Algèbre et Arithmétique Effectives – 16/09/25
Cours 2

La dernière fois

Théorème 1 : Soient $a, b \in \mathbb{Z}$, avec $(a,b) \neq (0,0)$. Alors il existe $d \in \mathbb{Z}_{\geq 0}$ tel que $\mathrm{pgcd}(a,b) = d$.

Théorème 2 : Soient $a, b \in \mathbb{Z}$ $b \neq 0$. Alors il existe un unique couple d'entiers (q,r) tel que

$$a = bq + r \quad \text{et} \quad 0 \leq r < |b|.$$

Algorithme 2 : Algorithme d'Euclide classique

EUCLIDE (a,b)

Entrées : Deux entiers $a, b \geq 0$
Sortie : $\mathrm{pgcd}(a,b)$

1. Tant que $b \neq 0$ faire
2. $c = b$
3. $b = a \bmod b$
4. $a = c$
5. Renvoyer a

$(a,b) \leftarrow (b, a \bmod b)$

Exemple : $a = 13$ $b = 21$

$$13 = 0 \cdot 21 + 13$$
$$21 = 1 \cdot 13 + 8$$
$$13 = 1 \cdot 8 + 5$$
$$8 = 1 \cdot 5 + 3$$
$$5 = 1 \cdot 3 + 2$$
$$3 = 1 \cdot 2 + 1 \quad = \mathrm{pgcd}(13,21)$$
$$2 = 2 \cdot 1 + 0$$

Nombres de Fibonacci

$F_0 = 0$
$F_1 = 1$
$\forall\, i \geq 2,\ F_i = F_{i-1} + F_{i-2}.$

Quelques rappels de théorie de la complexité

Dans l'analyse d'un algorithme un des buts est d'estimer le temps de calcul en fonction de la taille de l'entrée.

Pour un entier la taille est le nombre de bits nécessaires pour représenter cet entier en mémoire.

On dénote la taille d'un entier $a \in \mathbb{Z}$ par $\mathrm{len}(a)$ (de l'anglais length). On a :

$$\mathrm{len}(a) = \begin{cases} \lfloor \log_2 |a| \rfloor + 1, & \text{si } a \neq 0 \\ 0, & \text{si } a = 0 \end{cases}$$

Pour décrire la croissance du temps de calcul on utilise souvent une notation asymptotique.

Soient f, g deux fonctions à valeurs réelles :

- Grand-O (O) (limite supérieure – pire cas)

  $f(n) = O(g(n)) \iff \exists\ c, n_0 > 0$ telles que $|f(n)| \leq c|g(n)|\ \forall\ n \geq n_0$

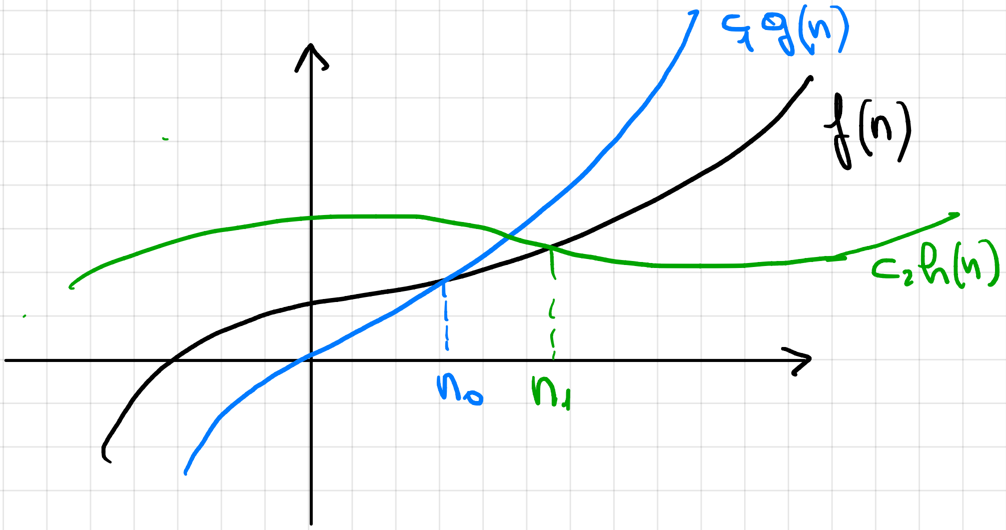
- Grand-Oméga (Ω) (limite inférieure – meilleure cas)

  $f(n) = \Omega(g(n)) \iff \exists\ c, n_0 > 0$ telles que $|f(n)| \geq c|g(n)|,\ \forall\ n \geq n_0$

- Thêta (Θ) – croissance exacte – cas moyen

  $f(n) = \Theta(g(n)) \iff f(n) = O(g(n))$ et $f(n) = \Omega(g(n))$.

