TP4 : tris naïfs

AP3 Algorithmique et programmation — L2 mathématiques

20 octobre 2016

Pour tester les fonctions différentes fonctions de tri codées dans ce TP, on pourra générer des listes aléatoires. Une méthode pour cela est la suivante :

```
import random # library with function about randomization

# random contains a function randrange(n,m) returning a random
    integer between n and m-1
li = [random.randrange(1,100) for i in range(20)]
```

Exercice 1 (Tri par sélection)**.**

(1) Écrire une fonction `find_max(li,d,f)` qui prend en paramètres une liste `li`, et des indices `d` et `f` tels que $0 \leq$ `d` $\leq$ `f` $\leq n-1$ où n est la longueur de `li` et renvoie l'indice d'un maximum de la liste `li` entre les indices `d` et `f`.

(2) S'aider de la fonction ci-dessus pour écrire une fonction `select_sort(li)` qui trie la liste `li` par la méthode de tri par sélection vue en cours. On rappelle rapidement en quoi consiste la méthode : chercher le maximum d'une liste ; l'insérer à la dernière place ; itérer ce principe sur la sous-liste privée de sa dernière position.

Exercice 2 (Tri bulle)**.**

(1) Coder une fonction `bubble_sort(li)` prenant en paramètre une liste `li` et renvoyant la liste triée par la méthode du tri bulle. On rappelle que la méthode du tri bulle s'appuie sur la même idée que le tri par sélection : mettre le maximum en bout de liste et recommencer sur la sous-liste privée de sa dernière position. En revanche, on ne séléctionne pas explicitement le maximum : on observe les éléments en position 0 et 1, puis 1 et 2, etc. ; à chaque fois, si celui de gauche est plus grand que celui de droite, on les échanges ; ce faisant, une fois la liste épuisée, le maximum se trouve en dernière position.

(2) Se souvenir de la position du dernier échange réalisé entre deux valeurs dans la boucle interne de l'algorithme permet éventuellement d'économiser quelques étapes. Pourquoi ?

Coder une fonction `bubble_sort_opt(li)` prenant en compte cette optimisation.

Exercice 3 (Tri par insertion)**.** Soit `li` une liste de longueur n. Le tri par insertion est une variante des tris précédents. Il consiste, à chaque phase $i \leq n-1$, à insérer la valeur d'indice i à sa place parmi les valeurs d'indices 0 à $i-1$ qui sont supposées

être déjà triées en ordre croissant mais ne sont pas nécessairement les $i-1$ plus petites valeurs de la liste.

Détaillons un peu. Une liste à un seul élément est triée par définition. Pour $i=2$, on cherchera à positionner `li[1]` correctement vis à vis de `li[0]` de façon à obtenir une liste triée constituée de ces deux éléments. Plus généralement, pour $i \leq n-1$, on supposera que les $i-1$ premiers éléments de `li` ont été ré-ordonnés de façon à ce que l'on ait `li[0]` $\leq$ `li[1]` $\leq \ldots \leq$ `li[i-1]` ; et on insère `li[i]` à sa place dans cette liste. Lorsqu'on arrive à $i=n$, on a `li[0]` $\leq$ `li[1]` $\leq \ldots \leq$ `li[n-1]`, c'est-à-dire que la liste est triée.

Toute la difficulté est alors dans l'*insertion* de `li[i]` dans la sous-liste triée `li[:i]` : on compare `li[i]` et `li[i-1]` et on les échange si besoin ; puis on recommence avec `li[i-1]` et `li[i-2]` etc. jusqu'à ce qu'un échange ne soit plus nécessaire. L'élément initialement en position i est alors inséré à la bonne position, faisant de la sous-liste `li[:i+1]` une liste triée.

(1) Écrire une fonction `insert_sort(li)` prenant en paramètres une liste `li` implémentant le tri par insertion rappelé ci-dessus.

(2) On peut améliorer légèrement la fonction ci-dessus en insérant l'élément `li[i]` dans `li[:i]` par recherche dichotomique. On rappelle que la recherche dichotomique **dans une liste triée** se fait comme suit : on a un élément à insérer à sa place dans la liste triée ; on le compare à l'élément du milieu(*-ish*) de la liste ; s'il est plus petit, on répète récursivement l'opération sur la sous-liste de gauche, sinon sur celle de droite ; les cas d'arrêt sont le cas d'une liste vide et le cas où l'on tombe tout pile sur la valeur qu'on essaye d'insérer.

(3) Cette optimisation peut être utile si une comparaison est coûteuse par rapport à l'échange de deux éléments. Qu'entendons-nous par cette phrase ?

À rendre pour le 27 octobre 2016

Pour une fois : rien ! Mais je vous conseille fortement de regarder le tri par insertion et d'essayer de l'implémenter car on ira peut-être un peu vite sur celui-ci la fois prochaine. En effet, on devra aussi couvrir les tris *évolués* que sont le tri fusion et le tri par tas, ce qui représente déjà une bonne masse de travail.