiempo de Ejecución (s) 32 bits	0,074	0,071	0,073	0,073
Tiempo de Ejecución (s) 64 bits	0,078	0,071	0,073	0,073
Tiempo de Ejecución (s) 128 bits	0,080	0,075	0,074	0,077
Tiempo de Ejecución (s) 256 bits	0,073	0,071	0,072	0,077
Tiempo de Ejecución (s) 1024 bits	0,071	0,074	0,074	0,090
Tiempo de Ejecución (s) 2048 bits	0,082	0,085	0,079	0,199
Promedio	0,076	0,073	0,074	0,091



ALGORITMO EXTENDIDO DE EUCLIDES:

	Algoritmo Extendido de Euclides- Iterativo	Algoritmo Extendido de Euclides- Recursivo	Algoritmo Extendido de Euclides BInario
Tiempo de	0,083	0,073	0,070

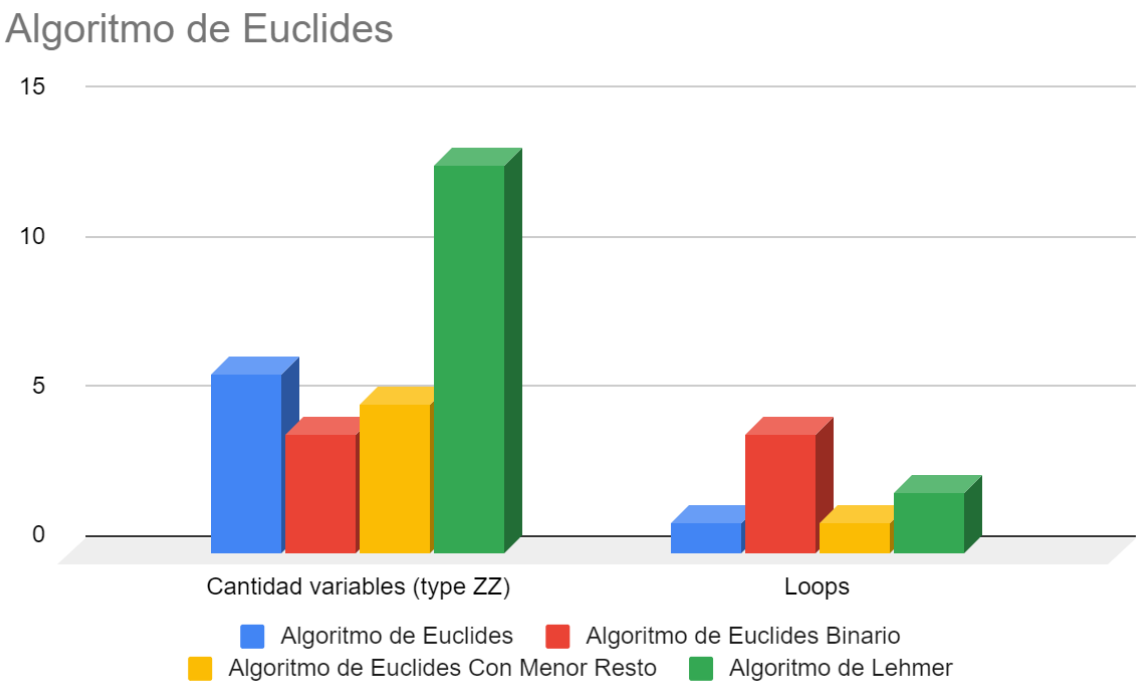
Ejecución(s) 8 bits			
Tiempo de Ejecución (s) 16 bits	0,083	0,074	0,073
Tiempo de Ejecución (s) 32 bits	0,089	0,075	0,072
Tiempo de Ejecución (s) 64 bits	0,083	0,080	0,080
Tiempo de Ejecución (s) 128 bits	0,077	0,075	0,076
Tiempo de Ejecución (s) 256 bits	0,091	0,078	0,081
Tiempo de Ejecución (s) 1024 bits	0,078	0,075	0,120
Tiempo de Ejecución (s) 2048 bits	0,085	0,086	0,186
Promedio	0,084	0,077	0,095