  $\iff \exists\ c_1, c_2, n_0 > 0$ telles que $|g(n)|\, c_1 \leq |f(n)| \leq |g(n)| \cdot c_2$



$f(n) = \Theta(g(n))$

$f(n) = \Omega(h(n))$

Exemple : $n^2 = \Theta(n^2) \subseteq \Theta(n^3) \subseteq \Theta(2^n)$.

Exemples

$\Theta(1)$: temps constant, peu importe la taille de l'entrée

$\Theta(\log(n))$: temps logarithmique

$\Theta(n)$: temps linéaire

$\Theta(2^n)$: temps exponentiel.

Théorème (Chapitre 3.3 de Shoup)

Soient $a, b \in \mathbb{Z}$:

1) On peut calculer $a \pm b$ en $\Theta(\mathrm{len}(a) + \mathrm{len}(b))$
temps linéaire

2) On peut calculer ab en $\Theta(\mathrm{len}(a) \cdot \mathrm{len}(b))$.

3) Si $b \neq 0$ on peut calculer le quotient q et le reste r de la division de a par b en $\Theta(\mathrm{len}(b)\,\mathrm{len}(q))$.

Attention : il existe des algorithmes plus rapides pour calculer des multiplications et des divisions.

Coût de l'algorithme d'Euclide

Nous devons estimer :

① Le nombre d'itérations, c'est-à-dire le nombre de divisions euclidiennes.

② coût de chaque itération, c'est-à-dire le coût de chaque division.

$r_0 = a > 0$
$r_1 = b > 0$

1) $r_0 = q_1 r_1 + r_2 \, , \quad 0 < r_2 < r_1$

2) $r_1 = q_2 r_2 + r_3 \, , \quad 0 < r_3 < r_2$

$\vdots$

i) $r_{i-1} = q_i r_i + r_{i+1} \, , \quad 0 < r_{i+1} < r_i$

$\vdots$

$\lambda-1$) $r_{\lambda-2} = q_{\lambda-1} r_{\lambda-1} + r_\lambda \, , \quad 0 < r_\lambda < r_{\lambda-1}$

λ) $r_{\lambda-1} = r_\lambda q_\lambda$ $\qquad$ $\text{pgcd}(a,b)$ $\qquad$ $r_{\lambda+1} = 0$

Combien ça vaut λ dans le pire des cas ?

Théorème : Soient $a, b \in \mathbb{Z}$ avec $a \geq b > 0$. L'algorithme d'Euclide effectue au plus

$$\frac{\log(b)}{\log(\phi)} + 1$$

divisions euclidiennes où $\phi := \frac{1+\sqrt{5}}{2}$ est le nombre d'or ($\phi^2 = \phi + 1$).

Dém

Puisque $b > 0$, alors $\lambda > 0$ (j'effectue au moins une division).

Si $\lambda = 1$ alors l'énoncé est vrai $\left(\text{car } \frac{\log(b)}{\log(\phi)} \geq 0\right)$.

On peut donc supposer $\lambda > 1$.

On montre que $\forall\ i = 0, \dots, \lambda - 1$, on a :

$$\boxed{r_{\lambda - i} \geq \phi^i}$$

On procède pour récurrence :

- initialisation :

$$i = 0 \Rightarrow r_\lambda \geq \underset{= F_2}{1} = \phi^0 \quad ✓$$

$$i = 1 \Rightarrow r_{\lambda-1} \geq r_\lambda + 1 \geq \underset{= F_3}{2} \geq \phi \sim 1{,}62 \quad ✓$$

Pour $i = 2, \dots, \lambda - 1$ on obtient par récurrence

$$r_{\lambda - i} \underset{q_{\lambda-(i-1)} > 0}{\geq} \underbrace{r_{\lambda-(i-1)}}_{\lambda - i + 1} + \underbrace{r_{\lambda-(i-2)}}_{\lambda - i + 2} \underset{\substack{\text{Hypothèse} \\ \text{de récurrence}}}{\geq} \phi^{i-1} + \phi^{i-2} =$$

$$= \phi^{i-2} \underbrace{(\phi + 1)}_{\phi^2} =$$

$$= \phi^i$$

Pour $i = \lambda - 1$ on a :

$$b = r_1 \geq \phi^{\lambda - 1}$$

$$\Updownarrow$$

$$b \geq \phi^{\lambda-1}$$

$$\Updownarrow$$

$$\log(b) \geq \log(\phi^{\lambda-1}) = (\lambda - 1) \log(\phi)$$

$$\Updownarrow$$

$$\lambda \leq \frac{\log(b)}{\log(\phi)} + 1$$

$\square$

Donc l'algorithme d'Euclide effectue au plus $\frac{\log(b)}{\log(\phi)} + 1$ itérations $\rightsquigarrow$ $\Theta(\operatorname{len}(b))$

À chaque itération le temps de calcul est $\Theta(\operatorname{len}(r_i)\operatorname{len}(q_i))$.

