Algorithmique : Introduction aux tris

Chapitre 16

Table des matières

Nous avons vu, au chapitre 14, que *la recherche d'un élément dans un tableau est beaucoup plus efficace si ce tableau est ordonné*[1]. La question que se propose d'aborder ce chapitre est : « comment classer les éléments d'un tableau ? ».

Cette question est suffisamment importante pour que de nombreux chercheurs se soient penchés sur le problème et aient proposé plusieurs dizaines d'algorithmes différents. Certains sont *spécifiques au données numériques*, certains *copient les données dans une nouvelle structure* — ce qui réclame de l'espace en mémoire —, certains *effectuent un tri sur place* — c'est à dire dans le même espace mémoire, ...

Ce chapitre aborde l'étude de deux tris très différents : le *tri par sélection* et le *tri fusion*. Ces algorithmes appartiennent aux deux catégories de tris que l'on peut rencontrer : *les tris élémentaires* et les *tris avancés*.

- *Les tris élémentaires* s'appuient sur des raisonnements simples, que l'on retrouve parfois dans la vie courante, comme le tri des cartes qu'un joueur possède dans sa main, ...

- *Les tris avancés* se basent sur des raisonnements plus sophistiqués qui nécessitent de nouvelles structures de données, la récursivité, ...

1 Algorithme de tri

Un **algorithme de tri** est, en informatique ou en mathématiques, *un algorithme qui permet d'organiser une collection d'objets selon un ordre déterminé*.

Les objets à trier font donc partie d'un *ensemble muni d'une relation d'ordre*. Les ordres les plus utilisés sont l'*ordre numérique* et l'*ordre lexicographique* (dictionnaire).

2 Le tri par sélection

C'est le tri le plus simple, une applet de démonstration se trouve à l'adresse : `http://lwh.free.fr/index.html`, une animation bien plus amusante sur Youtube : `https://www.youtube.com/watch?t=12&v=Ns4TPTC8whw` et toutes les informations sur la page Wikipedia : `http://fr.wikipedia.org/wiki/Tri_par_sélection`.

Idée. On recherche la plus petite valeur et on l'échange avec celle située à la première position. On recherche alors la deuxième plus petite valeur et on l'échange avec celle située à la deuxième position, et ainsi de suite.

Note. L'inversion des valeurs contenues dans deux variables est souvent employée en informatique, elle nécessite une troisième variable pour un stockage temporaire.

1. À vrai dire, ce n'est pas en cours d'informatique que vous avez découvert ceci : dans toutes les bibliothèques les livres sont classés de façon à rendre leur recherche plus rapide !

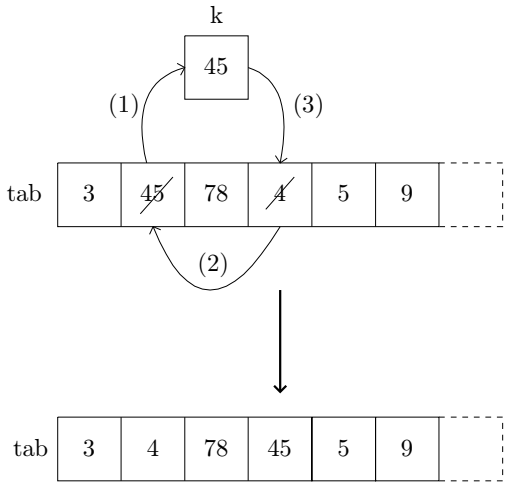

Figure 1. Inversion de deux cases d'un tableau.

Le *tri par sélection* est basé sur l'utilisation de deux boucles `POUR` imbriquées :

- La première boucle parcourt la liste des valeurs, de la première à la dernière ;
- La seconde boucle recherche la plus petite valeur, de la position courante (compteur de la première boucle) à la fin du tableau, puis l'inverse avec la position courante.

La partie gauche de la liste est donc triée au fur et à mesure de l'avancement de la première boucle.

