
L'algorithme binaire récursif de calcul de pgcd

Damien STEHLÉ

Rocquencourt, 15-03-2004

Travail en commun avec Paul Zimmermann

<http://www.loria.fr/~stehle/>



Sommaire

- 1. Quelques rappels.
- 2. Les divisions standard et binaire.
- 3. L'algorithme de Knuth et l'algorithme binaire récursif.
- 4. Analyse de complexité et considérations pratiques.

Arithmétique des entiers

- Nombres représentés en binaire, avec au plus n chiffres.
- Comparaison, addition, soustraction (naïves) en $O(n)$.
- Multiplication en $M(n) = O(n \log n \log \log n)$, à base de FFT.
- Division (naïve) d'un nombre de taille m par un nombre de taille $n < m$ en $O((m - n) \cdot m)$.
- Division d'un nombre de taille $2n$ par un nombre de taille n en $O(M(n))$, par itération de Newton et multiplication rapide.

Le problème du pgcd

- But : étant donnés a et b , calculer $g = \text{pgcd}(a, b)$.
- Mieux : calculer aussi A et B tels que $Aa + Bb = g$ (pgcd étendu).
- Modèle de complexité : opérations élémentaires sur les bits.
- Applications : calculs sur les rationnels, inversions modulaires,
- Dans la suite, $a, b \in \mathbb{Z}$ et $n = \max(\text{bits}(a), \text{bits}(b))$.

Petite histoire du pgcd

- ≈ -300 , Euclide : algo. d'Euclide, en temps **quadratique** $O(n^2)$
- 1938, Lehmer : accélération de l'algorithme d'Euclide (les bits forts suffisent pour calculer les premiers quotients).
- 1970, Schönhage et Strassen : multiplication rapide en temps
$$M(n) = O(n \log n \log \log n).$$
- 1970, Knuth : algorithme de Lehmer récursif, basé sur la multiplication rapide de Schönhage-Strassen. Complexité **quasi-linéaire** $O(M(n) \log^4 n)$.
- 1971, Schönhage : analyse précise de l'algorithme de Knuth. Complexité **quasi-linéaire** $O(M(n) \log n)$.

Le calcul de pgcd en pratique

- Algo. de Knuth efficace seulement pour de très grands nombres
- Pgcd sous-quadratique rarement implanté (Mathematica, Maple)
- Deux techniques pour accélérer facilement l’algorithme d’Euclide
 - 1) Lehmer : “Tant que c’est facile, n’utiliser que les bits forts”.
 - 2) Binaire : “Shifter et additionner sont plus rapides que diviser”.

Les résultats

- Incompatibilité des deux idées précédentes :
bits forts dans 1) \leftrightarrow bits faibles dans 2).
- Résultats :
 1. Définition d'une division (GB) par les bits faibles ayant la propriété de Lehmer (pour les bits faibles).
 2. Transposition de l'algo. de Knuth pour cette division.
 3. Au passage, l'algo. de Knuth garde la même complexité mais devient nettement plus simple.

La division Euclidienne “standard”

- Étant donnés $a, b > 0$, il existe un unique couple (q, r) t.q. :

$$a = bq + r, \quad 0 \leq r < b, \quad q \geq 0.$$

- Exemple : $a = 157 = 10011101$, $b = 59 = 111011$.
- $(q, r) = (2, 39) = (10, 100111)$.
- Remarque : $10011 = 19$, $111 = 7$ donnent le même quotient.
- Calcul de q en $O(M(n))$ par itération de Newton (théorie).
- Calcul de q en $O(n)$ car $q = O(1)$ (pratique).

L'algorithme d'Euclide "standard"

Input : $a, b \in \mathbb{Z}$ avec $a > b$.

Output : $g = \text{pgcd}(a, b)$, et M t.q. $M \begin{pmatrix} a \\ b \end{pmatrix} = \begin{pmatrix} g \\ 0 \end{pmatrix}$.

1. Si $b = 0$, renvoyer a et I_2 .

2. $q, r := \lfloor a/b \rfloor$.

3. $g, M' := \text{Euclide}(b, r)$.

