Une introduction aux techniques de tri

Tri Bulles et Tri par sélection

Dans la plupart des applications, il est indispensable que des listes ou des fichiers soient triés selon des critères numériques ou alphanumériques. Bien entendu, lors de la saisie des informations, celles-ci ne sont pas forcément introduites dans l'ordre du critère retenu. Par exemple, si l'on crée un fichier clients, les noms des clients seront saisis dans l'ordre d'inscription et non par ordre alphabétique. Il est donc presque toujours nécessaire d'utiliser un programme de tri qui met la liste ou le fichier en ordre. Il existe des dizaines de techniques de tris qui se différencient par leur complexité.

Nous n'étudierons ici que deux techniques qui ont l'avantage d'être simples à mettre en œuvre, même si ce ne sont pas, et de loin, les plus rapides : Le tri Bulles et le tri par sélection.

Les algorithmes de tri mettent en œuvre deux types de d'instructions : La comparaison et la permutation.

1. Comparaison : utilise une relation d'ordre définie sur les éléments de la collection

Lorsque le critère du tri porte sur des valeurs numériques, le programme est fidèle au sens commun en ce qu'il respecte la **relation d'ordre sur les nombres** (2 <= 3 est VRAI, 2 > 3 est FAUX).

Attention aux valeurs alphanumériques (chaînes de caractères) qui sont comparées caractère par caractère en comparant le **code ASCII (ou unicode)** de chacun d'eux. Rappel :

'0' = 48	'1' = 49	'2' = 50	etc.
'A' = 65	'B' = 66	'C' = 67	etc.
'a' = 97	'b' = 98	'c' = 99	etc.

Ainsi :

"A" est inférieur à "a"
"A" est inférieur à "B"
"AA" est inférieur à "AB"
"123" est inférieur à "A"
"$A" est inférieur à "12347"
"123" est inférieur à "14"
"2+2" est inférieur à "4" (le code ASCII de 2 est inférieur à celui de 4)
"DE DUPONT" est inférieur à "DE-DUPONT"
"" est inférieur à " " (une chaîne vide est inférieure à une chaîne contenant un espace)

=> En fait, la localisation des systèmes entraîne souvent des variations subtiles dans les relations d'ordres alphanumérique (traitement des accents par exemple).

2. Permutation

La permutation est le fait d'échanger le contenu de deux variables. Cette opération nécessite 3 transferts en mémoire.

On utilise une troisième variable "temp" de même type que les deux autres, de la même façon que l'on doit utiliser un troisième verre pour échanger le contenu d'un verre de vin avec celui d'un verre de limonade. Exemple.

```
Procédure echanger(entrée/sortie x, y : Entier)
  Variable locale
sauv_x : Entier
Début
        sauv_x ← x
x ← y
y ← sauv_x
Fin
```

Traduction en langage de programmation

Java	C++

Impossible !
 	
Une méthode ne peut changer la valeur de ses paramètres de type primitif.
 	
Le passage des paramètres se fait uniquement par valeur.
	void echanger(int & x, int & y)
	{ int sauv_x = x;
	x = y;
	y = sauv_x;
	}
	
	(avec technique de passe par référence)

3. Tri bulles

L'idée : Parcourir la liste en commençant par la fin, comparer l'élément de rang *i* avec son voisin immédiat de rang *i* - 1 et effectuer une permutation (échange de contenu) si l'élément de droite (de rang *i*) et inférieur à son voisin de gauche (de rang *i* -1).

Le nom de ce tri vient de ce que les éléments les plus petits (les plus "légers") remontent vers le début de la liste comme des bulles vers le haut d'un tube à essai.

Prenons comme exemple un tableau de 5 entiers à trier :

65	31	26	16	21	*Eléments*
1	*2*	*3*	*4*	*5*	*Indices*

Suivons le déroulement de l'idée. On veut amener l'élément le plus "léger" en "haut" du tableau (indice 1). Pour cela on commence la comparaison par le "bas", puis on "monte".
Le couple d'éléments comparés est souligné en pointillé, les permutations sont soulignées par un trait plein.

Θ <

Suivez la bulle...

65 31 26 16 21

Θ

65 31 26 16 21
16 26

Θ

65 31 16 26 21
16 31

Θ

65 16 31 26 21
16 65

On vient de déterminer t [1], poursuivons (à partir de la fin de nouveau, mais dans l'intervalle d'incides [2..5], car le premier n'est plus à traiter)

2ième tour :

16 65 31 26 21
21 26

16 65 31 21 26
21 31

16 65 21 31 26
21 65

3ième tour :

16 **21** 65 31 26
26 31

16 **21** 65 26 31
26 65

4ième tour :

16 **21** **26** 65 31
31 **65**

16 **21** **26** **31** **65** **Trié**

Nous constatons que nous avons réalisé n-1 tours.
Au premier tour nous avons effectué 4 comparaisons, au deuxième tour 3 comparaisons, au troisième 2 et au dernier 1 comparaison.

procédure TriBulles (e/s t : TABLEAU [1:n] d'entiers)
// Descr. : en sortie, c-a-d en fin de traitement, *t* sera trié en ordre croissant

variables locales
    i, j, temp : Entier
Début
    Pour i ← 1 **à** n - 1
    Pour j ← n **à** i + 1 (par pas de -1)
        Si t [j - 1] > t [j] **Alors**
            temp ← t [j - 1]
            t [j - 1] ← t [j]
            t [j] ← temp
        Fsi
    FinPour
    FinPour
Fin

A chaque tour de *i* (lorsque la boucle *j* est terminée), ce tri place en t [i] l'élément le plus petit rencontré dans la plage t [i] .. t [n]. Ce résultat est obtenu par une série de permutations d'éléments adjacents (t[j-1] et t[j]) réalisée dans la boucle interne *j*.

