Exercices 23 à 30

Exercice 23 (suite récurrente). On considère la suite définie par $u_0 = 1$ et la relation de récurrence

$$\forall n \geqslant 1, \quad u_n = u_{[n/2]} + u_{[n/3]},$$

où $[x]$ désigne la paretie entière de x (plus grand entier inférieur ou égal à x).

(1) Écrire en pseudo-code une fonction doublement récursive **u(n)** qui renvoie la valeur de u_n.
(2) Dessiner l'arbre des appels récursifs associés à l'appel u(17). Combien y a-t-il d'appels à la fonction u() ?

Exercice 24 (générateurs). Écrire en Python les générateurs suivants :

(1) `intercale(n)` délivre les `n` premiers termes de la suite

$$1, 2, 3, 4, 6, 5, 8, 10, 12, 7, 14, 16, 18, 20, \ldots$$

(un nombre impair, 1 nombre pair, 1 nombre impair, 2 nombres pairs, 1 nombre impair, 3 nombres pairs, etc.)

(2) `triplets(L)` délivre tous les triplets (x, y, z) formés d'éléments de `L` tels que $x < y < z$

(3) `quatrain(N)` délivre, dans l'ordre croissant, tous les nombres entiers inférieurs ou égaux à `N` dont l'écriture en base 2 utilise exactement 4 fois le chiffre 1

(4) `facteurs(n)` délivre, dans l'ordre croissant (et comptés avec multiplicité), les facteurs premiers de l'entier `n`$\geqslant 1$

(5) `sommes(L)` délivre, dans un ordre quelconque, toutes les sommes partielles possibles d'éléments de la liste `L` (utiliser la récursivité)

(6) `pythagore(N)` délivre, dans un ordre quelconque, tous les triplets (x, y, z) d'entiers tels que

$$1 \leqslant x \leqslant y \leqslant z \leqslant N \quad \text{et} \quad x^2 + y^2 = z^2.$$

Exercice 25 (analyse de code I). Déterminer ce qu'affichent les programmes Python ci-dessous.

```
def f(n):
    for k in range(1,n+1):
        if n%k==0:
            yield k

for x in f(30):
    print(x)
```

```
def g(n):
    if n>0:
        for k in g(n-1):
            yield k
        for k in range(n):
            yield n

print([x for x in g(5)])
```

```
def h(s,n):
    if n>0:
        for x in s:
            for t in h(s,n-1):
                yield x+t
    else:
        yield ''

print(list(h('ab',3)))
```

Exercice 26 (analyse de code II). Déterminer, pour chaque ligne d'instruction ci-dessous, le type de la variable `x` (nombre, liste, tuple, générateur, etc.) après exécution de la ligne considérée.

```
1 x = [1/a for a in range(1,10)]
2 x = (1/a for a in range(1,10))
3 x = sum([1/a for a in range(1,10)])
4 x = [sum(1/a for a in range(1,10))]
5 x = (sum(1/a for a in range(1,10)),)
6 x = [1/sum([a]) for a in range(1,10)]
7 x = (1/sum(a for a in range(1,10)))
```

Exercice 27 (nombres de Hamming). Un entier n est un *nombre de Hamming* s'il s'écrit sous la forme $n = 2^a 3^b 5^c$, où $a, b, c \in \mathbb{N}$.

(1) Déterminer à la main les 20 premiers nombres de Hamming.

(2) Pour tout ensemble fini $X \subset \mathbb{R}$, on définit

$$P(X) = \left\{ \prod_{x \in X} x^{k_x},\ (k_x) \in \mathbb{N}^X \right\}.$$

Autrement dit, $P(X)$ est formé de tous les produits d'éléments de X à des puissances entières positives quelconques. Pour quel ensemble X a-t-on égalité entre $P(X)$ et l'ensemble des nombres de Hamming ?

(3) Montrer que si $x \in X$ et $Y = X \setminus \{x\}$, alors

$$P(X) = \{x^p y,\ p \in \mathbb{N},\ y \in P(Y)\}.$$

(4) En déduire, en appliquant le principe *diviser pour régner*, le pseudo-code d'un générateur récursif **tous_produits(X,N)** qui prend en entrée une liste X de réels distincts strictement positifs et un entier N, et délivre (dans un ordre quelconque, mais sans répétition) tous les éléments de $P(X)$ inférieurs ou égaux à N.

(5) **(TP)** Traduire ce pseudo-code en langage Python.

(6) **(TP)** Tester le générateur `tous_produits()` pour `X=[2]`, pour `X=[2,3]` puis pour `X=[2,3,5]`.

(7) **(TP)** En déduire la valeur du 2020$^{\text{e}}$ nombre de Hamming.

Exercice 28 (recherche rapide du terme de rang k).

(1) Écrire, en s'inspirant de l'agorithme du tri rapide, le pseudo-code d'une fonction récursive **rang(L,k)** qui détermine, dans une liste L d'au moins k nombres, la valeur du terme de rang k (c'est-à-dire le k-ième plus petit élément de L).

(2) **(TP)** Implémenter cet algorithme en Python.

(3) **(TP)** Utiliser cette implémentation pour retrouver directement le résultat de la question 7 de l'exercice 27 à partir du générateur écrit à la question 6, sans utiliser de fonction de tri.

Exercice 29 (croissance exponentielle de la suite de Fibonacci).

(1) **(TP)** Écrire une fonction **non récursive** `fib(n)`, qui prend en entrée un entier $n \geqslant 1$ et renvoie la liste des termes $\mathcal{F}_0$, $\mathcal{F}_1$, $\ldots \mathcal{F}_n$ de la suite de Fibonacci. Tester la fonction `fib()` en affichant la liste `L` obtenue par `L = fib(40)`.

(2) **(TP)** Calculer, à l'aide de la liste `L` précédente, la liste `R` des termes $\mathcal{F}_2/\mathcal{F}_1, \mathcal{F}_3/\mathcal{F}_2, \ldots, \mathcal{F}_{40}/\mathcal{F}_{39}$, et afficher le résultat. Que constate-t-on ? Pouvait-on prédire la valeur numérique observée ?

(3) **(TP)** À l'aide de l'équivalent de $\mathcal{F}_n$ vu en cours, prédire une valeur approchée de $\mathcal{F}_n$ sous la forme $\mathcal{F}_n \simeq 10^x$ pour $n = 2^{20}$. Combien de chiffres a l'écriture décimale de $\mathcal{F}_n$? Que donne `fib(2**20)` ?

(4) **(TP*)** On considère la matrice $A = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}$. Montrer par récurrence que $A^n = \begin{pmatrix} \mathcal{F}_{n+1} & \mathcal{F}_n \\ \mathcal{F}_n & \mathcal{F}_{n-1} \end{pmatrix}$ pour tout $n \geqslant 2$, puis en déduire un moyen de calculer rapidement $\mathcal{F}_n$ pour $n = 2^{20}$. Vérifier que la valeur trouvée a bien le nombre de chiffres prédit à la question précédente.

Exercice 30 (cryptarithmes, TP). Un cryptarithme est une équation mathématique dans laquelle chaque lettre représente un chiffre, un même chiffre ne pouvant pas être représenté par deux lettres distinctes. Par ailleurs, aucun nombre de 2 chiffres ou plus ne peut commencer par 0.

Nous avons résolu en cours le cryptarithme SIX^2=TROIS, qui admet pur unique solution $169^2 = 28561$.

En utilisant la fonction `permutations` du module `itertools`, résoudre les cryptarithmes suivants :

(1) UN $\times$ UN + UN = DEUX

(2) SEND + MORE = MONEY