4. Renvoyer g et $M = M' \begin{pmatrix} 0 & 1 \\ 1 & -q \end{pmatrix}$.

La division binaire (1)

- C’est le “miroir” de la division Euclidienne.
- Si $a \neq 0$, $\nu_2(a)$ est le plus grand k t.q. $2^k \mid a$. Et $\nu_2(0) = \infty$.
- Pour a, b avec $\nu_2(a) < \nu_2(b)$, il existe un unique couple (q, r) t.

$$a = \frac{b}{2^{\nu_2(b) - \nu_2(a)}}q + r, \quad \nu_2(r) > \nu_2(b), \quad |q| < 2^{\nu_2(b) - \nu_2(a)}.$$

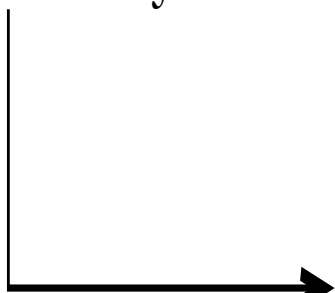
- Exemple : $a = 157 = 10011101$, $b = 4 * 59 = 11101100$.

$$\nu_2(b) - \nu_2(a) = 2$$

$$q = -1, \quad r = a + \frac{b}{4} = 11011000 (= 216)$$

$$\nu_2(r) = 3 > \nu_2(b).$$

La division binaire (2)

a	100010010111011		100010010111011
b	$ \begin{array}{r} 111101110111000 \\ \underline{\hspace{1.5em}} \\ 3 \end{array} $	SHIFT(3)	$ \begin{array}{r} + 111101110111 \\ \hline 101010000110010 \\ 111101110111000 \\ \text{SHIFT(2)} \\ + 1111011101110 \\ \hline 111001100100000 \\ \underline{\hspace{1.5em}} \\ 5 \end{array} $
			
	$q = 2^0 + 2^1 = 3$		

La division binaire (3)

- Le quotient binaire de a par b est exactement

$$\frac{a}{2^{\nu(a)}} \left(\frac{b}{2^{\nu(b)}} \right)^{-1} \text{ cmod } 2^{\nu(b) - \nu(a) + 1}.$$

- Calcul par itération de Newton en temps $O(M(n))$ (théorie).
- Calcul de q en $O(n)$ car $q = O(1)$ (pratique).
- Seuls les $\nu(b) - \nu(a) + 1$ derniers bits non nuls de a et b impor

L'algorithme d'Euclide binaire

Input : $a, b \in \mathbb{Z}$ avec $0 = \nu(a) < \nu(b)$.

Output : $g = \text{pgcd}(a, b)$, et M t.q. $M \begin{pmatrix} a \\ b \end{pmatrix} = \begin{pmatrix} g \\ 0 \end{pmatrix}$.

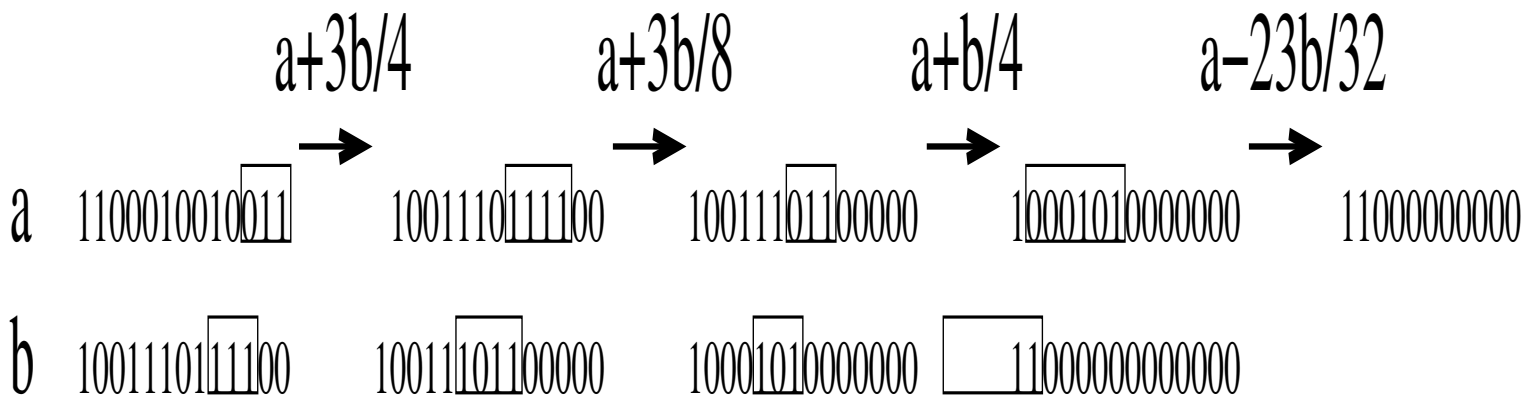
1. Si $b = 0$, renvoyer a et I_2 .

