TRIER

À la fin de ce chapitre, je sais :

- ☞ expliquer le fonctionnement d'algorithmes de tri simple,
- ☞ caractériser un tri (comparatif, en place, stable, en ligne)

Trier est une activité que nous pratiquons dès notre plus jeune âge : les couleurs, les formes, les tailles sont autant d'entrées possibles pour cette activité. Elle est également omniprésente dans l'activité humaine, dès l'instant où il faut rationaliser une activité. Dans le cadre du traitement de l'information, on y a si souvent recours qu'on n'y prête quasiment plus attention. C'est pourquoi les algorithmes de tri ont été intensément étudiés et raffinés à l'extrême : il s'agit d'être capable de trier n'importe quel jeu de données pourvu qu'un ordre puisse être défini et de sélectionner le tri le plus efficace selon le tri à effectuer.

Ce chapitre esquisse donc quelques algorithmes de tri simples à coder.

Dans tout le chapitre, on considère :

— un tableau noté t, indiçable avec des crochets, c'est-à-dire que l'on peut accéder à un élément de la case i en écrivant $t[i]$,

— les indices d'un tableau de taille n vont de 0 à $n-1$,

— des éléments à trier.

La plupart du temps, si le tri est générique ou par comparaison, le type des éléments du tableau importe peu, pourvu qu'on dispose d'un ordre sur ces éléments. On donnera au cours de l'année une définition précise d'une relation d'ordre, mais, pour l'instant, on considère juste qu'**il est possible de comparer deux éléments entre eux**. C'est-à-dire qu'on peut réaliser un test de type `t[i] < t[j]` et récupérer une réponse booléenne à ce test. C'est d'ailleurs ainsi que la plupart des contexte informatique définissent un ordre dans un type de donnée.

(P) En Python, on implémentera ces tableaux par des listes.

A Comment caractériser un algorithme de tri ?

Pour distinguer les différents algorithmes de tri, on dispose de nombreux qualificatifs. Les définitions suivantes en expliquent quelques uns parmi les plus courants.

■ **Définition 1 — Tri par comparaisons.** Un tri par comparaison procède en comparant les éléments deux à deux[a].

a. ce qui sous-entend que certains algorithmes, comme l'algorithme de tri par comptage, procèdent différemment.

■ **Définition 2 — Tri en place.** Un tri est dit en place s'il peut être directement effectué dans le tableau initial et ne nécessite pas l'allocation d'une nouvelle structure en mémoire de la taille du tableau initial[a].

a. Cette propriété est importante car la mémoire est un paramètre très couteux des systèmes électroniques, tant sur le plan énergétique que sur le plan financier.

■ **Définition 3 — Tri incrémental ou en ligne.** Il s'agit d'un tri capable de commencer le tri avant même d'avoir reçu l'intégralité des données[a].

a. Cette propriété est très intéressante dans le cadre du traitement en temps réel des systèmes dynamiques (qui évoluent dans le temps)

■ **Définition 4 — Tri stable.** Un tri est dit stable s'il préserve l'ordonnancement initial des éléments que l'ordre considère comme égaux mais que l'on peut distinguer[a].

a. par exemple, dans un jeu de cartes, deux valets (même type de carte) de couleurs différentes.

B Trier un tableau

a Tri par sélection

Le tri par sélection est un tri par comparaisons et échanges, en place, non stable et hors ligne.

Son principe est le suivant : on cherche le plus petit élément du tableau (de droite) et on l'insère à la fin du tableau trié de gauche. Au démarrage de l'algorithme, on cherche le plus petit élément du tableau et on l'insère à la première place. Puis on continue avec le deuxième plus petit élément du tableau que l'on ramène à la deuxième place et ainsi de suite.

Algorithme 1 Tri par sélection

```
1 : Fonction TRIER_SELECTION(t)
2 :     n ← taille(t)
3 :     pour i de 0 à n − 1 répéter
4 :         min_index ← i                          ▷ indice du prochain plus petit
5 :         pour  j de i + 1 à n − 1 répéter       ▷ pour tous les éléments non triés
6 :             si t[j] < t[min_index ] alors
7 :                 min_index ← j                  ▷ c'est l'indice du plus petit non trié !
8 :         échanger(t[i], t[min_index])           ▷ c'est le plus grand des triés !
```

b Tri par insertion

Le tri par insertion est un tri par comparaison, stable et en place. De plus, si les données ne sont pas toutes présentes au démarrage de l'algorithme, le tri peut quand même commencer, ce qui n'est pas le cas du tri par sélection. C'est donc également un tri en ligne (ou incrémental).

