Chap 17. Algorithmes de tri

Livre p 143 – Chap 10 Algorithmes de tri

1. Introduction

a. Besoins

Lorsque l'on récolte des informations de même nature, on peut les stocker dans une liste.
Pour traiter les informations récoltées il est souvent plus pratique que cette liste soit triée (dans l'ordre croissant, décroissant, alphabétique, …)

b. Opérations

Pour des raisons d'efficacité et de gestion de la mémoire, on considère généralement qu'on ne va pas créer une nouvelle liste ordonnée mais que l'on va modifier progressivement la liste de départ afin qu'elle soit ordonnée à la fin du processus.
On a alors différentes opérations disponibles :

- Parcourir la liste (dans un sens ou dans l'autre)
- Comparer deux éléments de la liste
- Echanger deux éléments de la liste

2. Tri par sélection

a. Algorithme

On va chercher successivement, de gauche à droite :

- Le plus petit élément à partir du début, une fois trouvé on l'échange avec le premier élément.
- Le deuxième plus petit élément à partir du deuxième élément (car on sait déjà que le premier est le plus petit !), une fois trouvé on l'échange avec le deuxième élément.
- Le troisième plus petit élément à partir du troisième élément (car on sait déjà que les deux premiers sont les plus petits !), une fois trouvé on l'échange avec le troisième élément.
- Ainsi de suite, on cherche le quatrième plus petit, le cinquième, … jusqu'au dernier à droite !

Remarque : Lorsque la partie droite ne contient plus qu'un seul élément, c'est lui le plus grand !
Avantages : A chaque étape, la partie gauche de la liste est parfaitement triée de manière définitive.
La partie droite, que l'on doit parcourir, contient de moins en moins d'éléments.
On doit effectuer, en moyenne, assez peu d'échanges.
Inconvénients : On doit parcourir la partie droite de la liste jusqu'au dernier élément à chaque étape.
On doit effectuer, en moyenne, énormément de comparaisons.

b. Exemple

Prenons la liste a = [*2, 7, 5, 1, 6*]
1ère étape : On parcourt les éléments 2, 5, 7, 1, 6, c'est « 1 » le plus petit… on l'échange avec le « 2 »
Nouvelle version de la liste : a = [**1**, *7, 5, 2, 6*]
2ème étape : On parcourt les éléments 7, 5, 2, 6, c'est « 2 » le plus petit… on l'échange avec le « 7 »
Nouvelle version de la liste : a = [**1, 2**, *5, 7, 6*]
3ème étape : On parcourt les éléments 5, 7, 6, c'est « 5 » le plus petit… il est déjà à sa place !
« Nouvelle version » de la liste : a = [**1, 2, 5**, *7, 6*]
4ème étape : On parcourt les éléments 7, 6, c'est « 6 » le plus petit… on l'échange avec le « 7 »
Nouvelle version de la liste : a = [**1, 2, 5, 6**, *7*]
5ème étape : Le dernier élément est forcément le plus grand, a = [**1, 2, 5, 6, 7**]… la liste est triée !

3. Tri par insertion

a. Algorithme

On va parcourir la liste de gauche à droite en prenant les éléments un par un à partir du 2ème élément. Chaque l'élément va être comparé avec la partie de la liste située à sa gauche, afin de l'insérer à sa place. Pour insérer l'élément à sa place, on parcourt la partie gauche de la droite vers la gauche (du plus grand au plus petit) en échangeant à chaque fois l'élément sélectionné avec celui auquel on le compare si l'élément sectionné est le plus petit, sinon on s'arrête là pour cet élément car il est alors à sa place !
Remarque : On n'effectue aucune action sur le premier élément de la liste car il n'y a rien à sa gauche…
Avantages : A chaque étape, la partie gauche de la liste est triée mais pas de manière définitive. Si la liste est déjà partiellement triée, cette méthode est très rapide.
Inconvénients : On doit effectuer, en moyenne, beaucoup de comparaisons et d'échanges.

b. Exemple

Prenons la liste a = [*2, 7, 5, 1, 6*]
1ère étape : On garde le 1er élément à sa place, il constitue le début de la liste triée, a = [**2**, *7, 5, 1, 6*]
2ème étape : On prend l'élément suivant « 7 »… il est à sa place par rapport au seul élément trié : « 2 » « Nouvelle version » de la liste : a = [**2, 7**, *5, 1, 6*]
3ème étape : On prend l'élément suivant « 5 »… on doit l'insérer entre le « 2 » et le « 7 » en effectuant un échange avec le 7 ». Nouvelle version de la liste : a = [**2, 5, 7**, *1, 6*]
4ème étape : On prend l'élément suivant « 1 »… on l'échange successivement avec le « 7 », le « 5 » puis le « 2 » pour le mettre à sa place. Nouvelle version de la liste : a = [**1, 2, 5, 7**, *6*]
5ème étape : On prend le dernier élément « 6 »… on doit l'insérer entre le « 5 » et le « 7 » en effectuant un échange avec le « 7 ». Nouvelle version de la liste : a = [**1, 2, 5, 6, 7**]… la liste est triée !

4. Autres algorithmes de tri

a. Tri à bulles

Le tri à bulles consiste à parcourir la liste autant de fois que nécessaire en comparant à chaque fois un élément avec le suivant pour savoir s'il faut les échanger ou pas… Quand il n'y a plus aucun échange à faire, la liste est triée ! Cet algorithme est peu utilisé mais facile à programmer.

b. Tri rapide et tri fusion

Le tri rapide (*quicksort* en anglais) et le tri fusion (*merge sort* en anglais) utilisent la méthode « Diviser pour régner » (qui sera vue en Terminale) qui consiste à diviser le problème en deux problèmes similaires indépendants plus petits. Ces deux algorithmes sont plus rapides que les tris précédents.

5. Instructions de tri en Python

a. La fonction et la méthode

- La méthode liste.*sort*() : Trie la liste sur laquelle on l'utilise, elle est alors modifiée.
- La fonction *sorted*(liste) : Renvoie une nouvelle liste triée sans modifier la liste utilisée.

b. Les arguments

Par défaut la liste est triée du plus petit au plus grand (ou alphabétique, de ″a″ à ″z″), mais on peut préciser *reverse = True* si on veut un tri descendant (du plus grand au plus petit, ou de ″z″ à ″a″).
Exemple : notes.*sort*(*reverse = True*) avec une liste de notes.
Pour des listes de listes ou des listes d'objets possédant des attributs, on peut préciser le critère principal pour le tri avec une clef dans une fonction lambda.
Exemple : liste.*sort*(key = *lambda* info : info[″NOM″]) avec une liste de dictionnaires du TP_13.