Artículo de sección
Revista digital Matemática, Educación e Internet (www.cidse.itcr.ac.cr/revistamate/). Vol. 10, N^o 2. Marzo 2010

Cálculo del máximo común divisor:¿Porqué no se usa el algoritmo de Euclides?.

Walter Mora F.
wmora2@yahoo.com.mx
Escuela de Matemáticas
Instituto Tecnológico de Costa Rica

Resumen

En este artículo se describen varios algoritmos para calcular el máximo común divisor de dos números enteros. Los algoritmos se pueden usar tanto para cálculo manual como para cálculo computacional. Las implementaciones se hacen en la hoja electrónica de OpenOffice.org, usando el lenguaje OOoBasic. En la presentación de los algoritmos se indica el fundamento teórico y las razones por las cuales cada algoritmo tiene mejor rendimiento que los anteriores. El desarrollo teórico solo requiere conocimientos básicos de divisibilidad.

Palabras claves: Máximo común divisor, teorema de división, algoritmo de Euclides, algoritmo del menor resto, algoritmo binario.

1.1 Introducción

> *"...tal parece que ni siquiera un procedimiento tan venerable como el algoritmo de Euclides puede soportar el progreso"*
> Donald Knuth ([6], pág 340).

El cálculo del máximo común divisor (mcd) de dos enteros grandes (y también de dos polinomios) es omnipresente en el cálculo con racionales, criptografía de clave pública y álgebra computacional. De hecho los cálculos algebraicos usuales gastan más de la mitad del tiempo de ejecución en el cálculo del máximo común divisor de enteros frecuentemente muy grandes ([4]). Como los cálculos son muy demandantes, se requiere algoritmos muy eficientes para el cálculo del mcd.

Antes de la década de los 70's, el cálculo se hacía con el algoritmo de Euclides clásico o con la versión mejorada de Lehmer[1] (1938). El algoritmo de Euclides (además de su gran valor teórico)

[1]Esta variante del algoritmo de Euclides se aplica para calcular el mcd(a,b) si a y b son números muy grandes. La idea es aplicar el algoritmo de Euclides usando, en los primeros pasos, $\lfloor a/10^k \rfloor$ y $\lfloor b/10^k \rfloor$ en vez de a y b. Una descripción completa se puede ver en [6], págs 345-348.

es sencillo de enunciar e implementar y es muy eficiente, pero hay algoritmos igual de sencillos y más rápidos. Si los números vienen codificados en binario, teóricamente habría una mejora del 60% ([10]) en la eficiencia. Estos algoritmos se usan desde hace unos cuarenta años atrás. El más popular es el algoritmo "binario" para el cálculo del mcd (algunos autores le llaman "algoritmo binario de Euclides). Este algoritmo fue descubierto por el físico Israelí J. Stein en 1961. D. Knuth hace la observación de que este algoritmo podría tener un pedigrí muy distinguido pues parece ser que ya era conocido en la antigua China (un siglo d.C.). 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). 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. Por ejemplo, $344 = (101011000)_2$, $344/2 = (10101100)_2$ y $2 \cdot 344 = (101011000)_2$. Refiriéndose al algoritmo binario Donald Knuth decía en 1980, "...parece que ni siquiera un procedimiento tan venerable como el algoritmo de Euclides puede soportar el progreso" ([6], pág 340).

Al igual que hay un algoritmo extendido de Euclides también hay una versión extendida del algoritmo binario más eficiente y también hay una versión para polinomios ($\mathbb{Z}[x]$ es tanto un dominio Euclidiano como un "dominio de Stein"). De nuevo aquí, mientras que el algoritmo de Euclides requiere, en general, división por polinomios de grado mayor o igual a uno, el algoritmo binario solo requiere dividir por x. Sin embargo, el cálculo del máximo común divisor de dos polinomios con coeficientes enteros no se hace con ninguno de estos algoritmos, más bien se usan algoritmos modulares (Mathematica) o el llamado "algoritmo heurístico para polinomios" (Maple).

El algoritmo de Euclides es muy adecuado para el tratamiento teórico que se hace en los libros de álgebra y teoría de números. Además es muy eficiente para el cálculo. El algoritmo binario y sus variantes (algunas más eficientes que el algoritmo original), aparece de manera natural en el contexto de la teoría algorítmica de números porque aquí si importa ganar en eficiencia. En todo caso, no estaría del todo mal si en los libros de teoría de números, además de incluir notas históricas, apareciera un epílogo contando por donde va la novela en nuestros días.

En este trabajo se muestran cuatro algoritmos: El algoritmo clásico de Euclides, el algoritmo de Euclides con "menor resto", el algoritmo binario y el algoritmo LSBGCD (left-shift binary algorithm) que vendría a ser como una versión binaria del algoritmo de Euclides. Como las implementaciones son sencillas, se implementan en la hoja electrónica de OpenOffice.org, usando el lenguaje OOoBasic.