2. $q, r := \mathbf{Div-Bin}(a, b)$.

3. $g, M := \mathbf{Gcd-Bin}(\frac{b}{2^{\nu(b)}}, \frac{r}{2^{\nu(b)}})$.

4. Renvoyer g et $\frac{1}{2^{\nu(b)}} M \begin{pmatrix} 0 & 1 \\ 1 & \frac{q}{2^{\nu(b)}} \end{pmatrix}$.

Un exemple



pgcd=3

$$\begin{bmatrix} 3 \cdot 2^{12} \\ 0 \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 1 & -23/32 \end{bmatrix} \begin{bmatrix} 0 & 1 \\ 1 & 1/4 \end{bmatrix} \begin{bmatrix} 0 & 1 \\ 1 & 3/8 \end{bmatrix} \begin{bmatrix} 0 & 1 \\ 1 & 3/4 \end{bmatrix} \cdot \begin{bmatrix} a \\ b \end{bmatrix}$$

L'algorithme de Knuth (1)

- Avec la division “standard”.
- Deux remarques préliminaires :
 - 1) Stocker les restes : $O(n^2)$, stocker les quotients : $O(n)$.
 - 2) Les k bits les plus significatifs permettent de calculer les $\approx k$ premiers bits de la suite des quotients.
- Algorithme de Knuth : utilisation récursive de 2) pour calculer tous les quotients et très peu de restes.
- **PROBLÈME** : 2) est faux si on enlève $\approx k$.

Réparation des quotients

– $a' = MSB_k(a)$ et $b' = MSB_k(b)$.

$$a', b' \longrightarrow r'_0, \dots, r'_{i+1}, q'_1, \dots, q'_i \qquad a, b \longrightarrow r_0, \dots, r_{j+1}, q_1, \dots, q_j.$$

i et j sont tels que :

$$\text{bits}(r'_i) \geq k/2 > \text{bits}(r'_{i+1}) \quad \text{et} \quad \text{bits}(r_j) \geq \text{bits}(a) - k/2 > \text{bits}(r_{j+1}).$$

Alors q'_1, \dots, q'_i “ressemblent” à q_1, \dots, q_j .

– MAIS il se peut que l'on ait :

1) $j = i$, $q_1, \dots, q_{i-1} = q'_1, \dots, q'_{i-1}$ mais $q'_i \neq q_i$.