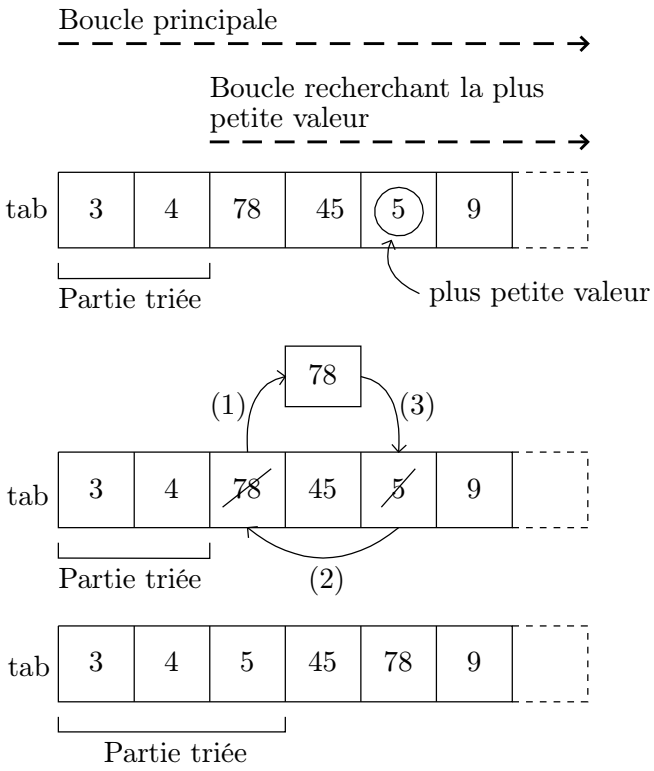

Figure 2. Illustration du tri par sélection.

Algorithme du tri par sélection

```
Procédure tri_sélection(tab, nb)
Déclarations
        Paramètre tab           tableau[20] d'Entiers
        Paramètre nb            Entier
        Variables i, j, k, min  Entiers
Début
        Pour i variant de 0 à nb-2 Faire
                min ← i
                // localisation du minimum
                Pour j variant de i+1 à nb-1 Faire
                        Si (tab[j] < tab[min]) Alors
                                min ← j
                        FinSi
                FinPour
                // inversion
                k ← tab[i]          // étape 1
                tab[i] ← tab[min]   // étape 2
                tab[min] ← k        // étape 3
        FinPour
Fin
```

nb est, dans cet algorithme, le nombre d'élements dans le tableau.

Note. L'algorithme du tri par sélection est un *algorithme de tri en place*.

3 Le tri par fusion

Le tri par fusion est basé sur la technique algorithmique « Diviser pour régner ». *L'opération principale de l'algorithme est la* ***fusion****, qui consiste à réunir deux listes triées en une seule*. L'efficacité de l'algorithme vient du fait que deux listes triées peuvent être fusionnées en temps linéaire.

Le tri fusion se décrit naturellement sur des listes et c'est sur de telles structures qu'il est à la fois le plus simple et le plus rapide.

« Diviser pour régner » (divide and conquer) est une technique algorithmique consistant à *découper un problème en sous-problèmes semblables mais de taille moindre* afin de diminuer l'effort à faire pour obtenir la solution à un problème. Chacun des sous-problèmes est résolu de façon récursive.

Le paradigme « Diviser pour régner » implique trois étapes à chaque niveau de la récursivité :

Diviser. le problème en un certain nombre de sous-problèmes qui sont des instances plus petites du même problème.

Régner. sur les sous-problèmes en les résolvant de manière récursive. Si la taille d'un sous-problème est suffisamment réduite, on peut le résoudre directement.

Combiner. les solutions des sous-problèmes pour produire la solution du problème originel.

Un exemple simple est la *recherche dichotomique*, qui consiste à diviser un ensemble de données ordonnées en deux et d'abandonner un des sous-ensembles au profit de l'autre.

Une applet de démonstration se trouve à l'adresse http://lwh.free.fr/index.html, une animation bien plus amusante sur Youtube : https://www.youtube.com/watch?v=XaqR3G_NVoo et la page Wikipédia est très complète http://fr.wikipedia.org/wiki/Tri_fusion.

Idée. Le cœur de l'algorithme repose sur l'idée qu'il est *simple de construire une liste résultant de la fusion de deux listes triées* : le plus petit élément de la liste à construire est soit le plus petit élément de la première liste, soit le plus petit élément de la deuxième liste. Ainsi, on peut construire la liste élément par élément en retirant tantôt le premier élément de la première liste, tantôt le premier élément de la deuxième liste (en fait, le plus petit des deux, à supposer qu'aucune des deux listes ne soit vide, sinon la réponse est immédiate).

