Implémentation des tris

mai 2019

1 Tri par sélection

Exercice 1 (Récursivité)

Écrire le tri par sélection du minimum de façon entièrement récursive : votre code ne doit comporter aucune boucle !

Exercice 2 (Validation d'un programme de tri)

Écrire une fonction `oracle_tri` *qui prend en argument :*

- *une fonction de tri (eh oui, on peut donner des fonctions en argument d'autres fonctions !)*
- *une liste*

qui exécute la fonction de tri sur cette liste, et vérifie que le résultat obtenu est conforme (qu'il s'agit bien des mêmes éléments en ordre croissant).

Exercice 3 (Génération de tests)

Écrire différentes fonctions générant des listes à trier :

- *déjà en ordre croissant, ou presque*
- *en ordre décroissant*
- *en ordre aléatoire*
- *avec ou sans doublons*
- *...*

Chacune de ces fonctions pourra être paramétrée par la taille de la liste à générer.

Exercice 4

Testez votre tri par sélection récursif sur une grosse *liste : que constatez-vous ? Quelle leçon en tirez-vous quant aux algorithmes récursifs en général ?*

Pensez à réutiliser vos méthodes de validation pour les autres tris de ce TP.

Vous pouvez même écrire une fonction qui automatise tout le processus : génération de listes de différentes formes et taille, puis test systématique avec l'oracle et affichage des éventuels cas problématiques.

2 Tri par fusion

Exercice 5

Implémentez un tri par fusion de la façon la plus simple et lisible possible. Vous pouvez utiliser les facilités de Python : tranchage de liste, recopie de liste, etc.

Exercice 6

Améliorez le programme précédent pour éviter de réallouer un nouveau tableau temporaire à chaque appel de `fusion` *:*

- *Toutes vos fonctions devront prendre en argument supplémentaire la liste temporaire ; elles ne seront jamais appelées directement par l'utilisateur.*
- *La fonction principale* `tri_fusion` *se contente de créer la liste temporaire (une copie de la liste à trier, par exemple), puis de faire le premier appel à la fonction auxiliaire qui prend cette liste temporaire en argument.*

Exercice 7

Améliorez encore le programme précédent pour que la fusion ne fasse qu'une seule recopie des éléments de la portion à fusionner : on jouera donc sur les deux listes reçues en arguments, l'une servant à recevoir les données et l'autre à construire le résultat.

3 Tri par tas

Pour ce TP, on considérera des tas dans lesquels c'est la valeur *minimale* qui est placée à la racine.

Exercice 8 (Structure de tas)

Programmer les fonctions suivantes, dans lequelles l'argument `T` *est un tas représenté sous forme d'une liste Python :*

1. `tas_vide()`
   renvoie un nouveau tas ne contenant aucun élément.
2. `inserer(x, T)`
   modifie `T` *pour y insérer la valeur* `x`*, de façon à conserver une structure de tas.*
3. `minimum(T)`
   renvoie l'élément minimal de `T`*, sans modifier ce dernier.*
4. `extraire_minimum(T)`
   renvoie l'élément minimal de `T`*, supprime cette valeur de* `T` *et le réordonne de façon à conserver une structure de tas.*

Exercice 9 (Opérations avancées)

Programmer les fonctions suivantes, dans lequelles l'argument `L` *est une liste Python* quelconque *:*

1. `tassifier(L)`
   modifie la liste `L` *de façon à ce qu'elle devienne un tas, contenant le même ensemble d'éléments*
   Pour les courageux : *il est possible de réaliser cette opération en place et en temps linéaire !*
2. `tri_par_tas(L)`
   trie la liste `L` *en deux temps :*
   (a) *transformation en tas*
   (b) *extraction successive des éléments minimaux de façon à reconstituer la liste triée*

Exercice 10 (Un peu de programmation orientée objet)

Construire une classe `Tas`*, en utilisant les fonctions programmées précédemment de façon pertinente.*