Donc le coût total est :

$$T = \sum_{i=1}^{\lambda} \operatorname{len}(r_i)\operatorname{len}(q_i) \underset{\uparrow}{=} \Theta(\operatorname{len}(a)\operatorname{len}(b))$$

Théorème 4.2 Shoup

<u>Théorème</u> : Le coût de calcul de l'algorithme d'Euclide est $\Theta(\operatorname{len}(a)\operatorname{len}(b))$.

Pourquoi le nombre d'or apparaît dans la démonstration ?

En effet dans la démonstration on aurait pu montrer que

$r_{\lambda-i} \geq F_{i+2}$, où F_{i+2} est le $i+2$-ème nombre de Fibonacci.

et utiliser la relation $F_{i+2} \geq \phi^i \quad \forall i \geq 0$ (démontrez la pour exercice).

Il existe une variante de l'algorithme d'Euclide :

Algorithme binaire du pgcd.

Cet algorithme utilise seulement des soustractions et des multiplications et divisions par 2 (càd des simples décalages de bits)

Il se base sur les propriétés suivantes :

- si $a \overset{}{=} 2K_1$ et $b = 2K_2$ sont pairs alors

$$\text{pgcd}(2K_1, 2K_2) = 2\,\text{pgcd}(K_1, K_2)$$

- si $a = 2K$ est pair et b est impair :

$$\text{pgcd}(2K, b) = \text{pgcd}(K, b)$$

- si a et b sont impairs

$$\text{pgcd}(a, b) = \text{pgcd}(a-b, b).$$

Algorithme d'Euclide Etendu

EuclideEtendu(a, b)

<u>Entrées</u> : $a, b \in \mathbb{Z}_{\geq 0}$, $(a,b) \neq (0,0)$

<u>Sorties</u> : un triplet (d, u, v) tel que $d = \text{pgcd}(a,b)$ et $d = au + bv$.

1. Si $b = 0$ alors
2. Renvoyer $(a, 1, 0)$
3. $(q, r) \leftarrow (a \text{ quo } b, a \bmod b)$
   (quotient)
4. $(d, u', v') \leftarrow$ EuclideEtendu(b, r)
5. Renvoyer $(d, v', u' - qv')$

<u>Preuve de correction</u>

On remarque que si on ignore les u et les v

dans l'algorithme, on retrouve exactement l'algorithme d'Euclide récursif.

Donc l'algorithme se termine et l'entier d est exactement $\text{pgcd}(a,b)$.

Il reste à montrer que $d = au+bv$, où u et v représentent la 2ème et 3ème composante de la sortie.

Pour cela on procède par récurrence sur b, c'est à dire sur la valeur strictement décroissante du second argument des appels récursifs.

<u>Initialisation</u> : Si $b=0$, EuclideEtendu$(a,0)$ renvoie $(a,1,0)$ et on a bien

$$\underset{\text{pgcd}(a,0)}{\underset{\parallel}{a}} = a\cdot 1 + b\cdot 0$$

<u>Hypothèse de récurrence</u> : Supposons que pour tout couple (x,y) avec $0 \leq y < b$ EuclideEtendu(x,y) renvoie

$$(d,u',v')$$

tel que $d = \text{pgcd}(x,y)$ et $u'x+v'y = d$.

Considérons maintenant $a,b>0$. On écrit alors

$$a = qb+r, \quad 0 \leq r < b.$$

L'algorithme effectue donc l'appel récursif :

$$(d,u',v') = \text{EuclideEtendu}(b,r).$$

Par hypothèse de récurrence, on a :

$$d = \text{pgcd}(b,r) \quad \text{et} \quad u'b+v'r = d.$$

En remplaçant $r = a-qb$ on obtient

$$u'b + v'(a-qb) = d$$
$$\Updownarrow$$
$$v'a + (u'-qv')b = d$$

et l'algorithme retourne exactement $(d, v', u' - qv')$.

$\square$

<u>Remarque</u> : Le temps d'exécution de l'algorithme d'Euclide étendu est du même ordre de l'algorithme d'Euclide.