Calcul du pgcd de deux entiers

1 Algorithme d'euclide

On se propose d'étudier l'efficacité respective de différents algorithmes pour le calcul du p.g.c.d de deux entiers a, b tels que $0 < b < a$.

1. Ecrire une procédure `pgcd` qui prend en entrée a, b calcule le pgcd de a et de b par l'algorithme d'Euclide (avec reste positif).
2. Ecrire une procédure `nb` qui prend en entrée a, b et calcule le nombre d'étapes nécessaires pour le calcul du pgcd de a et de b par l'algorithme d'Euclide (avec reste positif)
3. En choisissant au hasard le couple (a, b) dans l'intervalle $[2, 100]$, représenter la distribution du nombre d'étapes nécessaires pour le calcul du pgcd de a et de b.

Proposition 1 *Soit (F_n) la suite de Fibonacci. Si n est le nombre d'étapes de l'algorithme d'Euclide (avec reste positif) pour le calcul du pgcd de a et de b, alors $a \geqslant F_{n+1}$ et $b \geqslant F_n$.*

Preuve Soit n est le nombre d'étapes de l'algorithme d'Euclide (avec reste positif) pour le calcul du pgcd de a et de b. Ces n étapes s'écrivent :

$$\begin{array}{lcll}
a & = & bq_1 + r_1 & 0 < r_1 < b \quad (1) \\
b & = & r_1 q_2 + r_2 & 0 < r_2 < r_1 \quad (2) \\
\vdots & & & \\
r_{i-2} & = & r_{i-1} q_i + r_i & 0 < r_i < r_{i-1} \quad (i) \\
\vdots & & & \\
r_{n-3} & = & r_{n-2} q_{n-1} + r_{n-1} & 0 < r_{n-1} < r_{n-2} \quad (n-1) \\
r_{n-2} & = & r_{n-1} q_n & (n)
\end{array}$$

On remarque que $q_n \geqslant 2$, donc $r_{n-2} \geqslant 2 = F_3$, puis par récurrence descendante : $r_{i-2} = r_{i-1} q_i + r_i \geqslant r_{i-1} + r_i \geqslant F_{n-i+1} + F_{n-i} = F_{n-i+2}$ par définition de la suite de Fibonacci. Donc $a = r_{-1} \geqslant F_{n+2}$ et $b = r_0 \geqslant F_{n+1}$. □

Proposition 2 *Soit (F_n) la suite de Fibonacci. Le nombre d'étapes de l'algorithme d'Euclide (avec reste positif) pour le calcul du pgcd de F_{n+1} et F_{n+2} est n.*

Preuve Pour $i > 0$, $F_i > 0$ donc pour $i > 1$, $F_{i+1} > F_i$. Les étapes de l'algorithme d'Euclide (avec reste positif) pour le calcul du pgcd de F_{n+1} et F_n

sont :

$$\begin{array}{lcll} F_{n+1} & = & F_n + F_{n-1} & (1) \\ F_n & = & F_{n-1} + F_{n-2} & (2) \\ \vdots & & & \\ F_3 & = & 2F_2 & (n-1) \end{array}$$

□

Théorème 1 (Lamé) *Le nombre d'étapes de l'algorithme d'Euclide (avec reste positif) pour le calcul du pgcd de a et de b est majoré par cinq fois le nombre de chiffres de l'écriture en base 10 de b.*

Une analyse plus fine : si n est un entier naturel, on note $l(n)$ le nombre de chiffres de l'écriture en base 2 de n.

$$l(n) = \begin{cases} E(\log_2(a)) & \text{si } a > 0 \\ 1 & \text{si } a = 0 \end{cases}$$

On admet que l vérifie les propriétés suivantes :

$a + b$	$O(l(a) + l(b))$
ab	$O(l(a)l(b))$
$E(a/b)$	$O(l(a)l(E(a/b)))$
$a \bmod b$	$O(l(a)l(E(a/b)))$

Corollaire 1 *Le coût de l'algorithme d'Euclide est $O(l(a)l(b))$.*

Preuve Considérons les n étapes de l'algorithme :

$$\begin{array}{lcll} a & = & r_1 \bmod b & (1) \\ b & = & r_2 \bmod r_1 & (2) \\ \vdots & & & \\ r_{i-2} & = & r_i \bmod r_{i-1} & (i) \\ \vdots & & & \\ r_{n-2} & = & 0 \bmod r_{n-1} & (n) \end{array}$$