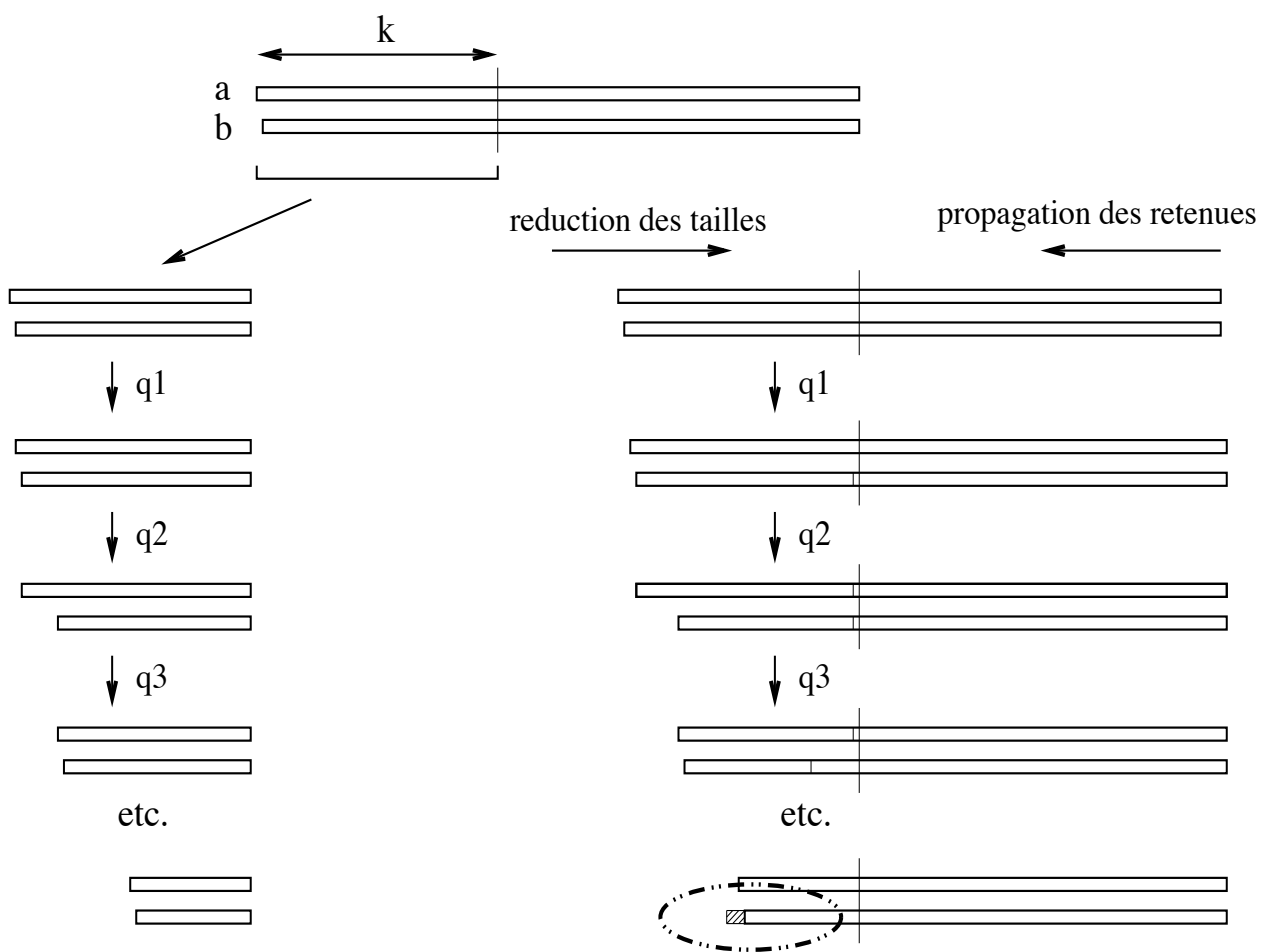
2) $j = i - O(1)$ et $q_1, \dots, q_j = q'_1, \dots, q'_j$.

3) $j = i + O(1)$ et $q_1, \dots, q_i = q'_1, \dots, q'_i$.

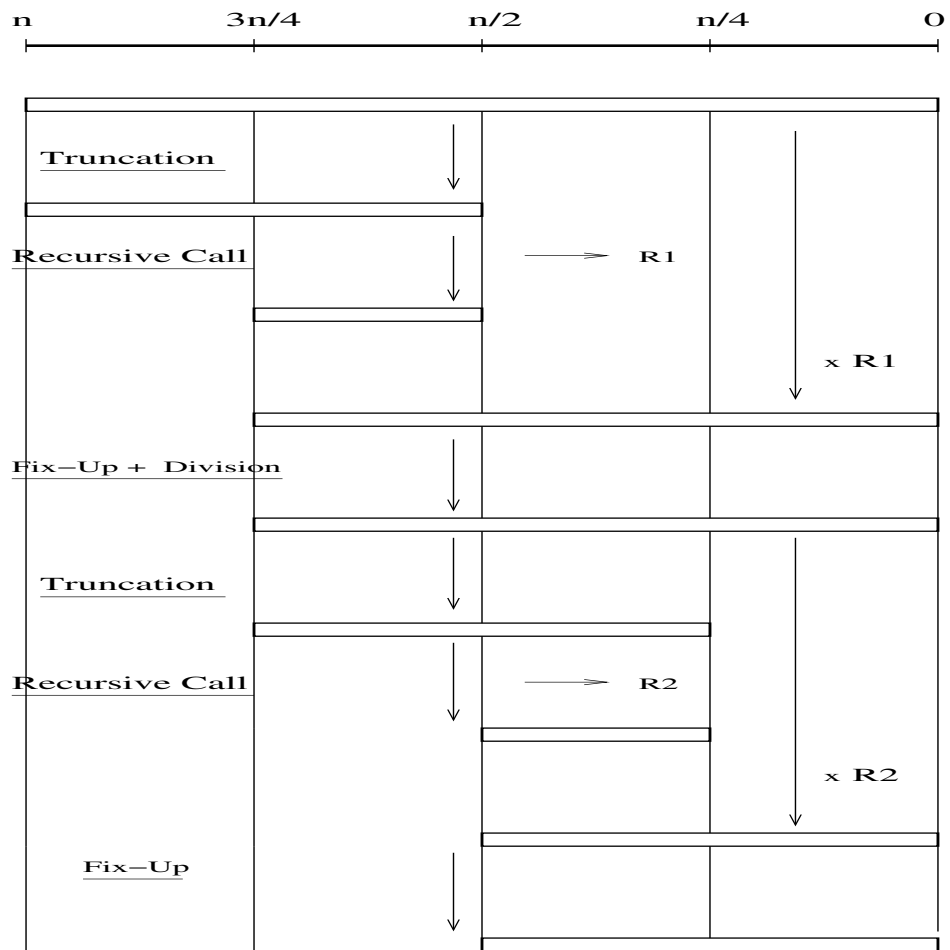
4) Des mélanges entre 1), 2) et 3).

