CIENCIA DE LA COMPUTACIÓN

ÁLGEBRA ABSTRACTA

TRABAJO DE INVESTIGACIÓN

- **Apaza Coaquira, Aarón Misash**
- **Choque Mayta, Gabriel Santiago**
- **Condori Gonzales, Jean Carlo**
- **Romero Guillen, Geraldo Mark**
- **Vilca Campana, Jose Enrique**

CCOMP3-1

2021

Los alumnos declaran haber realizado el trabajo de acuerdo a las normas De la universidad Católica San Pablo

Trabajo de Investigación

Resumen:

Se implementaron las siguientes soluciones con sus respectivas descripciones:

- Algoritmo de Euclides clásico: El más antiguo que analizaremos y por mucho tiempo el más eficiente que ha habido.
- Algoritmo de Euclides con menor resto: A diferencia de su predecesor, el módulo devuelve el valor absoluto del menor residuo.
- Algoritmo de Lehmer del mcd: Una mejora del simple pero lento algoritmo de Euclides para números muy grandes
- Algoritmo binario del mcd: Divisiones sucesivas entre dos. Operación barata para una computadora.

Criterios de Evaluación:

- Tiempo de ejecución respecto al número de bits.
- Número de loops o bucles

Algoritmo de mejor desempeño:

- Algoritmo binario del mcd.

Introducción:

El presente trabajo de investigación está motivado por la necesidad de eficiencia y reducción de costo computacional en los algoritmos de máximo común divisor. Para esto recurrimos a varios algoritmos que llegan a la misma solución, pero de distintas formas, de esta forma tendremos un enfoque variado para solucionar esta necesidad.

En el trabajo presentamos 4 algoritmos: Euclides clásico, Euclides con menor resto, Lehmer del mcd, Binario del mcd. A lo largo de este trabajo se verá desarrollado cada uno de estos algoritmos de manera más profunda, comparando cada algoritmo para llegar al más eficiente haciendo comparaciones y sustentandos matemáticamente.

Contenido Teórico

Algoritmo de euclides clásico

Definición

El algoritmo clásico de euclides haya el máximo común divisor de dos números enteros por medio de divisiones euclidianas sucesivas, encadenando el menor de los dos anteriores números junto al residuo positivo de dividir estos dos, hasta que el residuo de los dos números sea 0, entonces el máximo común divisor es el penúltimo resultado de las divisiones euclidianas.

Sustento matemático

La ecuación: A = B*q + R, donde q es coeficiente de A y B.

Si un entero D divide B y R, entonces también divide a A.

Si B = b*D y R = r*D.

La ecuación sería:

A = b*D*q + r*D
A = b*D*q + r*D
A = D(b*q + r), entonces, existe a = A/D.
a*D = D(b*q+r)

Asimismo, si un entero D divide A y B, entonces divide también a R.

De esta observación: mcd(A,B) = mcd(B,R), donde R es el resultado de A mod B

Asumiendo dos números a, b y que a >= b >= 0.
si b = 0, mcd(a,0) = a. sino, b > 0

Ref. [02] Capítulo 4: Euclid´s algorithms. Página 74.
Ref. [04] Euclid´s Algorithm. Página 5

Pseudo-algoritmo

INPUT: dos números enteros a y b siendo a ≥ b.
OUTPUT: máximo común divisor de a y b.
1. WHILE b <> 0 DO:
    1.1 Set r ← (a mod b) ,
    1.2 Set a ← b,
    1.3 Set b←r.
2. Return a.

Ref. [01]. Capítulo 2 : Mathematical Background. Sección: Algorithms in Z. Página 20.

Seguimiento numérico

144 = 89 · 1 + 55 ⇒ mcd(144, 89) = mcd(89,55)
89 = 55 · 1 + 34 = mcd(55,34)
55 = 34 · 1 + 21 = mcd(34, 21)
34 = 21 · 1 + 13 = mcd(21, 13)
21 = 13 · 1 + 8 = mcd(13, 8)
13 = 8 · 1 + 5 = mcd(8, 5)
8 = 5 · 1 + 3 = mcd(5, 3)
5 = 3 · 1 + 2 = mcd(3, 2)
3 = 2 · 1 + 1 = mcd(2, 1)
2 = 1 · 2 + 0 = mcd(1, 0) = 1

Implementación en C++

```
void classic_euclidean(ZZ a, ZZ b)
{
    ZZ c;
    while (b != 0)
    {
        c = a;
        a = b;
        b = myHugeModule(c, b);
    }
    cout << "classic euclidean\t: " << a << endl;
}
```

MCD resto mínimo

Definición

El algoritmo con menor resto es una variante para mejorar un poco el desempeño del algoritmo de Euclides. Kronecker [5] demostró 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.