1.2 Parte entera.

La función parte entera superior de un número x, denotada `ceil(x)`, devuelve el menor entero mayor o igual a x, es decir,

$$\lceil x \rceil = \text{Mín}\{n \in \mathbb{Z} \mid n \geq x\}.$$

Por ejemplo, `Ceil(2.25)=3`, `Ceil(2)=2` y `Ceil(-2.25)=-2`. La función parte entera inferior, denotada `Floor(x)`, devuelve el más grande entero menor o igual a x, es decir,

$$\lfloor x \rfloor = \text{Máx}\{n \in \mathbb{Z} \mid n \leq x\}.$$

Por ejemplo, `Floor(2.8)=2, Floor(-2)=-2` y `Floor(-2.3)=-3`.

Notemos que $\lceil x \rceil = \lfloor x \rfloor$ si y sólo si x es entero, en otro caso $\lceil x \rceil = \lfloor x \rfloor + 1$.

El entero "más cercano" a $x \geq 0$ es $\lfloor x + 1/2 \rfloor$ ([5],pág 70,ejercicio 5). En la sección 1.6 usaremos la fórmula (para $x \geq 0$),

$$\lfloor x + 1/2 \rfloor = \begin{cases} \lfloor x \rfloor & \text{si} \quad \text{pfrac}(x) \leq 1/2 \\ \\ \lfloor x \rfloor + 1 & \text{si} \quad \text{pfrac}(x) > 1/2 \end{cases} \tag{1.1}$$

donde la parte fraccionaria, denotada "pfrac", de un número real $x \geq 0$ se define con la ecuación

$$x = \lfloor x \rfloor + \text{pfrac}(x).$$

Por ejemplo, $2.71 = 2 + 0.71 \Rightarrow \text{pfrac}(x) = 0.71$. En la figura (1.1) se muestra la fórmula desde el punto de vista geométrico.

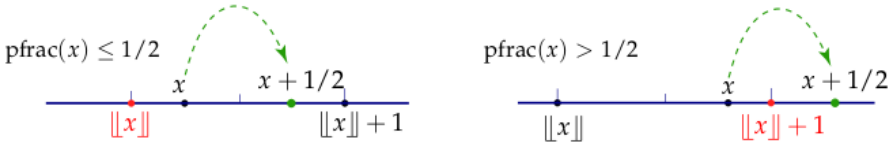


Figura 1.1

Aspectos computacionales. En OOoBasic tenemos la división entera `a\b` (con barra invertida) y la función `Int`.

$$\texttt{Int}(x) = \lfloor x \rfloor \quad \text{e} \quad \texttt{Int}(x) + 1 = \lceil x \rceil$$

$$\texttt{a\b} = \begin{cases} \lfloor a/b \rfloor & \text{si} \quad a/b \geq 0 \\ \\ \lceil a/b \rceil & \text{si} \quad a/b < 0 \end{cases}$$

Por ejemplo, `-7\2`$= -3$ y `3\2`$= 1$.

1.3 Teorema de la división.

El teorema de la división[2] establece que si $a, b \in \mathbb{Z}$ con $b \neq 0$, existen $q, r \in \mathbb{Z}$ únicos tales que

$$a = b \cdot q + r \quad \text{con} \quad 0 \leq r \leq |b|.$$

[2]Extrañamente a veces a este teorema se le llama "algoritmo de la división". En el contexto computacional, el algoritmo de a división se refiere a los pasos para dividir u por v en el caso de que u y v estén representados en base b. En este caso, el algoritmo calcula $\lfloor u/v \rfloor$.

A q se le llama *cociente* y r se le llama *resto* y, por supuesto, $r = a - bq$

Esta manera de enunciar este teorema es muy apropiada para fines teóricos. Desde el punto de vista computacional es mejor enunciar este teorema de otra manera. Recordemos que

$$\begin{cases} \operatorname{sgn}(b) = 1 & \text{si} \quad b > 0, \\ \operatorname{sgn}(b) = -1 & \text{si} \quad b < 0. \end{cases}$$

1.4 División con menor resto.

El teorema de la división[3] establece que si $a, b \in \mathbb{Z}$ con $b \neq 0$, existen $q, r \in \mathbb{Z}$ únicos tales que

$$a = b \cdot q + r \quad \text{con} \quad 0 \leq r \leq |b|.$$

A q se le llama *cociente* y r se le llama *resto* y, por supuesto, $r = a - bq$

Esta manera de enunciar este teorema es muy apropiada para fines teóricos. Que el resto sea positivo es adecuado, como vimos, para mostrar unicidad.

Sin embargo el resto no tiene porque ser positivo, por ejemplo si $a = 144$ y $b = 89$,

$$144 = 89 \cdot 1 + 55, \quad \text{resto} \quad r_2 = 55 < b = 89$$

$$144 = 89 \cdot 2 - 34, \quad \text{resto} \quad r_1 = 34 < b = 89$$

Cuando calculamos por ejemplo el máximo común divisor de dos números usando el algoritmo de Euclides, el número de pasos se reduce si tomamos el resto más pequeño en cada paso. Esto no afecta el algoritmo.

Veamos los cálculos. Recordemos que $\begin{cases} \operatorname{sgn}(b) = 1 & \text{si} \quad b > 0, \\ \operatorname{sgn}(b) = -1 & \text{si} \quad b < 0. \end{cases}$

[3]Extrañamente a veces a este teorema se le llama "algoritmo de la división". En el contexto computacional, el algoritmo de a división se refiere a los pasos para dividir u por v en el caso de que u y v estén representados en base b. En este caso, el algoritmo calcula $\lfloor u/v \rfloor$.