Le coût de la i-ème étape est $O(l(r_{i-2})l(E(r_{i-1}/r_{i-2}))) = O(l(r_{i-2})l(q_{i+2})))$ où $r_{i-2} = q_i r_{i-1} + r_i$. Donc le coût de l'algorithme est :

$$\sum_{i=-1}^{n-2} O(l(r_i)l(q_{i+2}))) \leqslant O(l(b))(\sum_{i=-1}^{n-2} l(q_{i+2})) = O(l(b))l(\prod_{i=1}^{n} q_i)) \leqslant O(l(b))O(l(a)).$$

En effet, $q_i r_{i-1} \leqslant r_{i-2}$ implique $(\prod_{i=1}^{n} q_i) \leqslant r_{-1}/r_{n-1} \leqslant r_{-1} = a$. □

4. On remplace dans l'algorithme d'Euclide le reste positif par le plus petit reste en valeur absolue. Ecrire une procédure `nb2` qui prend en entrée a, b et calcule le nombre d'étapes nécessaires pour le calcul du pgcd de a et de b par l'algorithme d'Euclide (avec plus petit reste).

5. En listant les couples (a, b), $b < a$, dans $[2, 100] \times [2, 100]$, représenter sur un même graphique les distributions du nombre d'étapes nécessaires pour le calcul du pgcd de a et de b avec les deux restes possibles. Que constatez-vous ?

6. Quelle suite introduire pour borner les entiers dont ce calcul de pgcd nécessite n étapes ? Obtient-on un équivalent du théorème de Lamé ?

On considère désormais l'algorithme suivant (dit algorithme binaire):

```
variables : u, v, w, e
u := a
v := b
e := 0
tant que u mod 2 = 0 et v mod 2 = 0   faire
                                      e := e + 1
                                      u := u/2
                                      v := v/2
fin faire
tant que (u > 0)                      faire
                                      tant que u mod 2 = 0
                                          u := u/2
                                      tant que v mod 2 = 0
                                          v := v/2
                                      si u < v alors
                                          w := v, v := u, u := w
                                      fin si
                                      u := u − v
fin faire
retourner 2^e v
```

7. Démontrer que cet algorithme calcule le p.g.c.d de a et de b.

8. Démontrer que le coût de cet algorithme est $O(max(l(a), l(b))^2)$.

9. Comment le comparer aux précédents ?

2 Algorithme d'Euclide étendu

On se propose d'étudier différents algorithmes pour le calcul de coefficients s, t tels que $sa + tb = d$ où d est le p.g.c.d de deux entiers a, b.

```
variables : u, v, q, r, s, t, s_1, t_1, s_2, t_2
u := a
v := b
s_1 := 1
t_1 := 0
s_2 := 0
t_2 := 1
r := u mod v
tant que (r > 0)                    faire :
                                    q := (u − r)/v
                                    u := v
                                    v := r
                                    s := s_1, t := t_1 , s_1 := s_2, t_1 := t_2
                                    s_2 := s − s_1 q
                                    t_2 := t − t_1 q
                                    r := u mod v
                                    fin faire
retourner (v, s_2, t_2)
```

10. Soit n le nombre d'étapes de l'algorithme d'Euclide pour le calcul du pgcd de a et de b. Ces n étapes s'écrivent :

a				$s_{-1} = 1$	$t_{-1} = 0$	(-1)
b				$s_0 = 0$	$t_0 = 1$	(0)
a	$=$	$bq_1 + r_1$	$0 < r_1 < b$	$s_1 = s_{-1} - q_1 s_0$	$t_1 = t_{-1} - q_1 t_0$	(1)
b	$=$	$r_1 q_2 + r_2$	$0 < r_2 < r_1$	$s_2 = s_0 - q_2 s_1$	$t_2 = t_0 - q_2 t_1$	(2)
$\vdots$						
r_{i-2}	$=$	$r_{i-1} q_i + r_i$	$0 < r_i < r_{i-1}$	$s_i = s_{i-2} - q_i s_{i-1}$	$t_i = t_{i-2} - q_i t_{i-1}$	(i)
$\vdots$						
r_{n-3}	$=$	$r_{n-2} q_{n-1} + r_{n-1}$	$0 < r_{n-1} < r_{n-2}$	$s_{n-1} = \cdots$	$t_{n-1} = \cdots$	$(n-1)$
r_{n-2}	$=$	$r_{n-1} q_n$				(n)

Démontrer que $s_i a + t_i b = r_i$, $s_i t_{i-1} - s_{i-1} t_i = (-1)^{i-1}$, $pgcd(s_i, t_i) = 1$, $\forall i$. En déduire une preuve de l'algorithme.