Utilizamos una función parte entera inferior denotada $Floor(x)$, devuelve el más grande entero menor o igual a x, es decir:

$$[[x]] \ = \ Máx\{n \in Z \mid n \leq x\}.$$

Por ejemplo, $Floor(2.8) \ = \ 2 \equiv [[2.8]] \ = \ 2$

Escogemos en cada paso el menor resto:

$$r \ = \ Mín(r1\,, r2) \ = \ Mín(\,|a \ - \ b \cdot [[a/b]]|,\ |a \ - \ b \cdot [[a/b \ + \ 1]]|\,)\,.$$

De esta manera:

$$r \leq b/2$$

.y esto se puede reducir a una sola operación:

$$r \ = \ Mín(r1, r2) \ = |\,a \ - \ b \cdot [[a/b \ + \ ½]]|.$$

Implementación:

La implementación es la misma que la del algoritmo de Euclides clásico, solo reemplazamos la operación de mínimo residuo ya puesta anteriormente:

```
Function mcdMenorResto(a,b) As Long
Dim c As Long, d As Long, r As Long
If a=0 Then
      c = b
Else
      c=a
      d=b
      While d<> 0
            r = c-d*Int(c/d+1/2)
            c = d
            d = r
      WhileEnd
End If
mcdMenorResto = Abs(c)
End Function
```

Seguimiento paso a paso:

$mcd(144,\ 89)$

$$r\ =\ Min(r1, r2)\ =\ |a - b\ \cdot\ [[a/b\ +\ ½]]|.$$

$r = |\ 144 - 89\ \cdot\ [[144/89\ +\ ½]]|\ = |\ 144 - 89\ \cdot\ [[1.6\ +\ 0.5]]| = |\ 144 - 89\ \cdot\ 2|\ = |\ 144 - 168|\ \rightarrow 34$
$144\ =\ 89 \cdot 2\ -\ 34\ \rightarrow\ mcd(89, 34)$

$r = |89 - 34\ \cdot\ [[89/34\ +\ ½]]|\ \ \ = |89 - 34\ \cdot\ [[2.6\ +\ 0.5]]|\ \ = |89 - 34\ \cdot 3|\ \ = |89 - 102|\ \ \rightarrow 13$
$89\ \ =\ 34 \cdot 3\ -\ 13\ \rightarrow\ mcd(34, 13)$

$r = |34 - 13\ \cdot\ [[34/13\ +\ ½]]|\ \ \ = |34 - 13 \cdot [[2.6\ +\ 0.5]]|\ \ = |34 - 13\ \cdot 3|\ \ = |34 - 39|\ \ \rightarrow 5$
$34\ \ =\ 13 \cdot 3\ -\ 5\ \ \rightarrow\ mcd(13, 5)$

$r = |13 - 5\ \cdot\ [[13/5\ +\ ½]]|\ \ \ = |13 - 5 \cdot [[2.6\ +\ 0.5]]|\ \ = |13 - 5\ \cdot 3|\ \ = |13 - 15|\ \ \rightarrow 2$
$13\ \ =\ 5 \cdot 3\ +\ 2\ \ \ \rightarrow\ mcd(5, 2)$

$r = |5 - 2\ \cdot\ [[5/2\ +\ ½]]|\ \ \ = |5 - 2 \cdot [[2.5\ +\ 0.5]]|\ \ = |5 - 2\ \cdot 3|\ \ = |5 - 6|\ \ \rightarrow 1$
$5\ \ =\ 2 \cdot 3\ +\ 1\ \ \ \rightarrow\ mcd(2, 1)$

$r = |2 - 1\ \cdot\ [[2/1\ +\ ½]]|\ \ \ = |2 - 1 \cdot [[2\ +\ 0.5]]|\ \ = |2 - 1\ \cdot 2|\ \ = |2 - 2|\ \ \rightarrow 0$
$2\ \ =\ 1 \cdot 2\ +\ 0\ \ \ \rightarrow\ mcd(1, 0) = 1$