Teorema 1.1

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

$$\begin{cases} q = \lfloor a/b \rfloor & \text{si} \quad b > 0, \\ q = \lfloor a/b \rfloor + 1 & \text{si} \quad b < 0, \ \text{y} \ \ a/b \notin \mathbb{Z} \end{cases} \tag{1.2}$$

entonces la división con resto se puede hacer de dos maneras,

$$\begin{cases} \textbf{a.)} \ a = bq + r_2 & \text{con} \quad 0 \leq r_2 < |b| \\ \textbf{b.)} \ a = b(q + \operatorname{sgn}(b)) - r_1 & \text{con} \quad 0 \leq r_1 < |b| \end{cases}$$

Además, si $a \geq 0$, $b > 0$ y $r = \min\{r_1, r_2\}$, entonces $r = |a - b \cdot \lfloor a/b + 1/2 \rfloor|$

¿Cómo llegamos a este resultado? Vamos a ver cómo.

Cálculo de q y r_i Sean $a,b \in \mathbb{Z}$ con $b \neq 0$. Usando el principio del buen orden se puede establecer que existe $q \in \mathbb{Z}$ tal que bq es el múltiplo de b más cercano a a por la izquierda (ver figura 1.2). Por tanto,

$$bq \leq a < bq + b \ \text{ si } \ b > 0 \qquad \text{y} \qquad bq \leq a < b(q-1) \ \text{ si } \ b < 0.$$

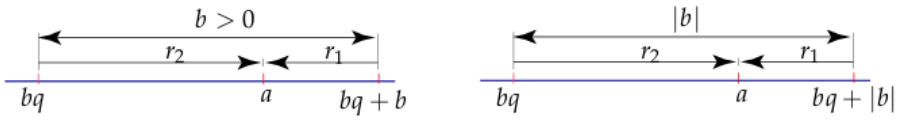


Figura 1.2

Entonces, podemos expresar a en términos de bq con resto positivo o en términos de $bq + |b| = b(q + \operatorname{sng}(b))$ con resto negativo (esta fórmula funciona para b positivo o negativo).

Recordemos que $q = \lfloor a/b \rfloor$ si $b > 0$ y $q = \lceil a/b \rceil$ si $b < 0$. Para usar un mismo q, usamos el hecho de que si $a/b \notin \mathbb{Z}$, entonces $\lceil a/b \rceil = \lfloor a/b \rfloor + 1$ (si $a/b \in Z$ el resto sería cero). Por tanto,

$$\begin{cases} q = \lfloor a/b \rfloor & \text{si} \quad b > 0 \\ q = \lfloor a/b \rfloor + 1 & \text{si} \quad b < 0 \end{cases} \tag{1.3}$$

Entonces tenemos (sin importar el signo de a y b),

$$\begin{cases} \textbf{a.)} \ a = bq + r_2 & \text{con} \quad 0 \leq r_2 < |b| \\ \textbf{b.)} \ a = b(q + \operatorname{sgn}(b)) - r_1 & \text{con} \quad 0 \leq r_1 < |b| \end{cases}$$

Ejemplo 1.1

a.) Si $a = 144$ y $b = 89$,

$$\begin{aligned} 144 &= 89 \cdot 1 + 55, \quad \text{con resto } r_2 = 55 < b = 89 \\ 144 &= 89 \cdot 2 - 34, \quad \text{con resto } r_1 = 34 < b = 89 \end{aligned}$$

b.) Si $a = 144$ y $b = -89$, entonces $q = \lfloor\!\lfloor 144/(-89) \rfloor\!\rfloor + 1 = -2 + 1 = -1$ y $q + \text{sgn}(b) = -2$. Entonces,

$$\begin{aligned} 144 &= -89 \cdot -1 + 55, \quad \text{con resto } r_2 = 55 < |b| = 89 \\ 144 &= -89 \cdot -2 - 34, \quad \text{con resto } r_1 = 34 < |b| = 89 \end{aligned}$$

Cálculo del menor resto. En la sección 1.6 vamos a necesitar el teorema de la división pero con el menor resto. Para simplificar los cálculos, queremos calcular el menor resto usando una fórmula directa. Como se observa en la figura 1.3, uno de los restos es menor que $|b|/2$. Si $r = \text{Mín}\{r_1, r_2\}$ entonces, existe $q' \in \mathbb{Z}$ tal que

$$a = bq' \pm r \quad \text{con} \quad 0 \leq r \leq |b|/2.$$

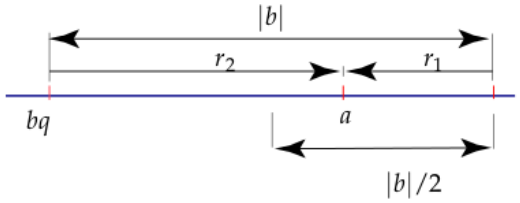


Figura 1.3

Para los cálculos que vamos a hacer aquí solo necesitamos tratar el caso en que $a \geq 0$ y $b > 0$. Para comparar los restos $a - b \cdot \lfloor\!\lfloor a/b \rfloor\!\rfloor$ y $b \cdot (\lfloor\!\lfloor a/b \rfloor\!\rfloor + 1) - a$ usamos el hecho de que $a/b = \lfloor\!\lfloor a/b \rfloor\!\rfloor + \text{pfrac}(a/b)$.

