TP 1

Ibrahim ALAME

2/10/2019

1 Itérations et récursions

1. Écrire un programme qui affiche le plus grand et le plus petit d'une suite d'entiers saisis. La suite se termine par un 0.
2. Écrire un programme qui détermine tous les diviseurs d'un nombre entier saisi plus grand que 1.
3. Écrire un programme qui simule la division euclidienne entre deux entiers saisis au clavier, sans utiliser l'opérateur /. Afficher le quotient et le reste.
4. Écrire un programme qui effectue la multiplication de deux entiers positifs m et n, sans utiliser l'opérateur *, selon le principe suivant :

$$m \times n = \begin{cases} m \times (n-1) + m & \text{si } n \text{ est impair} \\ (2m) \times (n/2) & \text{si } n \text{ est pair non nul} \end{cases}$$

   Exemple

$$\begin{aligned} 36 \times 7 &= 36 \times 6 + 36 \\ &= 72 \times 3 + 36 \\ &= 72 \times 2 + 108 \\ &= 144 \times 1 + 108 \\ &= 144 \times 0 + 252 \\ &= 252 \end{aligned}$$

   Écrire un programme qui lit deux entiers au clavier et affiche leurs produit selon l'algorithme récursif défini ci-dessus.
5. (a) Soit la somme $S_n = 1 + 2 + \cdots + (n-1) + n$. Vérifier que cette suite peut être définie par récurrence par :

$$\begin{cases} S_n = S_{n-1} + n \\ S_0 = 0 \end{cases}$$

   (b) En déduire deux fonctions python permettant de calculer la somme S_n par deux méthodes distinctes (itérative et récursive).

   (c) Écrire de même, deux fonctions python calculant le factoriel $n!$ d'un entier positif n donné, par deux méthodes distinctes.

   (d) On désigne par `lister(n)` une procédure permettant d'afficher à la sortie standard la suite naturelle de 1 à n. En remarquant que cette procédure vérifie la relation de récurrence :

$$\text{lister}(n) = \text{lister}(n-1), \text{afficher}(n)$$

   Écrire une fonction python qui affiche $1, 2, ..., n$ sans passer par une boucle.

(e) Sans modifier aucun caractère dans le code précédant, adapter la procédure `lister` pour qu'elle affiche la même suite dans l'ordre décroissant $n, ..., 2, 1$.

6. Soit la somme

$$S_n = \sum_{i=1}^{n} \frac{1}{i^2}$$

(a) Ecrire un programme python qui calcule la somme S_n par deux méthodes .

(b) Sachant que $\sum_{i=1}^{\infty} \frac{1}{i^2} = \frac{\pi^2}{6}$. Determiner le plus petit n pour que $|S_n - \frac{\pi^2}{6}| \leq 10^{-3}$.

2 Puissance $n^{\text{ième}}$ matricielle

1. Écrire un programme qui effectue la puissance $n^{\text{ième}}$, x^n où x et n sont deux entiers positifs, sans utiliser l'opérateur de puissance, selon le principe suivant :

$$x^n = \begin{cases} x \times x^{n-1} & \text{si } n \text{ est impair} \\ (x^2)^{n/2} & \text{si } n \text{ est pair non nul} \end{cases}$$

(a) Écrire un programme python qui lit deux entiers au clavier x et n et affiche la puissance x^n selon l'algorithme récursif défini ci-dessus.

(b) Écrire un programme analogue au précédent permettant d'effectuer la puissance $n^{\text{ième}}$ matricielle M^n où M est une matrice carré et n un entier positif.

3 Fibonacci dichotomique

On considère la suite de Fibonacci $(f_n)_n$ définie par :

$$\begin{cases} f_n = f_{n-1} + f_{n-2} \\ f_0 = 0, f_1 = 1 \end{cases}$$

1. Écrire, par deux méthodes, la fonction `fibo(n)` qui retourne la valeur f_n de la suite de Fibonacci pour un entier positif n donné.
2. On définit F_k, vecteur de couples de valeurs de la fonction de Fibonacci, par

$$F_k = \begin{pmatrix} f_k \\ f_{k-1} \end{pmatrix} \qquad \text{en particulier, } F_1 = \begin{pmatrix} f_1 \\ f_0 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$$

Étant donnée la matrice

$$M = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}$$

pour tout k ; $k \geq 2$, on a $F_k = MF_{k-1}$. Ainsi : $F_2 = MF_1$, $F_3 = MF_2 = M^2F_1$, ..., $F_n = M^{n-1}F_1$.

Pour calculer la valeur f_n on calculera la matrice M^{n-1} puis on retournera la première composante du vecteur $M^{n-1}F_1$, c'est-à-dire la valeur $M^{n-1}[0][0]$.

Écrire la fonction `fibo(n)` qui retourne la ne valeur f_n de la suite de Fibonacci avec une complexité $\Theta(\log n)$.

4 Stocks : répartition optimale

Une société commerciale dispose de E entrepôts et d'un stock S d'un produit à répartir entre les E entrepôts. Cette société connaît pour chaque entrepôt k, $0 \leq k < E$, et chaque quantité entière s de son stock, $0 \leq s \leq S$, le gain $g(k, s)$ qu'elle obtient en livrant à l'entrepôt k la quantité s de ce produit. Remarque : le gain obtenu est nul pour une livraison nulle. Cette société veut connaître le gain total maximum qu'elle peut obtenir en répartissant au mieux son stock S sur ses E entrepôts. En notant $m(k, s)$ le gain maximum obtenu en répartissant un stock s sur les k premiers entrepôts, le gain total maximum est $m(E, S)$:

1. base de la récurrence : donner les valeurs $m(0, s)$, $\forall s$; $0 \leq s \leq S$;
2. cas général : justifier que les valeurs $m(k, s)$ ($\forall k$; $1 \leq k \leq E$ et $\forall s$; $0 \leq s \leq S$) se calculent à l'aide de la formule :
$$m(k, s) = \max_{0 \leq s' \leq s} (g(k-1, s') + m(k-1, s-s'))$$
3. calculer une matrice $M[0..E][0..S]$ de terme général $M[k][s] = m(k, s)$;
4. donner la complexité de ce calcul ;
5. on note $l(k, s)$ la quantité de produit livrée au k-ième entrepôt (entrepôt de numéro $k-1$) dans une répartition optimale d'un stock s sur les k premiers entrepôts. Modifier le programme afin de calculer à la volée une matrice $L[0..E][0..S]$ de terme général $L[k][s] = l(k, s)$;
6. écrire un programme d'affichage d'une répartition optimale du stock S sur les E entrepôts (invariant, initialisation, condition d'arrêt, implication, programme commenté par l'invariant.)

Application numérique :
— Deux exemples :

$$G_1 = \begin{pmatrix} 0 & 10 & 20 & 30 & 40 & 50 & 60 \\ 0 & 10 & 20 & 40 & 45 & 45 & 45 \\ 0 & 5 & 20 & 25 & 30 & 35 & 40 \end{pmatrix} \qquad G_2 = \begin{pmatrix} 0 & 10 & 20 & 30 & 40 & 50 & 60 \\ 0 & 50 & 51 & 52 & 53 & 54 & 55 \\ 0 & 5 & 50 & 50 & 55 & 60 & 65 \end{pmatrix}$$