11. Démontrer que s_i et s_{i+1} sont de signes opposés et que la suite $(|s_i|)_i$ est croissante; de même, t_i et t_{i+1} sont de signes opposés et la suite $(|t_i|)_i$ est croissante. Démontrer aussi $r_{i-1}|s_i| \leqslant b$ et $r_{i-1}|t_i| \leqslant a$. En déduire que $|s_{n-1}| \leqslant b/2$ et $|t_{n-1}| \leqslant a/2$ (ils sont les plus petits possibles).

Remarque : À l'étape de l'algorithme, l'obtention de s_i et t_i nécessite d'effectuer quelques additions et multiplications en plus du calcul de q_i et r_i. Le coût total de cet algorithme est donc majoré par un O du cout de l'algorithme d'Euclide, soit $O(max(l(a), l(b))^2)$.

```
variables : u, v, w, e, s_1, t_1, s_2, t_2, a_2, b_2
u := a
v := b
e := 0
s_1 := 1
t_1 := 0
s_2 := 0
t_2 := 1
tant que (u > 0) et u mod 2 = 0 et v mod 2 = 0 faire
                                                    e := e + 1
                                                    u := u/2
                                                    v := v/2
                                              fin faire
a_2 := u
b_2 := v
tant que (u > 0) faire :
            tant que u mod 2 = 0 faire :
                                    u := u/2
                                    si s_1 mod 2 = 0 et t_1 mod 2 = 0 faire :
                                                                        s_1 := s_1/2
                                                                        t_1 := t_1/2
                                                                        fin faire
                                    sinon  faire :
                                                s_1 := (s_1 + b_2)/2
                                                t_1 := (t_1 − a_2)/2
                                                fin faire  fin si
            tant que v mod 2 = 0 faire :
                                    v := v/2
                                    si s_2 mod 2 = 0 et t_2 mod 2 = 0 faire :
                                                                        s_2 := s_2/2
                                                                        t_2 := t_2/2
                                                                        fin faire
                                    sinon  faire :
                                                s_2 := (s_2 + b_2)/2
                                                t_2 := (t_2 − a_2)/2
                                                fin faire  fin si
            si u < v alors w := v, v := u, u := w
                           w := s_1, s_1 := s_2, s_2 := w
                           w := t_1, t_1 := t_2, t_2 := w
            fin si
u := u − v
s_1 := s_1 − s_2
t_1 := t_1 − t_2
fin faire
retourner 2^e v, s_2, t_2
```

On appelle cet algorithme l'algorithme binaire étendu.

12 Démontrer que cet algorithme retourne (d, x, y) où d est le p.g.c.d. de a et b et x, y vérifient une relation de Bezout $xa + yb = d$. Comparer sur des exemples la taille de x et y avec ceux obtenus par l'algorithme d'Euclide étendu.

13 En utilisant un des deux algorithmes ci-dessus, en écrire un qui prend en entrée k entiers $a_1, ..., a_k$ non tous nuls, et retourne $(d, x_1, x_2, ..., x_k)$ où d est le p.g.c.d de $a_1, ..., a_k$ et $(x_1, x_2, ..., x_k)$ vérifient une relation de Bezout $\sum x_i a_i = d$.

14 Ecrire un algorithme qui prend en entrée deux entiers m et n et retourne un inverse de m modulo n, si ceux-ci sont premiers entre eux, un message d'avertissement sinon.

15 Ecrire un algorithme qui prend en entrée des entiers $m_1, ..., m_k$ et n et retourne les inverses de $m_1, ..., m_k$ modulo n, si ceux-ci sont premiers avec n, un message d'avertissement sinon.

16 Etablir une procédure qui prend en entrée $[n_1, n_2, ..., n_k], [a_1, a_2, ..., , a_k]$ et donne en sortie M tel que $M \equiv a_i [n_i]$, pour tout i.