Resultados N° de Bucles

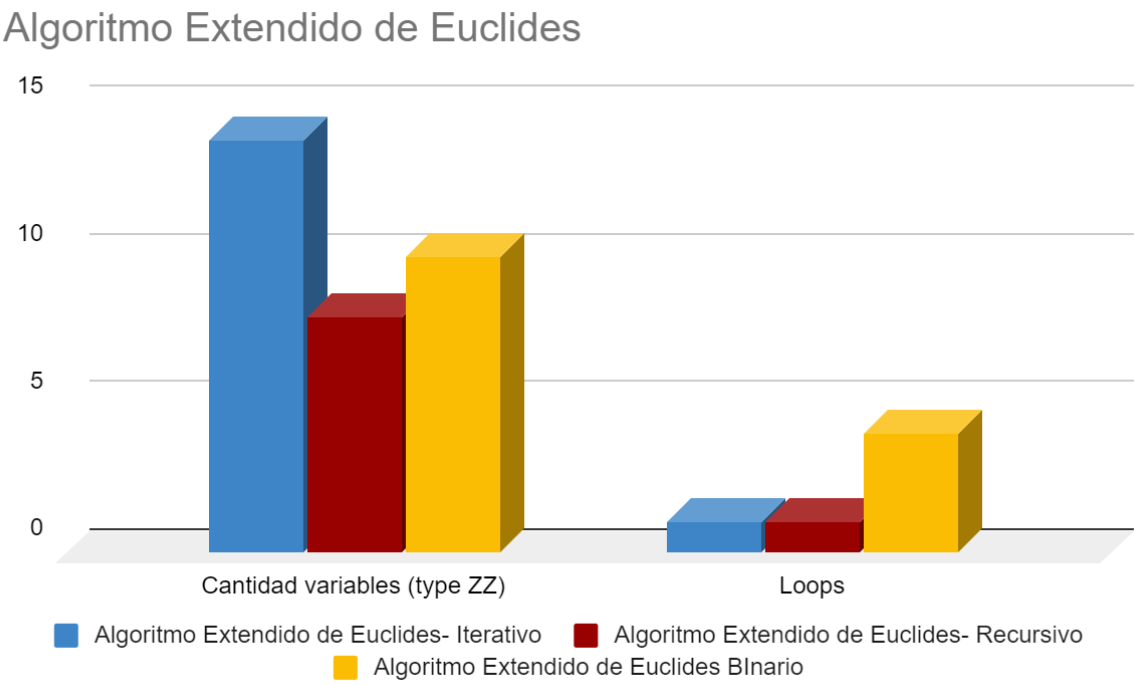
	Algoritmo de Euclides	Algoritmo de Euclides Binario	Algoritmo de Euclides Con Menor Resto	Algoritmo de Lehmer
Cantidad variables (type ZZ)	6	4	5	13

Loops	1	4	1	2



	Algoritmo Extendido de Euclides- Iterativo	Algoritmo Extendido de Euclides- Recursivo	Algoritmo Extendido de Euclides BInario
Cantidad variables (type ZZ)	14	8	10
Loops	1	1	4

Las variables del Algoritmo Recursivo de Euclides es 3 + 5x; al ser una función recursiva se crearán nuevas variables, por ende en la ecuación “x” dependerá de los valores ingresados.



4. CONCLUSIONES GENERALES

En conclusión, cuando realizamos este proyecto de investigación pudimos ver las distintas soluciones y razonamientos que se utiliza en cada algoritmo los cuales resuelven un mismo problema, Máximo Común Divisor y los valores que resuelven la ecuación ax+by=d (visto anteriormente), y con los análisis pudimos concretar la idea de que el algoritmo de Euclides más eficiente es el binario y el Algoritmo Extendido de Euclides más eficiente es el binario .

5. REFERENCIAS

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

[02] A computational introduction to Number Theory and Algebra. Victor Shoup. http://www.shoup.net/ntb/ntb-v2.pdf

[03] Numerical Recipes in C : The Art of Scientific Computing. William H. Press, Brian P. Flannery, Saul A. Teukolsky, William T. Vetterling

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

[05] A comparison of several greatest common divisor (GDC) Algorithms. https://www.ijcaonline.org/volume26/number5/pxc3874253.pdf

[06] 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

6. COMENTARIOS

Enlace GitHub: https://github.com/RodATS/Rodrigo_Torres_Sotomayor.git