3.1 Algorithme de fusion

Pour introduire l'algorithme de fusion, le plus simple est de partir d'un exemple. On cherche à réaliser la fusion de deux tableaux de même dimension tab_1 et tab_2 :

1. $\text{tab}_1 =$ | 5 | 9 | 11 | 17 | $\text{tab}_2 =$ | 3 | 4 | 11 | 13 | $\text{tab}_3 =$ | | | | | | | | |

   On note x, l'indice qui permet de parcourir le tableau tab_1. Au départ, $x = 0$.

   On note y, l'indice qui permet de parcourir le tableau tab_2. Au départ, y=0.

   On note i, l'indice qui permet de parcourir le tableau tab_3. Au départ, tab_3 est vide et $i = 0$.

2. $\text{tab}_1 =$ | 5 | 9 | 11 | 17 | $\text{tab}_2 =$ | 3 | 4 | 11 | 13 | $\text{tab}_3 =$ | 3 | | | | | | | |

   Au premier tour de boucle, $\text{tab}_1[0] \geqslant \text{tab}_2[0] \Rightarrow \text{tab}_3[0] = 3$.

   On incrémente de un les indices *qui sont entrés en jeu* : $i = 1$; $x = 0$; $y = 1$.

3. $\text{tab}_1 =$ | 5 | 9 | 11 | 17 | $\text{tab}_2 =$ | 3 | 4 | 11 | 13 | $\text{tab}_3 =$ | 3 | 4 | | | | | | |

   Au deuxième tour de boucle, $\text{tab}_1[0] \geqslant \text{tab}_2[1] \Rightarrow \text{tab}_3[1] = 4$.

   On incrémente de un les indices *qui sont entrés en jeu* : $i = 2$; $x = 0$; $y = 2$.

4. $\text{tab}_1 =$ | 5 | 9 | 11 | 17 | $\text{tab}_2 =$ | 3 | 4 | 11 | 13 | $\text{tab}_3 =$ | 3 | 4 | 5 | | | | | |

   Au troisième tour de boucle, $\text{tab}_1[0] \leqslant \text{tab}_2[2] \Rightarrow \text{tab}_3[2] = 5$.

   On incrémente de un les indices *qui sont entrés en jeu* : $i = 3$; $x = 1$; $y = 2$.

5. $\text{tab}_1 =$ | 5 | 9 | 11 | 17 | $\text{tab}_2 =$ | 3 | 4 | 11 | 13 | $\text{tab}_3 =$ | 3 | 4 | 5 | 9 | | | | |

   Au quatrième tour de boucle, $\text{tab}_1[1] \leqslant \text{tab}_2[2] \Rightarrow \text{tab}_3[3] = 9$.

   On incrémente de un les indices *qui sont entrés en jeu* : $i = 4$; $x = 2$; $y = 2$.

6. $\text{tab}_1 =$ | 5 | 9 | 11 | 17 | $\text{tab}_2 =$ | 3 | 4 | 11 | 13 | $\text{tab}_3 =$ | 3 | 4 | 5 | 9 | 11 | | | |

   Au cinquième tour de boucle, $\text{tab}_1[2] \leqslant \text{tab}_2[2] \Rightarrow \text{tab}_3[4] = 11$.

   On incrémente de un les indices *qui sont entrés en jeu* : $i = 5$; $x = 3$; $y = 2$.

7. $\text{tab}_1 =$ | 5 | 9 | 11 | 17 | $\text{tab}_2 =$ | 3 | 4 | 11 | 13 | $\text{tab}_3 =$ | 3 | 4 | 5 | 9 | 11 | 11 | | |

   Au sixième tour de boucle, $\text{tab}_1[3] \geqslant \text{tab}_2[2] \Rightarrow \text{tab}_3[5] = 11$.

On incrémente de un les indices *qui sont entrés en jeu* : $i=6$; $x=3$; $y=3$.

8. $tab_1 = \boxed{5}\boxed{9}\boxed{11}\boxed{17}$ $tab_2 = \boxed{3}\boxed{4}\boxed{11}\boxed{13}$ $tab_3 = \boxed{3}\boxed{4}\boxed{5}\boxed{9}\boxed{11}\boxed{11}\boxed{13}\boxed{\ }$

   Au septième tour de boucle, $tab_1[3] \geqslant tab_2[3] \Rightarrow tab_3[6] = 11$.

   On incrémente de un les indices *qui sont entrés en jeu* : $i=7$; $x=3$; $y=4$.

