TD 13 Tris de listes

Lycée Louis Thuillier - Informatique - PCSIB -

Comme nous l'avons étudié dans l'exemple précédent, il est souvent opportun de travailler avec une liste (ou un tableau à un dimension) **triée**. Par exemple, la recherche d'un élément dans la liste s'en trouve alors grandement accélérée. Il est donc nécessaire de savoir trier une liste.

Il existe bien évidemment des méthodes en langage Python qui permettent de faire cela. Citons par exemple *sort*() sur les listes.
Dans tout ce TD nous nous interdisons d'utiliser les méthodes de tri de Python : l'objectif est de comprendre comment ces dernières marchent.

```
>>> L=[2,5,1,8,99,3]

>>> L.sort()

>>> L
[1, 2, 3, 5, 8, 99]
```

L'objectif est donc double :
1. comprendre comment fonctionne chaque algorithme de tri
2. analyser leur différence et les discuter en comparant les complexités de chaque algorithme

Le tri par insertion

Fonction et procédure

On distingue en langage Python les "fonctions" et les "procédures" :

- **une fonction renvoie une valeur** : la liste triée s'obtient alors par affectation

```
>>> liste_triee = fonction_tri(liste_depart)
```

- **une procédure modifie la liste entrée en argument et ne renvoie rien** : la liste triée s'obtient juste en appliquant la procédure

```
>>> procedure_tri(liste_depart)
```

Attention ! Les deux (procédures et fonctions) sont des fonctions Python : elles n'implémentent de la même façon.

> *Application 1 :* ***Pour cette question seulement****, en utilisant la méthode sort() écrire la fonction fonction_tri et la procédure procedure_tri.*

Principe du tri par insertion

Le tri par insertion est la méthode la plus naturelle de tri :

- ▷ on sélectionne le premier élément de la liste à trier, et on le place dans une liste initialement vide
- ▷ on sélectionne le deuxième élément, puis on le compare au précédent afin de le mettre à sa place dans la sous-liste composé du premier élément précédent :
  - ▷ s'il est plus grand, on le met en dernière position
  - ▷ s'il est plus petit, on le met en première position

  La liste de deux élément ainsi créer est donc bien triée !
- ▷ on continue la méthode avec le troisième élément, en le plaçant au bon endroit de la liste triée
- ▷ ...

```
[2, 5, 1, 8, 99, 3]          []
[2, 5, 1, 8, 99, 3]      2   []
[5, 1, 8, 99, 3]             [2]
[1, 8, 99, 3]            5   [2]
[1, 8, 99, 3]                [2,5]
[8, 99, 3]               1   [2,5]
[8, 99, 3]                   [1,2,5]
[99, 3]                  8   [1,2,5]
[99, 3]                      [1,2,5,8]
[3]                      99  [1,2,5,8]
[]                       3   [1,2,5,8,99]
[]                           [1,2,3,5,8,99]
```

Vocabulaire : "insérer un élément à sa place" signifie ajouter l'élément à la liste tout en conservant le fait que la liste soit triée.

► Fonction *insere*

Afin de construire la fonction tri basée sur l'algorithme décrit précédemment, il semble utile de créer une **procédure** $inserer_triee$ qui prend en entrée un élément e et une liste triée L et place l'élément e "à sa place" dans la liste L.
Pour cela on utilisera la méthode $insert(k, e)$ qui permet d'insérer l'élément e dans la liste L afin qu'il ait l'indice k.

> ***Application 2 :*** *Écrire la procédure inserer_triee.*

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

▶ Tri par insertion

> ***Application 3 :*** *Écrire alors une fonction tri_insertion qui prend en entrée une liste L non triée et qui renvoie une liste contenant les éléments de L triés.*

.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....

Complexité de tri par insertion

> ***Application 4 :*** *Estimer la complexité de la fonction tri_insertion. On prendra comme d'ordinaire le pire des cas.*

Propriété. **Complexité d'un tri par insertion** La complexité d'un tri par insertion est ..... , avec n la taille de la liste à triée.

