RÉCURSIVITÉ

§1. Exemples

Exercice 1 — Pour chacune des situations suivantes, donner deux programmes : l'un récursif, et l'autre non :

a) le calcul des factorielles,

b) les coefficients binomiaux,

c) le calcul du PGCD,

d) l'algorithme de dichotomie.

Exercice 2 — Soit x un entier impair au moins égal à 3. On suppose qu'il admet une factorisation $x = pq$ avec $q \geqslant p \geqslant 3$ (pas forcément premiers).

a) Justifier qu'il existe deux entiers a et b, avec $a > b$, tels que $p = a - b$ et $q = a + b$.

b) Prouver que $b \leqslant (x-9)/6$.

c) On suppose disposer d'un programme efficace `Racine(`y`)` qui calcule $\lfloor\sqrt{y}\rfloor$. En remarquant que $x+b^2 = a^2$, écrire un programme `Rompre(`x`)` qui renvoie une liste `[`p`, `q`]` telle que $x = pq$ ou bien la liste `[`x`]` lorsque x est premier.

d) On suppose toujours x impair. Déduire de ce qui précède un programme `Factoriser(`x`)` qui construit et renvoie la liste des facteurs premiers impairs de x, avec multiplicité.

Exercice 3 — On donne un polynôme

$$\mathrm{P}(x) = a_d x^d + a_{d-1}x^{d-1} + \cdots + a_1 x + a_0,$$

qu'on représente par la liste de ses coefficients `[`a_0`, `a_1`, ...]`. Écrire un programme qui étant donnés x et cette liste calcule et renvoie $\mathrm{P}(x)$.

Exercice 4 — Soit $\mathrm{B} \geqslant 2$. Écrire des programmes récursifs qui calculent :

a) le nombre de chiffres de x en base B,

b) le nombre d'apparitions, dans l'écriture en base B de x, du chiffre $k \in \{0\,;\,\ldots\,;\,\mathrm{B}-1\}$,

c) le chiffre de poids fort de x en base B,

d) la somme des chiffres de x en base B.

Exercice 5 — Soit a/b un rationnel positif. Un développement en *fractions égyptiennes* est une écriture

$$\frac{a}{b} = \frac{1}{x_0} + \frac{1}{x_1} + \cdots + \frac{1}{x_{r-1}}$$

avec une suite *strictement* croissante de dénominateurs.

a) Justifier qu'une telle écriture est toujours possible.

b) Écrire un programme qui renvoie une liste de dénominateurs possibles, connaissant a et b.

c) Écrire un programme qui donne une solution la plus courte, c'est-à-dire utilisant aussi peu de fractions que possible (elle n'est pas forcément unique).

Exercice 6 — Écrire un programme qui étant donnée une liste `L` rangée dans l'ordre croissant et un élément x (pas forcément dans la liste) renvoie le plus petit indice i tel que `L[`i`]` est supérieur ou égal à x. On renverra la longueur de la liste si un tel indice n'existe pas.

Ici encore, on proposera une version récursive et une version non récursive.

Exercice 7 — Soit m un entier naturel. On coupe arbitrairement deux entiers x et y de la manière suivante :

$$x = a \times 2^m + b \quad \text{et} \quad y = c \times 2^m + d.$$

On forme ensuite leur produit

$$xy = ac \times 2^{2m} + [(a+b)(c+d) - ac - bd] \times 2^m + bd.$$

Le calcul de xy se ramène alors à trois multiplications plus petites : ac, bd et $(a+b)(c+d)$, ainsi que des additions et multiplications.

a) Quel est le coût, en fonction de leurs nombres de chiffres, de la multiplication $x \times y$, si l'on utilise l'algorithme appris à l'école ?

b) Et quel est le coût de l'addition ou de la soustraction ?

c) Écrire un algorithme de multiplication basé sur le calcul ci-desus, qui prend m de sorte à « couper en deux » les nombres x et y, et qui procède récursivement pour calculer les trois produits plus petits.

d) Montrer que le coût (en opérations sur les chiffres) de cet algorithme est proportionnel à $n^{\log_2(3)}$, où n est le nombre de chiffres de l'opérande le plus grand.

§2. Lien avec les suites récurrentes

Exercice 8 — On considère la suite $(\mathrm{F}_n)_{n\geqslant 0}$ de premiers termes $\mathrm{F}_0 = 0$, $\mathrm{F}_1 = 1$ et vérifiant la relation de récurrence $\mathrm{F}_{n+2} = \mathrm{F}_{n+1} + \mathrm{F}_n$.

a) Écrire un programme récursive qui exploite naïvement cette relation de récurrence pour calculer F_n.

b) Montrer que le nombre d'opérations arithmétiques effectuées par ce programme est proportionnel à

$$\left(\frac{1+\sqrt{5}}{2}\right)^n.$$

Jusqu'à quelle valeur de n est-il raisonnable de l'utiliser ?

c) Écrire un programme récursif calculant F_n en $O(n)$ opérations arithmétiques.