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

Implementación en C++

```
ZZ Euclides_menor_resto(ZZ &a, ZZ &b)
{
    if (a == 0)
        return b;
    while (module(a, b) != 0)
    {
        ZZ temp = abs(b);
        ZZ r = abs(a - b * (a / b + (module(a, b) >= b / 2)));
        b = r;
        a = temp;
    }
    return b;
}
```

Algoritmo de Lehmer

Definición:

El algoritmo de Lehmer es una mejora del simple pero lento algoritmo de Euclides para números muy grandes , Según [08] la mejora se basa en usar en los primeros pasos (u/10^k) y ($v/10^k$) en vez de $u\ y\ v$. D.Knuth [07] menciona que trabajar con solo los dígitos iniciales de números grandes, es posible hacer la mayoría de los cálculos con precisión simple aritmética, y hacer una reducción sustancial en el número de operaciones de precisión múltiple involucradas. La idea es ahorrar tiempo haciendo un cálculo “virtual” en lugar del real .

Por ejemplo , si tenemos dos números grandes $u' = 27182818$ y $v' = 10000000$, asumimos que trabajamos con una máquina de solo 4 dígitos ; es decir nuestros par de números u y v son enteros de multi precisión y necesitamos trabajar con enteros de precisión simple .

sea $u' = 21718$, $v' = 10001$, $u'' = 2719$, $v'' = 1000$, siendo u'/v' y u''/v'' aproximaciones de u/v .

$$u'/v' < u/v < u''/v'' \equiv 2.1715 < 2.71828 < 2.1719$$

Como Lhemer señaló que los cocientes de cada paso en la división de un algoritmo estándar son pequeños , [18] señala que observó que los cocientes 1,2 y 3 comprenden el 67,7% de todos los cocientes .Estos cocientes pueden hallarse con los dígitos iniciales y calcular la secuencia de cocientes siempre que sea correcta.

Si ejecutamos el algoritmo de Euclides simultáneamente en precisión simple (4 dígitos) de $u\ y\ v$ hasta que obtengamos diferente cociente tenemos :

u'	v'	q'
2718	1001	2
1001	716	1
716	285	2
285	146	1
146	139	1
139	7	19

u'	v'	q'
2719	1000	2
1000	719	1
719	281	2
281	157	1
157	124	1
124	33	3

Como notamos los primeros cinco cocientes son iguales , sin embargo la sexta iteración encontramos que los cocientes son diferentes $q' \neq q''$; por lo tanto los cálculos de

precisión simple se suspenden . Los cálculos con los números de $u\ y\ v$ de precisión múltiple son los siguientes:

$$\begin{array}{lll} u' & v' & q' \\ u_0 & v_0 & 2 \\ v_0 & u_{0-2v} & 2 \\ u_0 - 2v_0 & -u_0 - 3v_0 & 1 \\ -u_0 + 3v_0 & 3u_0 - 8v_0 & 1 \\ 3u_0 - 8v_0 & -4u_0 + 11v_0 & 1 \\ -4u_0 + 11v_0 & 7u_0 - 19v_0 & ? \end{array}$$

Este ejemplo mostró que solo cinco ciclos del algoritmo de Euclides se combinaron en un paso múltiple.

El método de Lehmer se puede formular de la siguiente manera:

Datos: $U, V \in Z$ *representando los enteros de precisión múltiple y p como los dígitos iniciales*
Salida: GCD(U,V)

Implementación

```
function LhemerGCD( U , V , p ):
   base = 10
        H =  pow(base,p)

   ZZ H pow(base,p)
   while(  V >= H ):

      //Initialize
      k = U.size() - p
      u = U / pow(base, k)
      v = V / pow(base,k)
      A = 1  B = 0  C = 0  D = 1

      while (v+C != 0 &&  v+D != 0):

         //Test quotient
         q0 = (u + A) / (v + C)
         q1 = (u + B) / (v + D)

         if (q0 != q1)
            break;
         endif
```

```
        //Emulate Euclid
        T = A - q0 * C     A = C      C = T
        T = B - q0 * D     B = D      D = T
        T = u - q0 * v     u = v      v = T

    //Multiprecision step
    if( B == 0):
        t = mod(U,V)
        U = V
        V = t
    endif

    else:
        t = A*U + B*V;
        w = C*U + D*V;
        U = t;
        V = w;
    endelse

    if(V == 0)
        return U
    endif

  endwhile

// si V es lo suficiente pequeño para representarse como single-precision usar gcd()
return gcd(U,V)

endfunction
```