▶ (Pour aller plus loin) Méthode et fonction

L'inconvénient de cette méthode de tri est qu'elle ne trie pas vraiment la liste L : la fonction *tri_insertion* est une fonction et non une méthode, elle crée une nouvelle liste et ne modifie pas la liste en entrée.

On propose alors la méthode suivante pour trier une liste L :
- ▷ on parcourt un à un tous les élément e de la liste L
- ▷ pour chaque élément, on va le comparer à l'élément le précédent et, si ils ne sont pas ordonnées, les inverser
- ▷ on répète le processus précédent jusqu'à ce que l'élément e soit à la "bonne place"

Application 5 : Compléter le code suivant pour la méthode de tri par insertion.

```
def tri_insertion_methode(n):
        n=len(L)
        for i in range(n):
                j = ..............

                while j > ................. and ........................... :

                        L[j],L[j-1] = ........................

                        j = ............................
```

Le tri fusion

Le tri fusion est un autre moyen de trier des listes d'éléments. Il peut paraître moins naturel que le tri par insertion mais il a l'avantage d'être plus efficace : sa complexité augmente plus lentement avec n, taille de la liste à trier.

Principe du tri fusion

Le tri fusion se base sur un principe récurent en algorithmie : un problème initial portant sur n données (ici la liste à trier) va être séparé en deux sous-problèmes portant chacun sur $n/2$ données. Pour un tri à fusion on va alors :
- ▷ séparer la liste initiale en deux sous-listes contenant chacune moitié moins d'éléments
- ▷ trier les deux sous-listes
- ▷ fusionner les deux listes triées en une seule

On soulève alors une question légitime ; comment va-t-on alors trier les deux sous-listes créées ? L'astuce est qu'on va les trier de la même façon que la liste originale : on va les séparer en deux, trier les deux plus petites listes et les fusionner. C'est ce qu'on appelle un **algorithme récursif** : l'algorithme s'utilise lui même pour fonctionner.

Fusion de listes triées

L'efficacité du tri fusion repose sur l'efficacité de la fusion entre deux listes triées. On souhaite créer une liste triée L contenant les éléments des deux sous listes triées L_1 et L_2, de longueur respectives n_1 et n_2. Voici le principe :
- ▷ les nombres i et j vont permettre de sélectionner les éléments des liste L_1 et L_2
- ▷ au début i et j sont nuls et $L = [\,]$
- ▷ tant que $i < l_1$ et $j < l_2$ on compare $L_1[i]$ à $L_2[j]$
  - ▷ si $L_1[i] < L_2[j]$ alors on ajoute $L_1[i]$ à la fin de L et on incrémente i de 1
  - ▷ sinon on ajoute $L_2[j]$ à la fin de L et on incrémente j de 1

- ▷ si $i = l_1$, on ajoute alors tous les éléments restants de L_2 à la fin de L
- ▷ si $j = l_2$, on ajoute alors tous les éléments restants de L_1 à la fin de L

Application 6 : *Appliquer à la main l'algorithme de fusion de liste sur deux liste* $[1, 3, 4, 8, 12]$ *et* $[0, 5, 6]$.

Application 7 : *Ecrire une fonction $fusion$ qui prend en entrée deux listes triées L_1 et L_2 et qui renvoie une liste triée, fusion des deux listes L_1 et L_2.*

Tri fusion et complexité

A l'aide de la fonction $fusion$ précédente, il est aisé de construire la fonction tri_fusion de manière récursive :

Application 8 : *Ecrire une fonction tri_fusion récursive qui prend en entrée une liste L et qui renvoie une liste contenant les éléments de L triés. On rappelle l'algorithme récursif :*
- ▷ *séparer la liste initiale en deux sous-listes contenant chacune moitié moins d'éléments*
- ▷ *trier les deux sous-listes*
- ▷ *fusionner les deux listes triées en une seule*

Il est difficile d'expliquer simplement la complexité du tri fusion. On retiendra la propriété suivante.

Propriété. Complexité du tri fusion
La complexité d'un tri fusion est quadratique $\sim n \ln n$, avec n la taille de la liste à triée.