- El menor resto es $a - b \cdot \lfloor\!\lfloor a/b \rfloor\!\rfloor$ si $\text{pfrac}(a/b) \leq 1/2$. En efecto,

$$\begin{aligned} a - b \cdot \lfloor\!\lfloor a/b \rfloor\!\rfloor &\leq b \cdot (\lfloor\!\lfloor a/b \rfloor\!\rfloor + 1) - a \\ 2a &\leq 2b \cdot \lfloor\!\lfloor a/b \rfloor\!\rfloor + b \\ 2a/b &\leq 2\lfloor\!\lfloor a/b \rfloor\!\rfloor + 1, \quad \text{como} \quad a/b = \lfloor\!\lfloor a/b \rfloor\!\rfloor + \text{pfrac}(a/b), \\ \text{pfrac}(a/b) &\leq 1/2. \end{aligned}$$

- De manera análoga, $a - b \cdot \lfloor\!\lfloor a/b \rfloor\!\rfloor \geq b \cdot (\lfloor\!\lfloor a/b \rfloor\!\rfloor + 1) - a$ si $\text{pfrac}(a/b) \geq 1/2$.

Así,

$$\text{el menor resto es } r = \begin{cases} a - b \cdot \lfloor\!\lfloor a/b \rfloor\!\rfloor & \text{si} \quad \text{pfrac}(a/b) \leq 1/2, \\ |a - b \cdot (\lfloor\!\lfloor a/b \rfloor\!\rfloor + 1)| & \text{si} \quad \text{pfrac}(a/b) > 1/2. \end{cases}$$

Como habíamos establecido antes (ecuación 1.4),

$$\lfloor\!\lfloor a/b + 1/2 \rfloor\!\rfloor = \begin{cases} \lfloor\!\lfloor a/b \rfloor\!\rfloor & \text{si} \quad \text{pfrac}(a/b) \leq 1/2, \\ \lfloor\!\lfloor a/b \rfloor\!\rfloor + 1 & \text{si} \quad \text{pfrac}(a/b) > 1/2, \end{cases}$$

entonces,

$$\text{el menor resto es } r = |a - b \cdot \lfloor\!\lfloor a/b + 1/2 \rfloor\!\rfloor|.$$

Aspectos computacionales. En OOoBasic de Libreoffice "el resto" se calcula con la función binaria `Mod`. Se implementa como `a Mod b`$= a - b \cdot$ `(a\b)` y tenemos

$$a = b \cdot \texttt{a\b} + \texttt{a Mod b}$$

Por ejemplo, si $a = -144$ y $b = -89$ entonces `a\b`$= 1$ y `a Mod b`$= -55$.

Si `a Mod b`< 0 y queremos el resto r_2 positivo, la figura 1.3 nos sugiere $r_2 =$ `a Mod b` $+ |b|$, en este caso,

$$a = b \cdot (\texttt{a\b} - \text{sgn}(b)) + \texttt{a Mod b} + \texttt{|b|}$$

El menor resto se calcula como $r = |a - b \cdot \lfloor\!\lfloor a/b + 1/2 \rfloor\!\rfloor| =$`Abs(a-b*Int(a/b-1/2))`.

1.5 Algoritmo de Euclides.

El algoritmo de Euclides encuentra el máximo común divisor de dos enteros. 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[4].

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

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

$r_n = \text{mcd}\,(a,b)$ pues $\text{mcd}\,(a,b) = \text{mcd}\,(b,r_1) = \text{mcd}\,(r_1,r_2) = \ ... \ = \text{mcd}\,(r_n,0) = r_n$.

EJEMPLO 1.1

Vamos a aplicar el algoritmo de Euclides para calcular $\text{mcd}\,(89,144)$. Aquí estamos aplicando el algoritmo sobre dos números consecutivos de Fibonacci. Este tipo de pares son los que le demandan mayor esfuerzo al algoritmo de Euclides ([3],págs 68-69) pues siempre $q = 1$.

$$\begin{array}{rcl@{\qquad}c@{\qquad}rcl} 144 & = & 89 \cdot 1 + 55 & \Longrightarrow & \text{mcd}\,(89,144) & = & \text{mcd}\,(89,55) \\ 89 & = & 55 \cdot 1 + 34 & & & = & \text{mcd}\,(55,34) \\ 55 & = & 34 \cdot 1 + 21 & & & = & \text{mcd}\,(34,21) \\ 34 & = & 21 \cdot 1 + 13 & & & = & \text{mcd}\,(21,13) \\ 21 & = & 13 \cdot 1 + 8 & & & = & \text{mcd}\,(13,8) \\ 13 & = & 8 \cdot 1 + 5 & & & = & \text{mcd}\,(8,5) \\ 8 & = & 5 \cdot 1 + 3 & & & = & \text{mcd}\,(5,3) \\ 5 & = & 3 \cdot 1 + 2 & & & = & \text{mcd}\,(3,2) \\ 3 & = & 2 \cdot 1 + 1 & & & = & \text{mcd}\,(2,1) \\ 2 & = & 1 \cdot 2 + 0 & & & = & \text{mcd}\,(1,0) = 1. \end{array}$$

