Problème de Mathématiques

Référence pp1810 — Version du 15 octobre 2025

On compare ici deux algorithmes de calcul du pgcd de deux entiers naturels.

1. On *rappelle* que le pgcd de deux entiers naturels a et b est l'unique $d \in \mathbb{N}$ qui divise à la fois a et b et tel que tout entier δ qui divise à la fois a et b divise également d.

Étant donnés deux entiers naturels a et b (non nuls), on pose

$$\Omega = \left\{ k \in \mathbb{N}^* \ : \ k \mid a \quad \text{et} \quad k \mid b \right\}$$

ainsi que

$$M = \max \Omega.$$

1. a. Démontrer que l'entier M est bien défini.

1. b. Démontrer que M est le pgcd de a et b.

1. c. Pour calculer M, on peut passer en revue tous les entiers compris entre 1 et a et retourner le dernier de ces entiers qui divise à la fois a et b.

Écrire *en langage Python* une fonction `gcd(a, b)` qui retourne le pgcd de a et b calculé selon la méthode qui vient d'être décrite.

2. La fonction Python `euclide(a,b)` retourne le pgcd de a et b calculé au moyen de l'algorithme d'Euclide.

```
def euclide(a, b):
  u, v = a, b
  while v!=0:
    u, v = v, u%v
  return u
```

2. a. Écrire *en langage Python* une fonction récursive `euclide_rec(a,b)` qui retourne le pgcd des entiers a et b calculé au moyen de l'algorithme d'Euclide.

2. b. En utilisant la fonction `euclide`, écrire *en langage Python* une fonction `gcd_trois(a,b,c)` qui retourne le pgcd des entiers a, b et c.

3. La suite de Fibonacci $(F_n)_{n\in\mathbb{N}}$ est définie par

$$F_0 = 0, \quad F_1 = 1 \quad \text{et} \quad \forall\, n \in \mathbb{N}, \quad F_{n+2} = F_{n+1} + F_n.$$

On *admet* que $(F_n)_{n\in\mathbb{N}}$ est une suite strictement croissante d'entiers naturels et que, lorsque n tend vers $+\infty$,

$$F_n \sim \frac{\varphi^n}{\sqrt{5}}$$

où $\varphi = (1+\sqrt{5})/2$.

3. a. Quel est le reste de la division euclidienne de F_{n+2} par F_{n+1} ?

3. b. En déduire le nombre u_n de divisions euclidiennes effectuées en calculant le pgcd de F_{n+2} et F_{n+1} avec la fonction `euclide`.

3. c. On note v_n, le nombre de divisions euclidiennes effectuées pour calculer le pgcd de F_{n+2} et F_{n+1} avec la fonction `gcd`. Comparer les ordres de grandeur de u_n et de v_n lorsque n tend vers $+\infty$.

4. Écrire *en langage Python* une fonction `fibo(n)` dont l'argument n est un entier naturel et qui retourne le nombre de Fibonacci F_n.

Solution ❀ Nombres de Fibonacci

1. a. Le nombre d'entiers $k \in \mathbb{N}^*$ qui divisent à la fois a et b est inférieur au nombre d'entiers $1 \leqslant k \leqslant a$. Par conséquent, Ω est une partie finie de $\mathbb{N}$.

D'autre part, 1 divise a et b, donc $1 \in \Omega$. En tant que partie finie et non vide de $\mathbb{N}$, Ω admet un plus grand élément.

1. b. Soit d, le pgcd de a et b.

Comme $M \in \Omega$, l'entier M est un diviseur commun à a et b, donc M divise d : il existe $q \in \mathbb{N}^*$ tel que $d = q \times M$ et comme $q \geqslant 1$, alors $M \leqslant d$.

Réciproquement, en tant que diviseur commun à a et b, le pgcd d appartient à Ω et comme M est le plus grand élément de Ω, on a donc $d \leqslant M$.

Finalement, on a bien $M = d$.

1. c. Tout diviseur commun à a et b est inférieur à a et à b. Pour limiter les calculs, on pose $m = \min\{a, b\}$ et on parcourt la liste des entiers $2 \leqslant k \leqslant m$ (la boucle `for` doit s'achever avec $k = m$) : on sait que 1 est un diviseur commun à a et b.

Pour chaque entier k, on calcule les restes de la division euclidienne de a par k et de la division euclidienne de b par k : s'ils sont tous les deux nuls, c'est que k est un diviseur commun de a et b et dans ce cas, on affecte la valeur de k à la variable d.