Exercice 9 — Reprendre l'exercice précédent pour une relation de récurrence

$$u_{n+2} = au_n + bu_{n+1}.$$

Qu'est-ce qui change, et qu'est-ce qui ne change pas ?

Exercice 10 — Écrire des programmes calculant le terme d'ordre n pour les suites suivantes. On donnera dans chaque cas une version récursive et une version non récursive :

a) $u_0 = 1$ et $u_{n+1} = 1 + nu_n$,

b) $u_0, u_1, u_2 \in \mathbf{R}$ et $u_{n+3} = au_n + bu_{n+1} + cu_{n+2}$,

c) $u_0 = u_1 = 1$ et $u_{n+2} = 1 + u_n u_{n+1}$,

d) $u_0 = 1$ et $u_{n+1} = u_n + u_{\lfloor n/2 \rfloor}$,

e) $u_0 = 1$ et $u_{n+1} = \sum_{k=0}^{n} u_k$,

f) $u_0 = 0$, $u_1 = 1$ et $u_{n+1} = \frac{1}{n+1} \sum_{k=0}^{n} u_k u_{n-k}$.

§3. Exponentiation rapide

Exercice 11 — Écrire un programme réalisant l'exponentiation rapide pour calculer x^n. On donnera une version récursive et une version non récursive.

Exercice 12

a) Soient x un réel positif et $a, b \geqslant 1$ des entiers. Prouver que

$$\left\lfloor \frac{\lfloor x/a \rfloor}{b} \right\rfloor = \left\lfloor \frac{x}{ab} \right\rfloor.$$

b) En déduire que pour tout entier $n \geqslant 0$ on a

$$\lfloor \ldots \lfloor \lfloor \lfloor x/a \rfloor /a \rfloor /a \rfloor \ldots /a \rfloor = \lfloor x/a^n \rfloor.$$

Exercice 13

a) Montrer que le nombre de multiplications réalisées par l'algorithme d'exponentiation rapide pour calculer x^n est proportionnel à $\log_2(n)$.

b) Donner le nombre précis de multiplications réalisées, en utilisant l'écriture binaire de n.

c) Est-ce optimal, c'est-à-dire peut-on calculer x^n dans un autre ordre de sorte à réaliser moins de multiplications que l'exponentiation rapide ?

Exercice 14 — Écrire un programme qui calcule le nombre minimal de multiplications à réaliser pour calculer x^n.

Exercice 15

a) Programme l'exponentiation rapide pour les éléments de $\mathscr{M}_2(\mathbf{Z})$.

b) En considérant les puissances de

$$\mathrm{A} = \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix},$$

écrire un algorithme pour calculer F_n (le nombre de Fibonacci) en $O(\ln n)$ opérations arithmétiques.

Exercice 16 — Écrire un programme qui réalise l'exponentiation rapide dans un monoïde quelconque. Il prendra en argument l'opération `op(`x`,` y`)` et le neutre `e` du monoïde. On donnera deux versions : l'une récursive, l'autre pas.

§4. Problèmes combinatoires

Exercice 17

a) Écrire un programme récursif qui construit la liste de listes représentant toutes les parties d'une liste donnée. Pour `[1, 2, 3]` par exemple, on attend un résultat du genre `[[], [1], [2], [3], [1, 2], [1, 3], [2, 3], [1, 2, 3]]`, peut-être dans un ordre différent.

Reprendre la question précédente pour obtenir uniquement les parties à k éléments.

Exercice 18 — Écrire un programme qui renvoie toutes les permutations d'une liste. On supposera, pour simplifier, que tous les éléments de cette liste sont distincts.

Exercice 19 — Une *partition* d'un entier naturel x est une somme

$$x = x_0 + x_1 + \ldots x_{r-1}$$

avec $x_0 \geqslant x_1 \geqslant \cdots \geqslant x_{r-1} \geqslant 1$. On représentera une telle partition par la liste `[`x_0`, ...]`.

a) Écrire un programme qui renvoie la liste de toutes les partitions de x dont le plus grand terme *est* M. C'est donc une liste de listes.

b) En déduire un programme qui renvoie toutes les partitions de x.

c) Écrire un programme qui renvoie la liste de toutes les partitions dont le dernier terme est au moins égal à m.

Exercice 20

a) Écrire un programme qui étant donnée une liste d'au moins deux objets, construit et renvoie toutes les partitions de cette liste en une liste de longueur 2 et une autre liste (peu importe l'ordre des termes).

b) En déduire un programme qui résout le problème du *compte est bon* : on donne un résultat R et une liste de nombres, et on cherche à former R en utilisant les nombres de cette liste (pas forcément tous, mais au plus une fois chacun) et les quatre opérations arithmétiques.

Exercice 21 — Écrire un programme qui range les entiers 1, 2, ..., n dans r listes, de sorte à ce que si x et y (pas forcément distincts) sont dans une liste de cette collection, alors $x+y$ n'y figure pas. Le nombre r de listes utilisées doit être aussi petit que possible.