$$\text{mcd}\,(89,144) = 1.$$

[4]La versión original de Euclides no es esta, en ese tiempo no había noción del cero ni se consideraba a la unidad como un divisor. Para los griegos antiguos dos enteros positivos eran o ambos iguales a la unidad o eran primos relativos o tenían un máximo común divisor. La versión original ([6],pág 336) sería (sin usar lenguaje geométrico): Sean a, b mayores que la unidad. Si b divide a a, el mcd es b, si el resto de dividir a por b es la unidad, los números son primos relativos, en otro caso reemplace el par de valores (a,b) por (r,b).

1.5.1 Algoritmo e implementación.

La implementación de este algoritmo se hace solo con fines ilustrativos. El algoritmo funciona bien si usamos $r =$`a Mod b`. Pero, para seguir el algoritmo al pie de la letra, vamos a usar restos positivos. Se necesitan tres variables: `c` para el nuevo dividendo, d para el nuevo divisor y r para el resto (positivo). La función que calcula q la llamamos `cquo`, es decir, `q=cquo(a,b)`. La función que calcula el resto positivo la llamamos `crem`, `r=crem(a,b)`.

Algoritmo 1.1: Máximo común divisor: Algoritmo de Euclides

Datos: $a, b \in \mathbb{Z},\ b \neq 0$.
Salida: $\text{mcd}\,(a,b)$

1 **if** $a = 0$ **then**
2 **return** $\text{mcd}\,(a,b) = b$
3 $c = a,\ d = b;$
4 **while** $d \neq 0$ **do**
5 $r =$crem$(c,d);$
6 $c = d;$
7 $d = r;$
8 **return** $\text{mcd}\,(a,b) = c;$

$$
\begin{array}{rcl}
\underset{c}{144} & = & \underset{d}{89}\cdot 1 + \underset{\text{crem(c,d)}}{55} \\
\swarrow & & \\
\underset{c}{89} & = & \underset{d}{55}\cdot 1 + \underset{\text{crem(c,d)}}{34} \\
\swarrow & & \\
\underset{c}{55} & = & \underset{d}{34}\cdot 1 + \underset{\text{crem(c,d)}}{21} \\
\swarrow & & \\
\underset{c}{34} & = & \underset{d}{21}\cdot 1 + \underset{\text{crem(c,d)}}{13} \\
\swarrow & & \\
\vdots & & \\
\underset{c}{2} & = & \underset{d}{1}\cdot 2 + 0
\end{array}
$$

$$\text{mcd}\,(89,144) = 1.$$

Implementación. La función `crem` necesita la función cquo.

```
Function cquo(a,b) As Long
 Dim q As Long
   If b=0 then
      msgbox "Error, b=0"
      Exit Function
   End If
   q = Int(a/b)
   If b<0 Then
      q = q+1
   End If
   cquo = q
End Function

Function crem(a,b) As Long
     crem = a-b*cquo(a,b)  rem resto positivo
End Function

Function mcdEuclides(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= crem(c,d)
        c = d
        d = r
    Wend
  End If
mcdEuclides = c
 End Function
```

1.6 Algoritmo de Euclides con menor resto.

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 $r_{i-1}/2$ con lo que, en general, podría haber una reducción en el número de divisiones. Kronecker estableció[5] 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.

Como $\text{mcd}\,(a,b) = \text{mcd}\,(|a|,|b|)$ vamos a suponer que $a \geq 0$ y $b > 0$. Recordemos que

$$a = b \cdot \lfloor\!\lfloor a/b \rfloor\!\rfloor + r_2. \;\; 0 \leq r_2 < b$$

$$a = b \cdot (\lfloor\!\lfloor a/b \rfloor\!\rfloor + 1) - r_1. \;\; 0 \leq r_1 < b$$

Para mejorar un poco el desempeño del algoritmo de Euclides, escogemos en cada paso el menor resto, es decir, $r = \text{Mín}\{r_1, r_2\} = \text{Mín}\{|a - b \cdot \lfloor\!\lfloor a/b \rfloor\!\rfloor|,\ |a - b \cdot (\lfloor\!\lfloor a/b \rfloor\!\rfloor + 1)|\}$. De esta manera $r \leq b/2$.

El algoritmo de Euclides sigue siendo válido pues si tomamos el menor resto r en cada paso, $\text{mcd}\,(a,b) = \text{mcd}\,(b,r)$ y de nuevo obtenemos una sucesión decreciente de restos, el último resto no nulo $|r_n|$ es el mcd de a y b. Por supuesto,

$$r = \text{Mín}\{r_1, r_2\} = a - b \cdot \lfloor\!\lfloor a/b + 1/2 \rfloor\!\rfloor.$$

■ **EJEMPLO 1.2**

Vamos a aplicar el algoritmo de Euclides, usando el menor resto, para calcular $\text{mcd}\,(89,144)$. Como $\text{mcd}\,(a,b) = \text{mcd}\,(|a|,|b|)$, en cada paso usamos dividendo y divisor positivo.