On retourne la valeur finale de d, qui est le dernier (et donc le plus grand) diviseur commun trouvé.

```
def gcd(a, b):
  m = min(a, b)
  d = 1
  for k in range(2, m+1):
    if (a%k==0) and (b%k==0):
      d = k
  return d
```

2. a. L'algorithme d'Euclide repose sur la relation

$$\forall\, b > 0, \qquad \text{pgcd}(a, b) = \text{pgcd}(b, r)$$

où r est le reste de la division euclidienne de a par b et sur le cas particulier :

$$\forall\, a \in \mathbb{N}, \qquad \text{pgcd}(a, 0) = a.$$

La version récursive de l'algorithme s'en déduit immédiatement.

```
def euclide_rec(a, b):
  if (b==0):
    d = a
  else:
    d = euclide_rec(b, a%b)
  return d
```

2. b. Il suffit de savoir que

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

(associativité du pgcd).

```
def gcd_trois(a, b, c):
  return euclide(a, euclide(b, c))
```

3. a. D'après la relation de récurrence :

$$F_{n+2} = F_{n+1} + F_n = 1 \times F_{n+1} + F_n$$

et comme la suite de Fibonacci est positive et strictement croissante, on en déduit que $0 \leqslant F_n < |F_{n+1}|$. La relation ci-dessus est donc bien la division euclidienne de F_{n+2} par F_{n+1} : le quotient est égal à 1 et le reste à F_n.

3. b. Dans la fonction `euclide`,

— Le couple `(u,v)` est initialement égal à (F_{n+2}, F_{n+1}) ;
— D'après la question précédente, à chaque étape, le couple (F_{k+1}, F_k) est remplacé par le couple (F_k, F_{k-1}) ;
— On sort de la boucle lorsque `v` devient nul et dans ce cas, le couple `(u,v)` a pour valeur $(F_1, F_0) = (1, 0)$.

On passe de F_{n+1} à $F_0 = F_{(n+1)-(n+1)}$ en effectuant $(n+1)$ itérations et une division euclidienne à chaque itération, donc on effectue $u_n = n+1$ divisions euclidiennes en tout.

REMARQUE.— On a démontré au passage que F_{n+2} et F_{n+1} étaient premiers entre eux.

3. c. On parcourt la liste des entiers compris entre 1 et

$$F_{n+1} = \min\{F_{n+1}, F_{n+2}\}$$

et pour chacun de ces entiers, on effectue deux divisions euclidiennes. On effectue en tout $v_n = 2F_{n+1}$ divisions euclidiennes.

❧ Lorsque n tend vers $+\infty$,

$$u_n \sim n \quad \text{et} \quad v_n \sim \frac{2\varphi^n}{\sqrt{5}}$$

donc $u_n = o(v_n)$ (puisque $|\varphi| > 1$). La fonction `euclide` est donc sensiblement plus efficace que la fonction `gcd`.

4. On retourne à part la valeur F_0. Pour calculer F_n avec $n \geqslant 1$, on effectue une boucle.

Initialisation
$(u, v) = (F_0, F_1) = (0, 1)$
Itération $(1 \leqslant k < n)$
Entrée de boucle
$(u, v) = (F_{k-1}, F_k)$
Sortie de boucle
$(u, v) = (F_k, F_{k+1}) = (F_k, F_k + F_{k-1})$
Terminaison
$(u, v) = (F_{n-1}, F_n)$

— L'entrée de la première itération $(k = 1)$ coïncide avec l'initialisation.
— La sortie de la k-ième itération coïncide avec l'entrée de la $(k+1)$-ième itération.
— La terminaison coïncide avec la sorte de la dernière itération $(k = n-1)$.

En retournant la valeur finale de v, la fonction `fibo` donne bien la valeur de F_n.

On insiste sur un détail essentiel : l'instruction

```
for k in range(1, n):
```

traduit exactement l'encadrement $1 \leqslant k < n$ qui figure sur le tableau.

```
def fibo(n):
  if (n==0):
    return 0
  else:
    u, v = 0, 1
    for k in range(1, n):
      u, v = v, u+v
    return v
```

❧ On calcule F_n en effectuant $(n-1)$ itérations de la boucle et chaque itération calcule une somme. Le nombre de sommes effectuées est donc équivalent à n : la complexité de la fonction `fibo` est donc linéaire.

REMARQUE.— On peut faire mieux ! En exploitant la relation de récurrence linéaire et l'algorithme d'exponentiation rapide, on peut écrire une fonction de complexité logarithmique.