\implies Il existe une procédure de “réparation” des quotients, qui coûte $O(M)$ mais elle complique notablement l'algorithme.

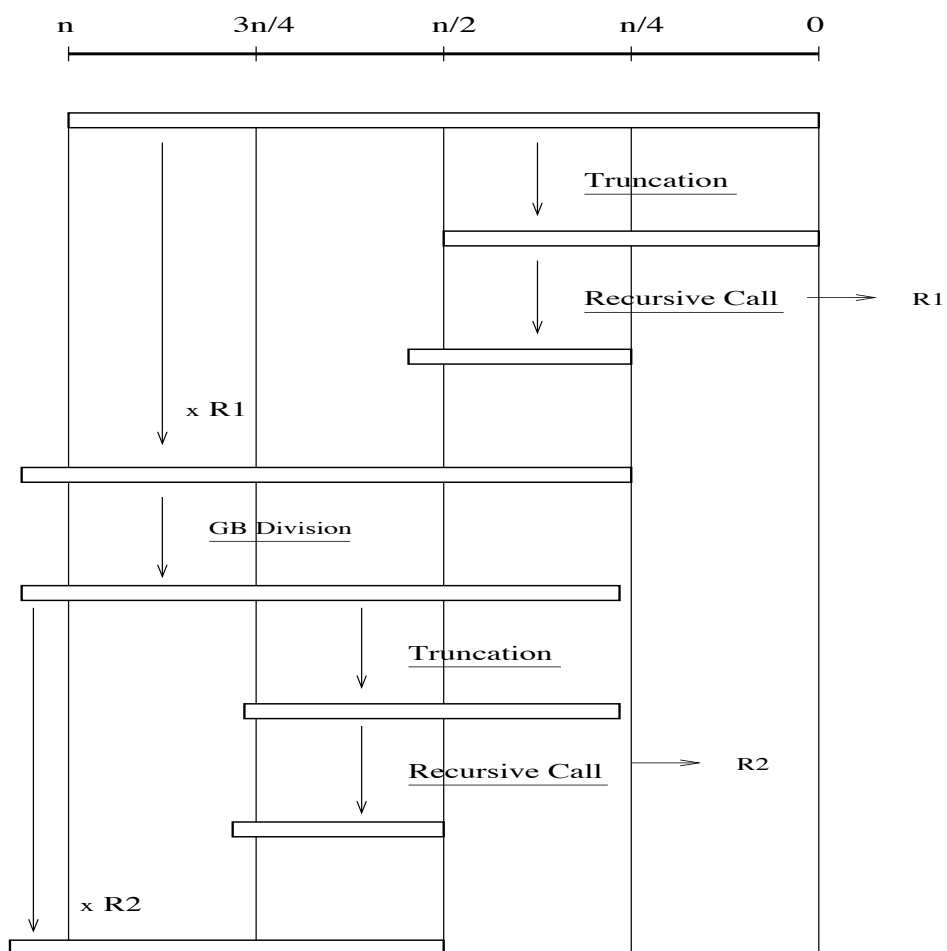
Origine du problème



L'algorithme de Knuth (2)



L'algorithme récursif binaire (1)



L'algorithme récursif binaire (2)

- Droite \leftrightarrow Gauche.
 - Division GB \leftrightarrow Division “standard”.
 - Les idées restent les mêmes sauf que :
 - 1) les bits forts sont remplacés par les bits faibles.
 - 2) les quotients calculés lors des appels récursifs sont corrects.
- \implies Il n’y a plus de procédure de réparation (Fixup).

