Informatique PT. 2024 - 2025

Mathématiques & Informatique - F. Gaunard
http://frederic.gaunard.com
Lycée Voltaire, Paris 11e.

1 Travaux Pratiques : Révisions d'algorithmique

Exercice 1.

Donner une fonction récursive qui détermine si un *flottant* appartient à une liste **triée**, en effectuant une recherche par dichotomie.

Exercice 2.

Écrire une fonction d'en-tête `def Doublon(L)` qui prend en entrée une liste `L` et qui renvoie `True` si elle contient deux fois un même élément, et `False` sinon.
Quelle est la complexité de votre fonction?

Exercice 3. Liste des nombres premiers

On rappelle que la commande `a//b` renvoie le quotient dans la division euclidienne de a par b, et `a%b` le reste.

1. Écrire une fonction d'en-tête `def diviseurs(n)` qui prend en entrée un entier positif n et renvoie la liste des diviseurs positifs de n.
2. Quelle est la complexité de votre fonction? Pouvez-vous donner une fonction en $O(\sqrt{n})$?
3. En déduire une fonction `def premier` qui prend en entrée un entier n puis renvoie `True` s'il est premier, et `False` sinon.
4. Afficher à l'écran la liste des nombres premiers inférieurs à 1000. Combien y en a-t-il?

Exercice 4. Algorithme d'Euclide

Implémenter l'algorithme d'Euclide, qui calcule le plus grand commun diviseur de deux entiers a et b non tous deux nuls, en se basant sur les propriétés suivantes :

i. $\text{pgcd}(a, 0) = a$,
ii. $\text{pgcd}(a, b) = \text{pgcd}(b, r)$ où r est le reste de la division euclidienne de a par $b \neq 0$.

Exercice 5. Plus grande somme d'une sous-liste

On considère une liste `L` d'entiers. On veut écrire un programme la plus grande valeur possible de la somme des termes consécutifs d'une sous-liste de L.

Par exemple, on devra avoir l'affichage suivant lors de l'exécution ci-dessous.

```
>>> recherche_s_liste([2, -2, -3, 2, 3, -4, 2, 3, -1])
6
```

Exercice 6. Suite et codage de Fibonacci avec le théorème de Zeckendorf

On rappelle que la suite de Fibonacci (F_n) est la suite de nombre réels définis par

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

Les premiers termes de la suite sont $0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55$...

Théorème 1. **Théorème de Zeckendorf**

Le **Théorème de Zeckendorf** dit la chose suivante.
Pour tout entier non nul n, il existe un unique entier k et un unique $k-$uplet d'entiers $(c_1, .., c_k)$ vérifiant

- ✗ $c_1 \geq 2$
- ✗ $c_i + 1 < c_{i+1}$

tels que

$$n = \sum_{i=1}^{k} F_{c_i}$$

Cette décomposition de n en somme de nombres de la suite de Fibonacci est appelée *décomposition de Zeckendorf de n*.

Par exemple $17 = 13 + 3 + 1 = F_7 + F_4 + F_2$ donc $k = 3$ et $\{c_1, c_2, c_3\} = \{2, 4, 7\}$.
Le but de cet exercice est d'écrire un programme qui renvoie cette décomposition pour un nombre n pris en argument.

1. Écrire une fonction récursive `Fibonacci(k):` qui renvoie le terme F_k de la suite (F_n).

2. Écrire une fonction récursive `Zeckendorf(n):` qui renvoie la décomposition de Zeckendorf de n.
On n'oubliera pas de réordonner la liste pour que les termes soient dans l'ordre croissant.