[5] L. Kronecker. "*Vorlesungen über Zahlentheorie*". Vol 1. Teubner. 1901. Hay una re-impresión de Springer-Verlag del año 1978.

$$\begin{array}{rcl@{\quad\Longrightarrow\quad}rcl}
144 &=& 89\cdot 2-34 & \mcd(89,144) &=& \mcd(89,34)\\
89 &=& 34\cdot 3-13 & &=& \mcd(34,13)\\
34 &=& 13\cdot 3-5 & &=& \mcd(13,5)\\
13 &=& 5\cdot 3+2 & &=& \mcd(5,2)\\
5 &=& 2\cdot 2+1 & &=& \mcd(2,1)\\
2 &=& 1\cdot 2+0 & &=& \mcd(1,0)=1
\end{array}$$

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

1.6.1 Implementación.

La implementación es la misma que la del algoritmo de Euclides clásico. Solo vamos a cambiar la función `crem(a,b)` por la nueva función `srem(a,b)` $= a-b\cdot\lfloor\!\lfloor a/b+1/2\rfloor\!\rfloor$. El valor absoluto no es necesario porque lo que interesa en el algoritmo es que los restos disminuyan (en valor absoluto).

```
 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)  rem q= Int(c/d+1/2)
        c = d
        d = r
    Wend
  End If
mcdMenorResto = Abs(c)
 End Function
```

Por ejemplo, si imprimimos cada paso de la implementación con $a=-144$ y $b=-89$, se obtiene

```
-144 = -89*2   + 34      mcd(-144,-89)= 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*3   + 1                    = mcd(2,1)
-2   =   1*-2  + 0                    = mcd(1,0)=1
```

1.7 Algoritmo binario.

El algoritmo binario opera con la misma idea, cambiar $\mcd(a,b)$ por el mcd de dos números eventualmente más pequeños, solo que esta vez solo restamos y dividimos por 2. El algoritmo opera con las siguientes teoremas,

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

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

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

La **Regla 3** se aplica como $\text{mcd}\,(a,b) = \text{mcd}\left(\frac{|a-b|}{2},\text{Mín}\{a,b\}\right)$. La prueba de estas reglas están al final de esta sección.

El algoritmo procede así: Supongamos que $a,b \in \mathbb{Z},\ a \geq 0, b > 0$. 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 $\frac{|a-b|}{2}$ sea par o impar.

■ **EJEMPLO 1.3**

Vamos a aplicar el algoritmo de binario para calcular $\text{mcd}\,(89,144)$ y $\text{mcd}\,(8,48)$. Recordemos que aquí la ganancia no esta en la disminución del número de pasos (de hecho podrían ser más pasos que los que utiliza el algoritmo de Euclides) sino en operar dividiendo por 2.

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

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

Por supuesto, en cálculo manual terminamos cuando obtenemos $\text{mcd}\,(0,d) = d$ o $\text{mcd}\,(1,d) = 1$.

El algoritmo funciona pues la aplicación de las reglas va produciendo una sucesión de pares tal que si (a',b') y (a'',b'') son dos pares consecutivos, entonces $0 \leq a'b' < a''b''$. Este un proceso finito que nos lleva hasta el par $\text{mcd}\,(0,d)$. Para ver esto, observemos que la regla 1 y la regla 2 siempre llevan a la regla 3. Al aplicar esta regla al nuevo par (a,b), si $a = b$ nos queda $\text{mcd}\,(0,\text{mín}\{a,b\})$ y terminamos; sino, supongamos que $0 < a < b$, entonces $\dfrac{b-a}{2} < b/2 < b$, es decir, en esta regla 3 cambiamos a y b por $a' = a$ y $b' = \dfrac{b-a}{2} < b$, por tanto el nuevo par (a',b') cumple $a'b' < ab$. El algoritmo termina cuando obtenemos el par $(0,d)$ y entonces $\text{mcd}\,(a,b) = 2^s \cdot \text{mcd}\,(0,d) = 2^s \cdot d$.

1.7.1 Algoritmo e Implementación.

En la primera parte del algoritmo,si a y $b \neq 0$ son pares, se dividen ambos por dos hasta que uno de los dos sea impar.

Luego, mientras $a \neq 0$, si uno es par y el otro impar, aplicamos la regla dos hasta ambos sean impares. Una vez que los dos son impares, aplicamos la regla tres. Las reglas dos y tres se aplican mientras $a = Abs(a-b)/2$ no se anule.

Aquí suponemos que $a, b \in \mathbb{Z},\ a \geq 0, b > 0$. La división por 2, denotada $quo(a,2)$ en el algoritmo, se hace con la división entera usual `a\2`.