L'algorithme récursif binaire (3)

Input : a, b t.q. $0 = \nu(a) < \nu(b)$.

Output : j et R t.q. $\begin{pmatrix} c \\ d \end{pmatrix} = 2^{-j} R \begin{pmatrix} a \\ b \end{pmatrix}$, et c, d sont les

deux restes successifs de la suite des restes de a et b t.q.

$$\nu(c) \leq \text{bits}(a)/2 < \nu(d).$$

1) $k := \lfloor \text{bits}(a)/2 \rfloor$.

2) Si $\nu(b) > k$, renvoyer $0, \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$.

3) $d := \lfloor k/2 \rfloor$.

4) $a := a_1 2^{2d+1} + a_0, b := b_1 2^{2d+1} + b_0$ avec $0 \leq a_0, b_0 < 2^{2d}$.

5) $j_1, R_1 = \mathbf{Half-GB-gcd}(a_0, b_0)$.

$$6) \begin{pmatrix} a' \\ b' \end{pmatrix} := 2^{2d+1-j_1} R_1 \begin{pmatrix} a \\ b \end{pmatrix}.$$

7) Si $\nu(b') > k$, renvoyer j_1, R_1 .

8) $(q, r) := \mathbf{Div-Bin}(a', b')$, $j_0 := \nu(b')$.

9) $d := k - \nu(b')$.

10) $\frac{b'}{2^{\nu(b')}} := b'_1 2^{2d+1} + b'_0$, $\frac{r}{2^{\nu(b')}} := r_1 2^{2d+1} + r_0$
avec $0 \leq b'_0, r_0 < 2^{2d+1}$.

11) $j_2, R_2 := \mathbf{Half-GB-gcd}(b'_0, r_0)$.

$$12) \begin{pmatrix} c \\ d \end{pmatrix} := 2^{2d+1-j_2} R_2 \begin{pmatrix} b' \\ r \end{pmatrix}.$$

13) Renvoyer $j_1 + j_0 + j_2$, $R_2 \begin{pmatrix} 0 & 2^{j_0} \\ 2^{j_0} & q \end{pmatrix} R_1$.

Un exemple

	q	n	Half-Gcd
10011111011101101101			10011111011101101101
11110111100010011100	-3	2	11110111100010011100
-11010001100001000	-1	1	
<u>100000100101000100000</u>	1	2	
<u>1001101111100000000</u>	1	2	
100001110011000000000	3	2	
<u>11110001110000000000</u>	-1	1	
<u>100101011000000000000</u>	-1	1	
101001110000000000000	-1	1	
100001000000000000000	1	1	
110010000000000000000	-1	2	
100000000000000000000	-1	1	
110000000000000000000	-3	2	
-100000000000000000000	-1	1	
100000000000000000000	1	1	
0			

Half-Gcd	
10011111011101101101	
11110111100010011100	
<u>1er appel recursif</u>	
$\swarrow \rightarrow$	$2^{-5} \begin{bmatrix} -16 & 44 \\ 28 & -13 \end{bmatrix}$
<u>1ere multiplication</u>	
100000100101000100000	
100110111100000000000	
100001100110000000000	
<u>2e appel recursif</u>	
$\swarrow \rightarrow$	$2^{-3} \begin{bmatrix} 8 & 6 \\ 4 & 11 \end{bmatrix}$
<u>2e multiplication</u>	
111100011100000000000	
100101011000000000000	11

Un exemple, suite

	q	n	
10011111011101101101			Fast-Gcd
11110111100010011100	-3	2	10011111011101101101
-11010001100001000	-1	1	11110111100010011100
100000100101000100000	1	2	Half-Gcd
100110111110000000	1	2	$\swarrow \rightarrow 2^{-10} \begin{bmatrix} 680 & 562 \\ 628 & 1023 \end{bmatrix}$
10000111001100000000	3	2	11110001110000000000
<u>11110001110000000000</u>	-1	1	100101011 <u>0000000000</u> ₁₁
<u>10010101100000000000</u>	-1	1	Half-Gcd
10100111000000000000	-1	1	$\swarrow \rightarrow 2^{-5} \begin{bmatrix} 24 & 4 \\ -22 & 39 \end{bmatrix}$
10000100000000000000	1	1	11001000000000000000
<u>11001000000000000000</u>	-1	2	10000000000000000000 ₁₆
<u>10000000000000000000</u>	-1	1	etc.
11000000000000000000	-3	2	
-10000000000000000000	-1	1	
10000000000000000000	1	1	
0			

Pourquoi les quotients sont-ils corrects ?