Un caractère est représenté en ASCII par un nombre. Par exemple, le caractère « A » est représenté par le nombre 65, « B » par 66, etc.. La commande `ord(c)` renvoie la valeur du nombre qui code le caractère `c` en ASCII. Réciproquement, la commande `chr(n)` renvoie le caractère codé par n. Par exemple

```
>>>ord('A')
65
>>>ord(' ') # pour l'espace
32
>>> chr(65)
'A'
```

Le théorème de Zeckendorf permet alors d'effectuer le *codage de Fibonacci* d'un nombre et donc d'un caractère. Plus précisément, connaissant la décomposition $Z_n = \{c_1, .., c_k\}$ de Zeckendorf d'un entier n, on peut écrire

$$n = \sum_{i=1}^{k} \alpha_i F_{c_i}, \qquad \text{où} \qquad \alpha_i = \begin{cases} 1, & \text{si } c_i \in Z_n \\ 0, & \text{sinon.} \end{cases}$$

ce qui va permettre de coder chaque caractère avec une suite de 0 et de 1.
Le théorème assure notamment que le codage ne peut pas comporter deux 1 côte-à-côte (il n'y a jamais deux nombres de Fibonacci consécutifs dans la décomposition). Ainsi, pour permettre de détacher les lettres en identifiant la fin du codage d'un caractère, on rajoute un 1 à la fin de la décomposition.
Par exemple, comme $65 = F_3 + F_6 + F_{10}$, le caractère « A » est codé par 000100100011.

3. Écrire des fonctions de codage et de décodage (selon le code de Fibonacci) d'une chaîne de caractères.
Par exemple

```
>>> codage_fibonacci('AMICALEMENT')
'0001001000110010000010110000010100110010
1000110001001000110000000010110010000100110
0100000101100100000100110001000010110000001
01011'
```

4. Décoder le message

```
'0010101000110010101000011000010101100000010101100101000101100
0010101100101010000110000010010011001001001001100000101011000
1001001100010100100110001000010011001010100001100101000100110
0010101100101011'
```

Exercice 7. Matrices stochastiques

On dit qu'une matrice carrée $M \in \mathcal{M}_n(\mathbb{R})$ est *stochastique* si tous ses coefficients sont positifs ou nuls et si, la somme de chacune de ses lignes vaut 1.
Écrire une fonction `def test_stochastique(M)` qui prend en argument une matrice M (implémentée sous la forme d'un tableau `ndarray` 2D) et qui renvoie `True` ou `False` selon que celle-ci est stochastique ou non.

On pourra commencer par écrire une fonction récursive `somme(L)` qui renvoie la somme des termes d'une liste.

Exercice 8. Tri à bulles ou *bubble sort*

L'algorithme de tri à bulles n'est, en pratique, guère utilisé. C'est néanmoins un exemple classique, car il est particulièrement formateur. Le principe sous-jacent est de faire remonter progressivement vers la droite les plus grands termes de la liste considérée, à la manière des plus grosses bulles d'air qui sont les premières à remonter à la surface d'un liquide.

Plus précisément, le principe est le suivant:

i. on parcourt les éléments k premiers éléments de la liste (en commençant avec k qui vaut la longueur de la liste), et à chaque fois qu'un élément est plus grand que le suivant, on les échange (à l'issue de cette opération, le plus grand élément est alors en dernière position);
ii. on recommence l'opération précédente, en ne considérant pas le dernier élément de la liste, puis en ne considérant pas les deux derniers éléments etc... jusqu'à ce que la liste soit entièrement triée.

1. Écrire une procédure Python qui implémente le tri à bulles en place. On commencera par écrire une procédure `remonte` qui implémente l'étape *i*.
2. Combien de comparaisons sont nécessaires lors de l'exécution de cet algorithme? Quelle est la complexité du tri à bulles?
3. Modifier la procédure pour obtenir une complexité en $O(n)$ dans le meilleur des cas.
4. Application: en déduire un programme calculant la médiane d'une série de données contenue dans une liste.

Exercice 9. Tri par dénombrement ou *counting sort*

Soit $N \in \mathbb{N}^*$. On cherche à trier une liste `L` d'entiers positifs strictement inférieurs à N.
Certaines valeurs peuvent apparaître plusieurs fois dans la liste initiale, et devront donc apparaître autant de fois dans la liste triée.

1. Écrire une fonction `Comptage`, d'arguments `L` et `N`, renvoyant une liste `P` de longueur N dont le k-ième élément est le nombre d'occurrences de l'entier k dans `L`.
2. En utilisant la fonction `Comptage`, écrire une fonction `Tri_par_denombrement`, d'arguments `L` et `N`, renvoyant une liste correspondant au tri de `L`.
3. Quelle est la complexité de cette fonction de tri (en fonction de la longueur n de `L`)? Commenter.