Algoritmo 1.2: Algoritmo binario para el mcd

Datos: $a, b \in \mathbb{Z},\ a \geq 0, b > 0$
Salida: $\text{mcd}\,(a,b)$

```
1  g = 1;
2  while a mod 2 = 0 And b mod 2 = 0 do
3  |  a = quo(a,2), b = quo(b,2);
4  |  g = 2g //removiendo potencias de 2
5  while a ≠ 0 do // Ahora, a o b es impar
6
7  |  if a mod 2 = 0 then
8  |  |  a = quo(a,2)
9  |  else if b mod 2 = 0 then
10 |  |  b = quo(b,2)
11 |  else ; // ambos impares
12 |
13 |  |  t = quo(|a − b|,2);
14 |  |  if a ≥ b then ; // reemplazamos máx{a,b} con quo(|a − b|,2)
15 |  |
16 |  |  |  a = t
17 |  |  else
18 |  |  |  b = t
19 |
20 return g · b;
```

Implementación. La implementación es directa. Usamos `a\b` para la división por dos y `a Mod 2=0` como prueba de paridad.

```
Function mcdBinario(u,v) As Long
Dim t As Long,  g As Long, a As Long, b As 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
Wend
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
Wend
mcdBinario=g*b
End Function
```

Prueba de las reglas. La prueba de las reglas es como sigue.

Prueba de la regla 1. Sean $d = \text{mcd}\,(a,b)$ y $d' = \text{mcd}\left(\frac{a}{2},\frac{b}{2}\right)$. 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, $\frac{d}{2} = \frac{a}{2}x + \frac{b}{2} \geq d'$ 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 $d = \text{mcd}\,(a,b)$ y $d' = \text{mcd}\left(\frac{a}{2},b\right)$. 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 $\frac{a}{2} = \frac{k}{2}d$, 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 $d' \leq d$. $\therefore\ d = d'$.

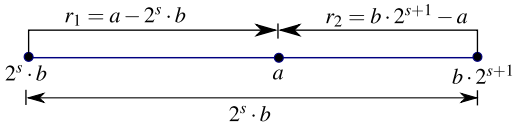
Prueba de la regla 3. Sean $d = \text{mcd}\,(a,b)$ y $d' = \text{mcd}\left(\frac{|a-b|}{2},b\right)$. Como $d|a$ y $d|b$ entonces $d||a-b|$. Como a es impar, d es impar pero $|a-b|$ es par (a y b son impares), luego si $|a-b| = kd$, k debe ser par, así que podemos dividir por dos a ambos lados, $\frac{|a-b|}{2} = \frac{k}{2}d$. 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 $d = d'$.

1.8 Algoritmo LSBGCD (left-shift binary algorithm)

Este algoritmo debe su nombre al hecho de que se hace multiplicación por 2. En representación binaria el efecto de multiplicar por dos es un desplazamiento (en una posición), de la representación binaria original, hacia la izquierda.

En este algoritmo se encuentra $s \in \mathbb{N}$ tal que $2^s \cdot b \leq a \leq b \cdot (2^{s+1})$ y, como en el algoritmo de Euclides con menor resto, se toma r como el menor resto entre $a - 2^s \cdot b$ y $b \cdot 2^{s+1} - a$.



De esta manera tenemos en el primer paso $a = b \cdot 2^{s'} \pm r \implies \text{mcd}\,(a,b) = \text{mcd}\,(b,r)$; donde s' es s o $s+1$ dependiendo que cual resto sea el menor.

Ejemplo 1.2

Sea $a = 55367$ y $b = 28731$. En cada paso tomamos el menor resto $r = \text{mín}\{a - 2^s \cdot b,\ b \cdot 2^{s+1} - a\}$.

$$
\begin{array}{rcl@{\quad\Longrightarrow\quad}l@{\quad}c@{\quad}l}
55367 &=& 28731 \cdot 2^1 - 2095 & \text{mcd}\,(55367,28731) &=& \text{mcd}\,(28731,2095) \\
28731 &=& 2095 \cdot 2^4 - 4789 & & =& \text{mcd}\,(2095,4789) \\
4789 &=& 2095 \cdot 2^1 + 599 & & =& \text{mcd}\,(2095,599) \\
2095 &=& 599 \cdot 2^2 - 301 & & =& \text{mcd}\,(599,301) \\
599 &=& 301 \cdot 2^1 - 3 & & =& \text{mcd}\,(301,3) \\
301 &=& 3 \cdot 2^7 - 83 & & =& \text{mcd}\,(3,83) \\
83 &=& 3 \cdot 2^5 - 13 & & =& \text{mcd}\,(3,13) \\
13 &=& 3 \cdot 2^2 + 1 & & =& \text{mcd}\,(3,1) \\
3 &=& 1 \cdot 2^1 + 1 & & =& \text{mcd}\,(1,1) \\
1 &=& 1 \cdot 2^0 + 0 & & =& \text{mcd}\,(1,0) = 1
\end{array}
$$

Como se observa, la sucesión de restos no es una sucesión estrictamente decreciente, pero cada resto r_i está en un intervalo $[0,d_i]$ y el nuevo resto r_{i+1} está en en un intervalo $[0,d_{i+1}] \subset [0,d_i]$, es decir, cada nuevo r_{i+1} está en un intervalo cada vez más pequeño.

