CPES Scientifique

Informatique

TP 8

Quelques tris

1 Algorithmes de tri

1.1 Tri par propagation (tri « à bulles »)

Le « tri à bulles » est un algorithme de tri relativement naïf. Son principe est le suivant. On se donne une liste `x`. En utilisant la fonction « `liste_triee` » du TP 07, on peut déterminer si `x` est triée. Si ce n'est pas le cas, il existe un indice i de `x` tel que $\texttt{x}[i] > \texttt{x}[i+1]$. Si on échange les éléments d'indices i et $i+1$ de `x`, alors on obtient une liste qui est « mieux triée ». On peut à nouveau tester si cette liste « mieux triée » est réellement triée, et si ce n'est pas le cas, répéter le même processus.

Question 1.1. *Écrire une fonction Python qui prend argument une liste d'entiers* `x` *et qui renvoie une liste triée par ordre croissant en utilisant le principe ci-dessus. On pourra utiliser la fonction «* `liste_triee` *» du TP 07.*

Il existe un moyen d'améliorer le principe ci-dessus. Notons que chaque appel à la fonction « `liste_triee` » sur une liste `x` coûte de l'ordre de `len(x)` opérations. De plus, lorsque la liste n'est pas triée, la recherche d'un indice i tel que $\texttt{x}[i] > \texttt{x}[i+1]$ peut coûter de l'ordre de `len(x)` opérations. Au lieu d'utiliser explicitement la fonction « `liste_triee` », on peut en un seul parcours à la fois tester si la liste est triée, et effectuer les permutations des $\texttt{x}[i]$ et $\texttt{x}[i+1]$ tels que $\texttt{x}[i] > \texttt{x}[i+1]$.

D'autre part, l'ordre de parcours de `x` peut permettre d'économiser un certain nombre d'opérations.

Question 1.2. *Considérons la fonction suivante :*

```
def f(x):
  i = 0
  while (i < len(x)-1):
    if (x[i+1] < x[i]) :
      tmp = x[i]
      x[i] = x[i+1]
      x[i+1] = tmp
    i = i+1
```

Après l'exécution de `f` *sur une liste* `y`*, que contient* $\texttt{y}[\texttt{len(y)} - 1]$ *?*

Question* 1.3. *Écrire une fonction Python qui prend argument une liste d'entiers* `x` *et qui renvoie une liste triée par ordre croissant en utilisant le principe ci-dessus,* **et qui n'utilise pas la fonction « `liste_triee` » du TP 07**.

1.2 Tri par sélection

L'idée du tri par sélection est la suivante. Soit `v` un élément maximal d'une liste `x`. Alors la liste obtenue en échangeant `v` avec le dernier élément de `x` est « mieux triée » que `x`. On peut répéter le même procédé sur cette nouvelle liste. Mais attention, `v` étant l'élément maximal de `x`, une fois placé à la fin, on sait qu'il est à la bonne position. On ne doit donc pas chercher l'élément maximal de la nouvelle liste, mais l'élément maximal de la nouvelle liste **privée de son dernier élément**.

Il est utile de se donner la variante suivante de la recherche d'élément maximal :

Question 1.4. *Écrire une fonction qui prend en arguments une liste* `x` *et un entier* n *et qui cherche l'indice du plus grand élément de la liste* $[\texttt{x}[0]\ ,\ \ldots\ ,\ \texttt{x}[m-1]]$ *où* $m = \min(n, \texttt{len}(\texttt{x}))$.

Considérons une application de l'idée ci-dessus sur la liste

$$[9\ ,\ 4\ ,\ 7\ ,\ 2\ ,\ 0]$$

On a, au départ :

$$\begin{array}{l}[\ 9\ ,\ 4\ ,\ 7\ ,\ 2\ ,\ 0\ ] \\ \ \ \uparrow \\ i\_max\end{array}$$

ce qui nous amène à la liste

$$\begin{array}{l}\qquad\qquad\qquad\qquad\qquad\ \ \downarrow \\ [\ 0\ ,\ 4\ ,\ 7\ ,\ 2\ ,\ 9\ ] \\ \qquad\quad\ \uparrow \\ \qquad i\_max\end{array}$$

Dans cette nouvelle liste, i_max est l'indice de l'élément maximal de la liste $[0, 4, 7, 2]$ (qui correspond à la partie de la liste ci-dessus strictement à gauche de la flèche $\downarrow$).

En itérant, on obtient :

$$\begin{array}{l}\qquad\qquad\qquad\ \ \downarrow \\ [\ 0\ ,\ 4\ ,\ 2\ ,\ 7\ ,\ 9\ ] \\ \qquad\ \uparrow \\ \quad i\_max\end{array}$$

L'étape suivante nous donne

$$\begin{array}{l}\qquad\qquad\ \ \downarrow \\ [\ 0\ ,\ 2\ ,\ 4\ ,\ 7\ ,\ 9\ ] \\ \qquad\ \uparrow \\ \quad i\_max\end{array}$$

et enfin

$$\begin{array}{l}\qquad\ \ \downarrow \\ [\ 0\ ,\ 2\ ,\ 4\ ,\ 7\ ,\ 9\ ] \\ \ \ \uparrow \\ i\_max\end{array}$$

Question 1.5. *Écrire une fonction Python qui implémente un algorithme de tri selon le principe ci-dessus.*

1.3 Tri par insertion

L'idée du tri par insertion est de parcourir la liste à trier, et de la modifier au fur et à mesure, de telle sorte que la partie à gauche de la position courante soit toujours triée. Pour cela, en parcourant la liste, on insère l'élément courant à sa bonne place dans la partie gauche de la liste.

Cette insertion est réalisée au moyen d'une boucle. Supposons que i est la position courante dans `x` et que `v` est l'élément de `x` à la position i. Pour $j < i$, si $\texttt{v} < \texttt{x}[j]$ alors `v` doit être placé à gauche de la position d'indice j, et on continue à parcourir `x` vers la gauche, jusqu'à arriver soit à $j = 0$, soit à un j tel que $\texttt{x}[j] \leq \texttt{v}$. Dans les deux cas, on a trouvé la bonne place pour `v`.

Question* 1.6. *Écrire une fonction Python qui implémente un algorithme de tri selon le principe ci-dessus.*