R **Un tableau ne comportant qu'un seul élément est déjà trié.**

Le principe du tri par insertion est le suivant : on considère deux parties gauche (triée) et droite (non triée) du tableau. Puis, on cherche à insérer le premier élément non trié (du tableau de droite) dans le tableau trié (de gauche) à la bonne place. Au démarrage de l'algorithme, on considère le tableau (trié) constitué par le premier élément et on cherche à insérer le deuxième élément à la bonne place dans ce tableau.

Algorithme 2 Tri par insertion

```
1 : Fonction TRIER_INSERTION(t)
2 :     n ← taille(t)
3 :     pour i de 1 à n-1 répéter
4 :         à_insérer ← t[i]
5 :         j ← i
6 :         tant que t[j-1] > à_insérer et j>0  répéter
7 :             t[j] ← t[j-1]                          ▷ faire monter les éléments
8 :             j ← j-1
9 :         t[j] ← à_insérer                           ▷ insertion de l'élément
```

c Tri par comptage

Le tri par comptage est un tri par dénombrement de valeurs entières. Il ne porte donc que sur des tableaux contenant des entiers. C'est un tri non comparatif, stable et hors ligne.

Le principe du tri par comptage est de compter le nombre d'occurrences de chaque valeur entière puis de construire un nouveau tableau à partir de ce comptage.

R Ce tri est certes non comparatif car il n'utilise pas explicitement de comparaisons. Mais l'ordre des entiers est utilisé lors du comptage.

Algorithme 3 Tri par comptage

```
1 : Fonction TRIER_COMPTAGE(t, v_max)          ▷ v_max est le plus grand entier à trier
2 :     n ← taille(t)
3 :     c ← un tableau de taille v_max + 1 initialisé avec des zéros
4 :     pour i de 0 à n − 1 répéter
5 :         c[t[i]] ← c[t[i]] + 1              ▷ compter les occurrences de chaque élément du tableau.
6 :     résultat ← un tableau de taille n
7 :     i ← 0
8 :     pour v de 0 à v_max répéter            ▷ On prend chaque valeur possible dans l'ordre
9 :         si c[v] > 0 alors                  ▷ Si l'élément v est présent dans le tableau
10 :            pour j de 0 à c[v] - 1 répéter
11 :                résultat[i] ← v            ▷ alors écrire autant de v que d'occurrences de v
12 :                i ← i + 1                  ▷ à la bonne place, la ième !
13 :    renvoyer résultat
```

C Comparatif des tris

Les tableaux 1 et 2 résument les caractéristiques et les complexités des principaux algorithmes de tri au programme.

D Trier avec les fonctions natives de Python

P Il existe deux solutions natives pour trier une liste en Python :

1. appeler la fonction `sorted` sur une liste ou un dictionnaire. Cette fonction renvoie une nouvel objet trié sans modifier l'original. Par exemple, `T = sorted(L)`. Dans le cas d'un dictionnaire, ce sont les clefs qui sont triées.
2. appeler la méthode `sort` de la classe List. Cette méthode effectue un tri en place sur une liste et la modifie. Par exemple, `L.sort()`. Cette méthode n'existe pas pour les dictionnaires.

Tris	**Comparaison**	**Stable**	**En place**	**En ligne**
par sélection	oui	non	oui	non
par insertion	oui	oui	oui	oui
par comptage	non	oui	non	non
fusion	oui	oui	oui (mais [1])	non
rapide	oui	non	oui	non

TABLE 1 – Comparatif des caractéristiques des algorithmes de tri. Ils existent de nombreuses variantes de ces algorithmes. Les informations de ce tableau concernent les versions implémentées en cours ou en TP.

Tris	Complexité temporelle **pire des cas**	Complexité temporelle **moyenne**	Complexité temporelle **meilleur des cas**	Complexité spatiale
par sélection	$O(n^2)$	$O(n^2)$	$O(n^2)$	$O(1)$
par insertion	$O(n^2)$	$O(n^2)$	$O(n)$	$O(1)$
par comptage	$O(n+v\_max)$	$O(n+v\_max)$	$O(n+v\_max)$	$O(n+v\_max)$
fusion	$O(n\log n)$	$O(n\log n)$	$O(n\log n)$	$O(n)$
rapide	$O(n^2)$	$O(n\log n)$	$O(n\log n)$	$O(\log n)$

TABLE 2 – Complexité des différents algorithmes de tri.

Les tris opérés par ces fonctions sont garantis stables. Cela signifie que lorsque plusieurs enregistrements on la même clef, leur ordre original est préservé.

La documentation Python est à consulter ici pour les usages plus spécifiques de ces fonctions. On peut notamment spécifier :

— un tri ascendant ou descendant à l'aide du paramètre optionnel booléen `reverse`,

— à la fonction l'élément à utiliser pour le tri (en cas d'élément multiple) et même le comparateur à l'aide du paramètre optionnel `key`.

P Par défaut, Python trie les n-uplets dans l'ordre lexicograhique.