Esto es así pues si $a > b$, $a = b \cdot 2^{s_1} + r_1 \implies 0 \leq r_1 < b \cdot 2^{s_1}$. Si $r_1 > 0$. En el siguiente paso tendríamos dos casos posibles,

a.) si $b > r_1$ entonces $b = r_1 \cdot 2^{s_2} + r_2 \implies 0 \leq r_2 < r_1 \cdot 2^{s_2} < b \cdot 2^{s_1}$. Si $r_2 > 0$, la última desigualdad se cumple pues si $r_1 \cdot 2^{s_2} \geq b \cdot 2^{s_1}$ entonces $b = r_1 \cdot 2^{s_2} + r_2 \geq b \cdot 2^{s_1} + r_2$ $(\Rightarrow\Leftarrow)$.

b.) si $r_1 > b$ entonces $r_1 = b \cdot 2^{s_2} + r_2 \implies 0 \leq r_2 < b \cdot 2^{s_2} < b \cdot 2^{s_1}$. Si $r_2 > 0$, la última desigualdad se cumple pues si $b \cdot 2^{s_2} \geq b \cdot 2^{s_1}$ entonces $r_1 = b \cdot 2^{s_2} + r_2 \geq b \cdot 2^{s_1} + r_2$, por tanto $a = b \cdot 2^{s_1} + r_1 \geq b \cdot 2^{s_1} + b \cdot 2^{s_1} + r_2 > b \cdot 2^{s_1+1}$ en contradicción con la escogencia de s_1 (ver figura 1.8).

Si llamamos a los nuevos dividendos $d_1 = b, d_2, d_3$, etc., entonces en el $n-$ésimo resto tendríamos

$$0 \leq r_n < d_n \cdot 2^{s_n} < d_{n-1} \cdot 2^{s_{n-1}} < \ldots < b \cdot 2^{s_1}$$

Es decir, cada nuevo resto r_i está en un intervalo $[0, d_i \cdot 2^{s_i}]$ cada vez más pequeño. Como el número de intervalos es finito, la sucesión de restos es finita y por tanto en algún momento $r_{n+1} = 0$.

Este algoritmo no es tan rápido como el algoritmo binario, pero su versión extendida si es más eficiente que la versión extendida de Euclides y la versión extendida del algoritmo binario.

1.8.1 Algoritmo e Implementación.

El algoritmo es como sigue,

Algoritmo 1.3: Algoritmo LSBGCD

Datos: $a, b \in \mathbb{Z}^+$ y $a > b$
Salida: $\text{mcd}\,(a,b)$

while $b \neq 0$ **do**
  Calcule s tal que $b \cdot 2^s \leq a < 2^{s+1}b$;
  $t = \text{Mín}\{a - b \cdot 2^s, b \cdot 2^{s+1} - a\}$;
  $a = b$; $b = t$;
  if $a < b$ **then**
    Intercambiar(a,b)
return a

Implementación. Necesitamos una función `Min(a,b)`. Aunque podemos usar la función `Mín` de Calc (vía `createUnoService("com.sun.star.sheet.FunctionAccess")`), aquí no vamos a usar esta posibilidad, más bien usamos una función sencilla.

```
Function Min (a,b) As Long
Dim m As Long
 m=a
 If a>b Then
    m=b
 End If
Min= m
End Function

Function LSBMCD(u,v) As Long
Dim a As Long, b As Long, t As Long, aux As Long
Dim s As Integer
a=Abs(u) : b=Abs(v)   rem debe ser a>b
While b<>0
    s=0
```

```
    While b*2^s <=a
       s=s+1
    Wend
    s = s-1
    t=Min(a-b*2^s,b*2^(s+1)-a)
    a = b : b = t
    If a<b Then
       aux=a
       a=b
       b=aux
    End If
Wend
LSBMCD = a
End Function
```

Bibliografía

[1] M. Atallah, M. Blanton (2010). *Algorithms and theory of computation handbook. General concepts and techniques.* Chapman & Hall. CRC applied algorithms and data structures series. 2nd ed.

[2] H. Cohen (1993). *A Course in Computational Algebraic Number Theory.* Springer-Verlag, 1993.

[3] E. Bach, J. Shallit (1996). *Algorithmic Number Theory, Vol. 1: Efficient Algorithms.* Cambridge, MA: MIT Press, 1996.

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

[5] D. Knuth (1981). *The Art of Computer Programming. Volume 1: Fundamental Algorithms.* Addison-Wesley. 2nd ed.

[6] D. Knuth (1981). *The Art of Computer Programming. Volume 2: Seminumerical Algorithms.* Addison-Wesley. 2nd ed.

[7] G. Norton (1987). A shift-remainder GCD algorithm. Proceedings of the 5th international conference, AAECC-5 on Applied Algebra, Algebraic Algorithms and Error-Correcting Codes, p.350-356, 1987.

[8] J. Shallit, J. Sorenson (1994). Analysis of a Left-Shift Binary GCD Algorithm. J. Symbolic Computation (1994) 17, 487-511

[9] A. Stepanov (2007). Notes on Programming. En http://www.stepanovpapers.com

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

[11] A.Weilert (2000). $(1+i)-$ary GCD Computation in $Z[i]$ as an Analogue to the Binary GCD Algorithm.J. Symbolic Computation (2000) 30, 605-617.