TD 3 - Boucles multiples - Algorithmes de tri

Mise en route

Rapidement :

Ecrire une fonction `recur3` qui à un entier n renvoie v_n, la suite réelle v étant définie par :

$$\begin{cases} v_0 = 0, v_1 = 1, v_2 = 3 \\ \forall n \in \mathbb{N}, v_{n+3} = 4v_{n+2} - 5v_{n+1} + 2v_n \end{cases}.$$

Boucles multiples

1 – Ecrire une fonction `sommesimple` qui à un entier n renvoie $\displaystyle\sum_{i=1}^{n} \frac{(i+3)^2}{i+2}$.

2 – Ecrire une fonction `sommedouble` qui à un entier n renvoie $\displaystyle\sum_{1\leqslant i,j\leqslant n} \frac{(i+j)^2}{i^2+j^2} = \sum_{i=1}^{n}\sum_{j=1}^{n} \frac{(i+j)^2}{i^2+j^2}$.

3 – Ecrire une fonction `sommeproduit` qui à un entier n renvoie $\displaystyle\sum_{i=1}^{n}\prod_{j=1}^{n} \frac{(i+j)^2}{i^2+j^2}$.

4 – On veut écrire une fonction `sommetriangulaire` qui à un entier n renvoie $\displaystyle\sum_{1\leqslant i\leqslant j\leqslant n} \frac{(i+j)^3}{i^3+2j^3}$.

On utilisera dans une première version le fait que $\displaystyle\sum_{1\leqslant i\leqslant j\leqslant n} \frac{(i+j)^3}{i^3+2j^3} = \sum_{i=1}^{n}\sum_{j=i}^{n} \frac{(i+j)^3}{i^3+2j^3}$,

puis dans une deuxième version le fait que $\displaystyle\sum_{1\leqslant i\leqslant j\leqslant n} \frac{(i+j)^3}{i^3+2j^3} = \sum_{j=1}^{n}\sum_{i=1}^{j} \frac{(i+j)^3}{i^3+2j^3}$.

Listes

Voici quelques notions sur les listes :

- On dispose sur le type `list` des opérateurs de concaténation `+` et de concaténation itérée `*`.
- La principale méthode sur les listes qu'on emploiera est `append`. Si `toto` est une liste, `toto.append` est une fonction et pour `a` un objet `toto.append(a)` rajoute l'élément `a` à `l`, en dernier élément.
- Une liste `l` de longueur `n` (`len(l)`) admet comme indices les entiers entre `0` et `n-1`, mais également les négatifs entre `-1` et `-n`, ce qui permet d'accéder aux éléments par l'extrêmité droite de la liste (`l[-1]` est le dernier terme de la liste, `l[-2]` l'avant-dernier,...).
- `l[i:j]` désigne la sous-liste de `l` constituée des éléments de `l` d'indices compris entre `i` et `j-1` (pour `i<j` de même signe)
- `l[:j]` correspond à `l[0:j]` pour `j`$\geqslant$`0` et à `l[-n:j]` pour `j<0`.
- `l[i:]` correspond à `l[i:n]` pour `i`$\geqslant$`0` et à `l[i:0]` pour `j<0`.
- `l[:]` correspond à `l[0:n]`, c'est donc une sous-liste triviale, avec les mêmes valeurs que `l`.

Le tri par sélection

On considère une liste `l` dont les éléments sont d'un type dont les valeurs peuvent être comparées par l'opérateur `<`. Le but est de modifier cette liste de manière à ce qu'elle contienne les mêmes éléments avec la même occurrence, mais triés dans l'ordre croissant.

Du fait qu'on modifie la liste, plutôt que de créer une autre liste sans changer la liste en argument, on parle de tri "sur place" ou "en place".

Le tri par sélection consiste à parcourir la liste pour trouver le minimum, dont on échange la place avec la valeur en début de liste, puis à reparcourir le reste de la liste pour réitérer l'opération.

Plus précisément :

- on cherche le minimum de `l[0:]`, puis on le place à l'indice `0` (on échange deux valeurs de la liste, il nous faut donc la position du minimum dans la liste)
- on cherche ensuite le minimum de `l[1:]`, puis on le place à l'indice `1`.
- etc

Exemple : Au cours des opérations successives, la liste `[5,3,2,1]` passe par les étapes :

```
[1,3,2,5]
[1,2,3,5]
[1,2,3,5]
```

1 – Ecrire une fonction `tri_selec` correspondante.

2 – Démontrer la terminaison et la validité de cette fonction

3 – Discuter de façon précise de la complexité (des complexités) de cette fonction.

Autres tris

Tri à bulles

Le tri à bulles consiste à parcourir la liste et à échanger deux éléments consécutifs quand ils ne sont pas dans le bon ordre. On reparcourt le tableau jusqu'à ce que le tableau soit trié. On voit donc les éléments les plus grands migrer progressivement vers la fin du tableau, d'où le nom de l'algorithme par analogie avec des bulles qui remontent progressivement à la surface d'un liquide.

Plus précisément :

- On parcourt la liste `l[:n]` avec l'indice `j`. Si `l[j+1]<l[j]`, on échange `l[j]` et `l[j+1]`.
  Une fois le parcours terminé, la plus grande valeur de `l[:n]` se retrouve à l'indice `n-1`.
- On parcourt ensuite la liste `l[:n-1]` avec l'indice `j`. Si `l[j+1]<l[j]`, on échange `l[j]` et `l[j+1]`.
  Une fois le parcours terminé, la plus grande valeur de `l[:n-1]` se retrouve à l'indice `n-2`.
- etc

Exemple : Au cours des opérations successives, la liste `[5,3,2,1]` passe par les étapes :

```
[3,5,2,1]          [2,3,1,5]          [1,2,3,5]
[3,2,5,1]          [2,1,3,5]
[3,2,1,5]
```

1 – Ecrire une fonction `tri_bulles` correspondante.

2 – Démontrer la terminaison et la validité de cette fonction

3 – Discuter de façon précise de la complexité (des complexités) de cette fonction.

Variante : Lorsqu'au cours d'un parcours de la liste, on n'a pas besoin d'intervertir des valeurs, c'est que le tableau est déjà trié, et on peut donc arrêter la procédure. Modifier le code de `tri_bulles` en introduisant un compteur booléen de manière à arrêter l'exécution du code dès que le tableau est trié.

Tri par insertion

Le tri par insertion d'une liste `l` consiste à trier `l[:2]`, puis à trier `l[:3]` et ainsi de suite.

Quand `l[:i]` est trié, il suffit, pour trier `l[:i+1]`, de faire "reculer" `l[i]` jusqu'à ce qu'il trouve sa place. On l'aura donc inséré dans la sous-liste le précédant, d'où le nom du tri.

Pour faire "reculer" `l[i]`, on l'échange avec son prédécesseur jusqu'à ce qu'on rencontre une valeur plus petite.

Exemple : Au cours des opérations successives, la liste `[5,3,2,1]` passe par les étapes :

```
[3,5,2,1]        [3,2,5,1]        [2,3,1,5]
                 [2,3,5,1]        [2,1,3,5]
                                  [1,2,3,5]
```

1 – Ecrire une fonction `tri_selec` correspondante.

2 – Démontrer la terminaison et la validité de cette fonction

3 – Discuter de façon précise de la complexité (des complexités) de cette fonction.

Conclusion

Comparer les algorithmes de tri vus dans ce TD.