3 Quelques applications

3.1 Développement d'un rationnel en fractions continues

A voir plus tard.

3.2 Calcul du logarithme discret par un algorithme « Baby step, giant step »

Soit g un élément de $\mathbb{F}_p^*$; Ecrire une procédure qui prend en entrée un élément x de $\mathbb{F}_p^*$ et retourne un entier n si $g^n = x$, un message d'erreur sinon (si g n' est pas générateur, x n'est pas forcément une puissance de g!) On utilisera la méthode suivante :

```
variables : m, q,r, h, L1, L2;
m := [√(p − 1)]
h := inverse(g) mod p − 1
L1 := [(g^m, 1), ((g^m)^2, 2), ..., ((g^m)^{m−1}, m − 1)]
L2 := [(xh, 1), (xh^2, 2), ..., (xh^{m−1}, m − 1]]
comparer L1 et L2.
```

Si l'algorithme retourne q et r tels que $(g^m)^q = xh^r$, alors $x = g^{mq+r}$ et $n = mq + r$ convient. Quel est l'intêret de cet algorithme ?

3.3 Le théorème chinois avec correction d'erreurs

Proposition 3 *Soient a, b, m, p des entiers positifs tels que : $b < a$, $a > 4mp > 0$. On effectue l'algorithme d'Euclide étendu de a par b (dont on reprend les notations ci-dessus). Alors :*

Il existe un unique entier i tel que $r_i \leqslant 2m < r_{i-1}$. Cet entier i vérifie :
$\forall (x, y)$ tel que $|xa + yb| \leqslant m$ et $0 < |y| \leqslant p$,
$\exists u \in \mathbb{Z}$ tel que $xa + yb = ur_i, x = us_i$ et $y = ut_i$.

Preuve Considérons les n étapes de l'algorithme d'Euclide étendu pour le calcul du pgcd de a et de b. Ces n étapes s'écrivent :

a				$s_{-1} = 1$	$t_{-1} = 0$	(-1)
b				$s_0 = 0$	$t_0 = 1$	(0)
a	$=$	$bq_1 + r_1$	$0 < r_1 < b$	$s_1 = s_{-1} - q_1 s_0$	$t_1 = t_{-1} - q_1 t_0$	(1)
b	$=$	$r_1 q_2 + r_2$	$0 < r_2 < r_1$	$s_2 = s_0 - q_2 s_1$	$t_2 = t_0 - q_2 t_1$	(2)
$\vdots$						
r_{i-2}	$=$	$r_{i-1} q_i + r_i$	$0 < r_i < r_{i-1}$	$s_i = s_{i-2} - q_i s_{i-1}$	$t_i = t_{i-2} - q_i t_{i-1}$	(i)
$\vdots$						
r_{n-3}	$=$	$r_{n-2} q_{n-1} + r_{n-1}$	$0 < r_{n-1} < r_{n-2}$	$s_{n-1} = \cdots$	$t_{n-1} = \cdots$	$(n-1)$
r_{n-2}	$=$	$r_{n-1} q_n$				(n)

Soient alors (x, y) tels que $|w = xa + yb| \leqslant m$ et $|y| \leqslant p$, alors, modulo a, $w \equiv yb$ et $r_i = s_i a + t_i b \equiv t_i b$, donc a divise $t_i w - y r_i$. Or, $|t_i w - y r_i| \leqslant |t_i w| + |y r_i| \leqslant |t_i| r_{i-1}/2 + p r_i < a/2 + a/2 <\leqslant a$ ($|t_i w| \leqslant |t_i| m < |t_i| r_{i-1}/2 \leqslant a/2$ (cf. exercice 11) et $p r_i < pm \leqslant a/2$ par hypothèse) donc $t_i w - y r_i = 0$. $0 = t_i w - y r_i = t_i(xa + yb) - y r_i = t_i xa + y(t_i b - r_i) = t_i sa - y(s_i a) = a(t_i x - s_i y)$ donc $t_i x - s_i y = 0$. Or, t_i et s_i sont premiers entre eux, donc d'après le lemme de Gauss, il existe u tel que $x = us_i$ et $y = ut_i$; alors, $w = xa + yb = u(s_i a + t_i b) = ur_i$. □

Méthode. Soit $N = n_1 ... n_k$ où les n_i sont premiers entre eux deux à deux. Soit S un entier tel que $0 < S < N$ caractérisé par $A = (a_1, ..., a_k)$ tels que $S \equiv a_i [n_i]$. On suppose qu'après un envoi brouillé, on reçoit $B = (b_1, ..., b_k)$ au lieu de $(a_1, ..., a_k)$. On se propose de montrer que si S est suffisamment petit par rapport à N et si l, le nombre d'indices i sur lesquels A et B diffèrent, est lui aussi suffisamment petit, alors on peut retrouver S.