Traduction dans votre lg de programmation préféré :

Il arrive parfois que le tableau soit trié avant que la boucle la plus externe (*i*) ne soit terminée. Exemple.

Prenons comme autre exemple un tableau de 5 entiers à trier :

1	*2*	*3*	*4*	*5*	*Indices*
26	31	65	16	21	*Eléments*

i = 1, détermination de t [1]

j = 5	26	31	65	**16**	21
j = 4	26	31	**16**	65	21
j = 3	26	**16**	31	65	21
j = 2	**16**	26	31	65	21

i = 2, détermination de t [2]

j = 5	16	26	31	**21**	65	
j = 4	16	26	**21**	31	65	
j = 3	16	**21**	26	31	65	**Trié !**

Il est ici inutile de poursuivre, car la boucle interne n'engendrera plus aucune permutation.

Nous pouvons donc améliorer la procédure TriBulles en évitant les parcours sans échanges: Lorsque aucune permutation n'est réalisée dans la boucle interne *J* la boucle externe *i* peut se terminer.

Exercice : proposer une variante de la procédure TriBulles qui tienne compte de cette idée.

```
procédure TriBulles ( t  : TABLEAU (1:n) d'entiers  entrée-sortie )
// Descr. : en sortie, t sera trié en ordre croissant
variables locales
        i, j : Entier
        .
Début
        .
        .
        .
        .
        .
        .
        .
        .
        .
        .
        .
Fin
```

4. TRI PAR SELECTION

Nous avons vu qu'à chaque tour de *i* (lorsque la boucle *j* est terminée), le tri Bulles place en t [i] l'élément le plus petit rencontré dans la plage t [i] .. t [n].

Ce placement est obtenu par une série de comparaisons ET de permutations.

Dans notre exemple, afin de placer l'élément minimum (16) à sa place (t[1]) nous avons effectué quatre permutations !

Il aurait été plus économique de n'effectuer qu'une permutation dés lors que nous savons que 16 est le plus petit élément de la liste en cours. Cette idée a permis de mettre au point un algorithme de tri qui porte le nom de *tri par sélection*.

Nous avons déjà étudié l'algorithme de recherche linéaire d'un minimum dans une liste. Ici, c'est la boucle interne qui détermine la valeur de l'indice d'un élément minimum (dans une portion droite de la liste). A la sortie de la boucle interne (fin de la recherche du minimum) on place le minimum en début de la sous-liste.

```
procédure TriSelection ( t : TABLEAU [1:n] d'entiers  (entrée-sortie) )
// Descr. : en sortie, t sera trié en ordre croissant

variables locales
        i, j, min,  : Entier
Début
        Pour i ← 1 à n - 1
        min ← i
        // recherche de l'indice du minimum dans l'intervalle t[i]..t[n]
        Pour j  ← i + 1 à n
                Si t [ j ] < t [ min ] Alors
                        min ← j
                Fsi
        FinPour
        // Si on a trouvé plus petit alors on échange le contenu
        Si i <> min Alors
                 echanger  t [ i ], t [ min ]
        FSi
        FinPour
Fin
```

5. RELATION D'ORDRE

Nous avons vu qu'il existe des relations d'ordre « **naturelles** » : sur les nombres, pour les chaînes de caractère… Ces relations d'ordre sont des relations d'ordre *par défaut.*

Cela sous-entend que le développeur peut aller outre la relation naturelle et décider d'une relation d'ordre spécifique, pour des besoins métier.

Ceci a été vu dans le cas d'un jeu de carte, si l'on souhaite, par exemple, que "10" soit supérieur à "9" par exemple, ou l'as ("1" supérieur au"ROI") (méthode *compare*)

Attention, la relation d'ordre naturelle doit être cohérente (on dit *consistante*) avec l'égalité (redéfinition de *equals,* compatible avec la comparaison faible == *en PHP, ...*).

PHP offre plusieurs fonctions de tri Voir :
https://www.php.net/manual/fr/array.sorting.php

Ces fonction travaillent soient sur les valeurs du tableau, soit sur les clés (une *array* pouvant référencer soit un tableau en mémoire avec des indices numériques, soir un tableau associatif). Par exemple *sort* utilise la relation d'ordre dite naturelle (pré-définie) et *usort* utilise une relation d'ordre définie par l'utilisateur (le *u* de *usort*)

6. COMPARAISON DES TRIS EN TERME DE PERFORMANCE.

Le calcul de la complexité s'effectue en calculant le nombre de comparaisons et d'échanges à effectuer dans le pire cas.

En nombre de comparaisons, Il y a en moyenne autant de comparaisons dans les deux tris, soit : $\frac{n(n-1)}{2}$

En nombre d'échanges (un échange correspond à trois transferts)

Tri Bulles	Tri par sélection
$\frac{n(n-1)}{4}$	$n-1$
Complexité d'Ordre(n^2)	Complexité d'Ordre(n)

Exemple d'ordre de grandeur pour une liste de 65 535 éléments

Nombre de comparaison : 2 147 385 345

Nombre d'échanges :

Tri Bulles	1 073 692 676
Tri par sélection	65 534

Quelques qualités des algorithmes de tris :

- tri stable : préserve l'ordre des éléments égaux
- tri en place : n'utilise qu'un nombre limité d'espace supplémentaire
- tri interne / externe : Le tri externe utilise de la mémoire disque en plus de la mémoire vive.

Plus loin : tri par insertion · par tas · par base · rapide · fusion ·