Seguimiento Numérico:

input:

u = 27182
v = 10000
p= 3

Reducimos nuestros números de precisión múltiple a precisión simple

u = 271
v = 100

Cálculo de cocientes simultáneamente hasta que los cocientes sean diferentes

u'	v'	q'
272	100	2
100	72	1
72	28	2
28	16	1

u'	v'	q'
271	101	2
101	69	1
69	32	2
32	5	6

cuando son diferentes completamos los pasos con los números de precisión múltiple

u'	v'	q
2818	1546	1

Cálculo de cocientes simultáneamente hasta que los cocientes sean diferentes

u'	v'	q'
282	154	1
154	128	1
128	26	4
26	24	1

u'	v'	q'
281	155	1
155	126	1
126	29	4
29	10	9

cuando son diferentes completamos los pasos con los números de precisión múltiple

u'	v'	q
274	176	1

cómo 274 y 176 son ahora lo suficientemente pequeñas , del tamaño de un entero de precisión simple , podemos usar un algoritmo gcd como euclides o cualquier otro.

Euclides para 274 y 176

1. 274=176*1+98
2. 176=98*1+78
3. 98=78*1+20
4. 78=20*3+18
5. 20=18*1+2
6. 18=2*9+0

El MCD de 274 y 176 es igual a 2.

```
ZZ Lehmer_del_mcd(ZZ U, ZZ V, ZZ p)
{
    ZZ H(empower(ZZ(10), p));
    while (V >= H)
    {
        //Initialize
        ZZ k, u, v, A, B, C, D, q0, q1, T, t, w;
        k = ZZtoStr(U).size() - p;
        u = U / empower(ZZ(10), k);
        v = V / empower(ZZ(10), k);
        A = 1;
        B = 0;
        C = 0;
        D = 1;
        while (v + C != 0 && v + D != 0)
        {
            //Test quotient
            q0 = (u + A) / (v + C);
            q1 = (u + B) / (v + D);
            if (q0 != q1)
                break;
            //Emulate Euclid
            T = A - q0 * C;
            A = C;
            C = T;
            T = B - q0 * D;
            B = D;
            D = T;
            T = u - q0 * v;
            u = v;
            v = T;
        }
        //Multiprecision step
```

```
        if (B == 0)
        {
            t = module(U, V);
            U = V;
            V = t;
        }
        else
        {
            t = A * U + B * V;
            w = C * U + D * V;
            U = t;
            V = w;
        }
        if (V == 0)
            return U;
    }
    return Euclides_con_menor_resto(U, V);
}
```

Algoritmo binario

Definición:

La idea principal de este algoritmo es cambiar el $mcd(a,\ b)$por dos números equivalentes más pequeños. Se basa en restas y divisiones por 2, siguiendo 3 reglas.

El enfoque matemático sobre el que se sustenta

Las tres reglas en las que se basa este algoritmo son:

a) $Si\ a, b\ son\ pares\ \Rightarrow\ mcd(a,\ b)\ =\ 2\ mcd\ (\frac{a}{2}, \frac{b}{2})$ Regla 1

b) $Si\ "a"\ es\ par\ y\ "b"\ impar\ \Rightarrow\ mcd(a,\ b)\ =\ mcd\ (\frac{a}{2}, b)$ Regla 2

c) $Si\ a, b\ son\ pares\ \Rightarrow\ mcd(a,\ b)\ =\ mcd\ (\frac{|a-b|}{2}, b)\ =\ mcd\ (\frac{|a-b|}{2}, a)$ Regla 3

En la regla 3 se debe de operar como $mcd(a,\ b)\ =\ mcd\ (\frac{|a-b|}{2}, Min(a, b))$

Funciona de la siguiente manera, supongamos dos números $a,\ b$. Si $a\ y\ b$ son pares entonces aplicamos la regla 1 n veces, hasta que alguno de los dos sea un impar. Al final de esto se multiplica por $2^{\ n}$ como contrapartida por usar la primera regla n veces. Seguido si aún $a\ o\ b$ es par, entonces usamos la regla 2 hasta que los dos sean impares. Una vez los dos sean impares se usa la tercera regla y luego se alterna entre las reglas 2 y 3 de acuerdo a el cociente $\frac{|a-b|}{2}$ sea par o impar.