Résultat 1 : Soient a, b, a' et b' tels que :

$$a' = a \bmod 2^l \quad \text{et} \quad b' = b \bmod 2^l,$$

avec $l \geq 2\nu(b) + 1$. Supposons que $0 = \nu(a) < \nu(b)$.

Soient $(q, r) = \mathbf{Div-Bin}(a, b)$ et $(q', r') = \mathbf{Div-Bin}(a', b')$.

Alors : $q = q'$ et $r = r' \bmod 2^{l-\nu(b)}$.

Pourquoi les quotients sont-ils corrects ?

Résultat 2 : Soient a, b, a' et b' tels que :

$$a' = a \bmod 2^{2k+1} \quad \text{et} \quad b' = b \bmod 2^{2k+1},$$

avec $k \geq 0$. Supposons que $0 = \nu(a) < \nu(b)$.

Soient $r_0 = a, r_1 = b, r_2, \dots$ la suite des restes de (a, b) ,
et q_1, q_2, \dots celle des quotients.

Soient $r'_0 = a', r'_1 = b', r'_2, \dots$ la suite des restes de (a', b') ,
et q'_1, q'_2, \dots celle des quotients.

Si r_{i+1} est le premier t.q. $\nu(r_{i+1}) > k$, alors pour tout $j \leq i$:

$$q_j = q'_j \quad \text{et} \quad r_{j+1} = r'_{j+1} \bmod 2^{2k+1-\nu(r_j)}.$$

Taille des matrices renvoyées

- Pire cas ($n \geq 8$) : $\forall i, q_i = \pm 1$ et $\nu(r_{i+1}) = \nu(r_i) + 1$.
- Similarité avec le pire cas pour la division standard (Fibonacci).
- Soient a et b avec $0 = \nu(a) < \nu(b)$.

Si M est la matrice des quotients qui “fait gagner n bits”,
les coefficients de M ont au plus $n(1 + \log \frac{1+\sqrt{17}}{4}) \approx 1.36 \cdot n$

- \implies Au plus $n / \log \frac{\sqrt{17}-1}{2} \approx 1.56 \cdot n$ quotients
($1.44 \cdot n$ pour la division “standard”).

Complexité asymptotique de l'algorithme

- $H_n = 2H_{\frac{n}{2}+1} + O(M(n))$.
- $F_n = H_n + F_{\alpha n} + O(M(n))$, avec $\alpha = \frac{1}{2}(1 + \log \frac{1+\sqrt{17}}{4}) < 1$.
- D'où : $F_n, H_n = O(M(n) \log n)$.
- Estimation heuristique du $O()$:
 - 1) tous les quotients sont petits.
 - 2) une matrice qui fait gagner k bits a des coefficients de taille

$$\implies G_n \approx \frac{17}{4} M(n) \log n.$$

Optimisations pratiques

- Calculer plusieurs petits quotients à la fois pour “remplir” un mot machine, en n'utilisant que deux mots machines des restes.
- Dans **Half-Gcd**, renvoyer les restes courants en plus de la matrice.
- Lors des appels à **Half-Gcd** au sommet de la récursion, ne renvoyer que les restes courants.
- Pour Karatsuba et Toom-Cook, changer **Half-Gcd** pour annuler γn bits au lieu de $n/2$ (avec γ à optimiser).

Benchmarks

“ F_n ” : Test avec les Fibonacci consécutifs F_{n-1} et F_n .

“ G_n ” : Idem pour un pire cas de la division binaire.

type, n	Magma V2.10-12	Mathematica 5.0	Fast-GB-gcd (GNU I
$F_n, 10^6$	2.89	2.11	0.70
$F_n, 2 \cdot 10^6$	7.74	5.46	1.91
$F_n, 5 \cdot 10^6$	23.3	17.53	6.74
$F_n, 10^7$	59.1	43.59	17.34
$G_n, 5 \cdot 10^5$	2.78	2.06	0.71
$G_n, 10^6$	7.99	5.30	1.94