Plus précisément, soit P un majorant de tous les produits $n_{i_1} .. n_{i_l}$ et soit $M < N$ tel que $N \geqslant 4P^2 M$. On suppose que $0 \leqslant S \leqslant M$. On applique la proposition ci-dessus à $a = N$, b l'entier modulo N caractérisé par les valeurs B par le lemme chinois, $p = P$ et $m = MP$. Alors avec les notations de la proposition, $S = r_i / t_i$.

Preuve Soit I le sous-ensemble de $\{1, ..., k\}$ sur lesquels A et B diffèrent et soit $y := \prod_{j \in I} n_j$.

- Si $i \notin I$, alors $S \equiv a_i = b_i \equiv b, [n_i]$, donc $yS \equiv yb, [n_i]$;

- Si $i \in I$, alors $yS \equiv 0 \equiv yb$, $[n_i]$, puisque n_i divise y.

Donc $yS \equiv yb$, $[N]$, i.e. il existe x tel que $yS = yb + Nx$; Comme $0 \leqslant y \leqslant P$ et $0 \leqslant yS \leqslant mp = M$, la proposition ci-dessus assure l'existence de u entier tel que $yS = r_i u$ et $y = t_i u$, ainsi $S = yS/y = r_i/t_i$. □

17 Simuler une correction d'erreurs : entrer un nombre S, calculer A, effectuer une modification de A (en utilisant le hasard), appliquer la méthode (attention aux constantes). Tester aussi les limites.

3.4 Reconstruction d'un rationnel à partir de ses premières décimales

On rappelle que les rationnels sont les nombres ayant un développement décimal périodique à partir d'un certain rang. On va montrer que si le dénominateur d'un rationnel est suffisamment petit, ce rationnel peut-être reconstruit à partir de ses premières décimales :

Soit p un entier naturel strictement positif. Soit k tel que $d^k \geqslant 4p^2$ ($k \geqslant \ln 4p^2$, si $d = 10$). Soit $z = \frac{s}{t}$ avec $0 < s < t \leqslant p$; Soient $a_1, ..., a_k$ les k premiers chiffres de son développement décimal (i.e. $a_1 d^{k-1} + ... + a_k = E(d^k z)$). On effectue l'algorithme d'Euclide étendu de $a = d^k$ par $b = E(d^k z)$ jusqu'à l'obtention de l'entier i tel que $r_i \leqslant 2p < r_{i-1}$ (cf. proposition ci-dessus dans le cas où $m = p$). Alors $z = \frac{-s_i}{t_i}$.

Preuve $b = E(d^k z)) = E(d^k \frac{s}{t}))$. Donc, b est le quotient de la division euclidienne de $d^k s$ par t. Soit r le reste de cette division :

$$d^k s = bt + r, \quad 0 \leqslant r < t.$$

Donc $r = d^k s - bt$ avec $0 \leqslant t \leqslant p$ et $0 \leqslant r < t \leqslant p$. On applique la proposition ci-dessus (dans le cas où $m = p$) ; il existe un entier u tel que $r = sa - tb = ur_i, s = us_i, -t = ut_i$. Ainsi, $z = \frac{s}{t} = \frac{-s_i}{t_i}$. □

18 Simuler une telle reconstruction : se donner p , calculer k; puis tester sur divers choix de s, t.

Références

[K] Knüth D.E. : The art of computer programming. Vol 2 (seminumerical algorithms) Addison-Wesley.

[NQ] Naudin P. et Quitte C. : Algorithmique algébrique et exercices. Masson.

[R] Robin G. : Algorithmique et cryptographie. Mathématiques et applications. Ellipses.

[S] Shoup V : A computational Introduction to Number Theory and Algebra. Cambridge University Press.