El pseudo-algoritmo

```
Function mcdBinario (u,v)
t Long, g Long, a Long, b Long
g=1
a=abs(u)
b=abs(v)
While a mod 2 == 0 And b Mod 2 == 0
      a=a/2
      b=b/2
      g=2*g
WhileEnd
While a<>0
      If  a Mod 2 == 0 Then
            a=a/2
      ElseIf  b Mod 2 == 0 Then
            b=b/2
      Else t=abs(a-b)/2
            If a>= b Then
                  a=t
```

```
            Else b=t
            End If
      End If
WhileEnd
return g*b
End Function
```

Seguimiento numérico

$$
\begin{aligned}
mcd(89,\ 44) &= mcd(89, 44) \quad \rightarrow \ Regla\ 2 \\
&= mcd(22,\ 89) \quad \rightarrow \ Regla\ 2 \\
&= mcd(11,\ 89) \quad \rightarrow \ Regla\ 3 \\
&\quad mcd(\tfrac{|11-89|}{2},\ Min(11, 89)) \\
&= mcd(39,\ 11) \quad \rightarrow \ Regla\ 3 \\
&\quad mcd(\tfrac{|39-11|}{2},\ Min(39, 11)) \\
&= mcd(14,\ 11), \quad \rightarrow \ Regla\ 2 \\
&= mcd(7,\ 11) \quad \rightarrow \ Regla\ 3 \\
&\quad mcd(\tfrac{|7-11|}{2},\ Min(7, 11)) \\
&= mcd(2,\ 7) \quad \rightarrow \ Regla\ 2 \\
&= mcd(1,\ 7) \quad \rightarrow \ Regla\ 3 \\
&\quad mcd(\tfrac{|1-7|}{2},\ Min(1, 7)) \\
&= mcd(3,\ 1) \quad \rightarrow \ Regla\ 3 \\
&\quad mcd(\tfrac{|3-1|}{2},\ Min(3, 1)) \\
&= mcd(1,\ 1) \quad \rightarrow \ Regla\ 3 \\
&\quad mcd(\tfrac{|1-1|}{2},\ Min(1, 1)) \\
&= mcd(0,\ 1) \\
&= 1
\end{aligned}
$$

Implementación en C++

```
ZZ binario(ZZ u, ZZ v)
{
    ZZ t, g, a, b;
    g = 1;
    a = abs(u);
    b = abs(v);
    while (a % 2 == 0 && b % 2 == 0)
    {
        a = a / 2;
        b = b / 2;
        g = 2 * g;
    }
    while (a != 0)
    {
        if (a % 2 == 0)
            a = a / 2;
        else if (b % 2 == 0)
            b = b / 2;
        else
        {
            t = abs(a - b) / 2;
            if (a >= b)
                a = t;
            else
                b = t;
        }
    }
    return g * b;
}
```

Análisis de Algoritmos

Características del procesador y sistema operativo:

- AMD Ryzen 5 3400G with Radeon Vega Graphics (8 CPUs), ~3.7Ghz
- Windows 10 Pro 64 bits
- 16384MB RAM

Tiempo de ejecución vs. Nro. de Bits

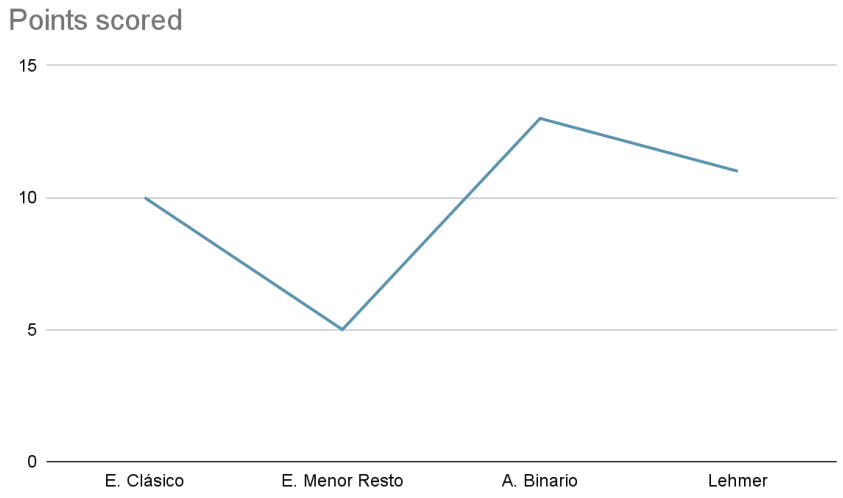
128 bits											
	Intentos										Promedio
	1	2	3	4	5	6	7	8	9	10	
Euclides clásico.	0,01	0,009	0,011	0,01	0,009	0,009	0,01	0,009	0,01	0,01	0,0098
Euclides con menor resto	0,01	0,011	0,011	0,011	0,011	0,01	0,012	0,01	0,01	0,014	0,011
Lehmer del mcd	0,023	0,015	0,016	0,017	0,013	0,016	0,014	0,014	0,016	0,013	0,0157
Binario del mcd	0,009	0,009	0,01	0,013	0,01	0,009	0,01	0,009	0,01	0,009	0,0097

512 bits											
	Intentos										Promedio
	1	2	3	4	5	6	7	8	9	10	
Euclides clásico.	0,017	0,013	0,018	0,016	0,021	0,014	0,018	0,015	0,014	0,016	0,0162
Euclides con menor resto	0,013	0,014	0,012	0,012	0,013	0,014	0,012	0,014	0,011	0,013	0,0128
Lehmer del mcd	0,018	0,02	0,021	0,023	0,022	0,021	0,018	0,018	0,022	0,019	0,0202
Binario del mcd	0,009	0,014	0,017	0,014	0,011	0,012	0,01	0,019	0,012	0,01	0,0128

1024 bits											
	Intentos										Promedio
	1	2	3	4	5	6	7	8	9	10	
Euclides clásico.	0,017	0,016	0,018	0,017	0,013	0,015	0,016	0,017	0,015	0,013	0,0157
Euclides con menor resto	0,015	0,016	0,016	0,015	0,016	0,014	0,14	0,016	0,014	0,015	0,0277
Lehmer del mcd	0,016	0,015	0,016	0,015	0,017	0,014	0,014	0,014	0,015	0,018	0,0154
Binario del mcd	0,011	0,012	0,013	0,013	0,013	0,013	0,011	0,014	0,013	0,013	0,0126

2048 bits											
	Intentos										Promedio
	1	2	3	4	5	6	7	8	9	10	
Euclides clásico.	0,023	0,019	0,02	0,021	0,019	0,024	0,019	0,023	0,021	0,024	0,0213
Euclides con menor resto	0,024	0,021	0,021	0,02	0,02	0,022	0,02	0,021	0,02	0,021	0,021
Lehmer del mcd	0,019	0,021	0,019	0,021	0,022	0,02	0,019	0,019	0,023	0,023	0,0206
Binario del mcd	0,013	0,014	0,014	0,013	0,018	0,012	0,014	0,016	0,014	0,013	0,0141

Número de loops



Conclusión

El algoritmo binario resulta ser el más eficiente computacionalmente, debido a que si bien es cierto requiere de un mayor procedimiento para resolver el máximo común divisor, la ventaja de este algoritmo yace en las operaciones dividiendo y multiplicando por dos, desde un punto de vista computacional, estas operaciones se hacen en representación binaria, así que solo se requeriría de un desplazamiento de bits. [7] pág 339.

Si bien es cierto que existen muchas variantes de este algoritmo en el presente trabajo se utilizó el más conocido mayor justificar matemáticamente.

Bibliografía:

[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
[04] Chapter 10. Number theory and Cryptography. https://silo.tips/download/chapter-number-theory-and-cryptography-contents
[05] Euclidean algorithm - From Wikipedia Ref. [22][23]. Euclidean algorithm - Wikipedia
[06] Introducción a la Teoría de Números. Ejemplos y algoritmos. Walter Mora. introducción-teoría-números.pdf (tec.ac.cr)

[07] D. Knuth (1981). The Art of Computer Programming. Volume 2: Seminumerical Algorithms.

[08] T. Jebelean (1993). “Comparing several GCD algorithms”. En ARITH-11: IEEE Symposium on Computer Arithmetic. IEEE, New York, 180-185