9. On copie maintenant le dernier élément de tab_1 dans tab_3.

   $tab_1 = \boxed{5}\boxed{9}\boxed{11}\boxed{17}$ $tab_2 = \boxed{3}\boxed{4}\boxed{11}\boxed{13}$ $tab_3 = \boxed{3}\boxed{4}\boxed{5}\boxed{9}\boxed{11}\boxed{11}\boxed{13}\boxed{17}$

Algorithme de fusion de deux tableaux différents

```
Fonction fusion(tab1, tab2, nb)
Déclarations
        Paramètres tab1, tab2   tableaux[nb] d'Entiers
        Paramètre nb            Entier
        Variables i,x,y         Entiers
Début
        x ← 0
        y ← 0
        Créer tab3 tableau[2 * nb]
        Pour i variant de 0 à 2 * nb - 1 Faire
                Si ((x <= nb-1 ET y <= nb-1 ET tab1[x] <= tab2[y]) OU (y = nb) Alors
                        tab3[i] ← tab1[x]
                        x ← x + 1
                Sinon
                        tab3[i] ← tab2[y]
                        y ← y + 1
                FinSi
        FinPour
        Retourner tab3
Fin
```

Remarque. Pour simplifier l'algorithme, les deux tableaux ont été supposés de même taille `nb`.

Algorithme de fusion de deux sous-segments d'un tableau

```
Fonction fusion(tab, nb)
Déclarations
        Paramètres tab          tableau[nb] d'Entiers
        Paramètre nb            Entier
        Variables i,x,y         Entiers
Début
        x ← 0
        y ← nb
        Créer tab3 tableau[2 * nb]
        Pour i variant de 0 à 2 * nb - 1 Faire
             Si ((x <= nb-1 ET y <= 2*nb - 1 ET tab[x] <= tab[y]) OU (y = 2 * nb) Alors
                  tab3[i] ← tab[x]
                  x ← x + 1
             Sinon
```

```
                    tab3[i] ← tab[y]
                    y ← y + 1
                FinSi
        FinPour
        Retourner tab3
Fin
```

3.2 Algorithme de tri par fusion

Remarque. Bien que l'algorithme soit naturellement récursif, nous allons, pour des raisons de simplicité, utiliser un raisonnement itératif. L'objectif ici est de comprendre la méthode.

Note. Toujours pour les mêmes raisons de simplicité, nous nous limitons aux tableaux constitués d'un nombre d'éléments égal à une puissance de 2.

Comment trier le tableau $A =$ | 42 | 63 | 32 | 14 | 58 | 69 | 51 | 17 | 19 | 84 | 35 | 22 | 93 | 11 | 67 | 36 | ?

1. Chaque case du tableau est un mini-segment de longueur 1, déjà trié donc.

   On regroupe par 2 et on fusionne, l'une après l'autre, les 8 paires de segments :

   $$A = \boxed{42\ 63} \boxed{14\ 32} \boxed{58\ 69} \boxed{17\ 51} \boxed{19\ 84} \boxed{22\ 35} \boxed{11\ 93} \boxed{36\ 67}$$

2. On regroupe 2 par 2 les 8 segments et l'on fusionne l'une après l'autre les 4 paires de segments :

   $$A = \boxed{14\ 32\ 42\ 63} \boxed{17\ 51\ 58\ 69} \boxed{19\ 22\ 35\ 84} \boxed{11\ 36\ 67\ 93}$$

3. On regroupe 2 par 2 les 4 segments et l'on fusionne l'une après l'autre les 2 paires de segments :

   $$A = \boxed{14\ 17\ 32\ 42\ 51\ 58\ 63\ 69} \boxed{11\ 19\ 22\ 35\ 36\ 67\ 84\ 93}$$

4. On fusionne cette paire de segments :

   $$A = \boxed{11\ 14\ 17\ 19\ 22\ 32\ 35\ 36\ 42\ 51\ 58\ 63\ 67\ 69\ 84\ 93}$$

4 L'efficacité des algorithmes

Au chapitre précédent, la *notion de complexité* a été abordée. Nous allons introduire, dans cette partie, quelques règles simples qui permettent de se faire une idée de l'efficacité d'un algorithme.

- Une *affectation* ou l'*évaluation d'une expression* ont un temps d'exécution petit. Cette durée constitue souvent l'unité de base dans laquelle on mesure le temps d'exécution d'un algorithme.

- Le temps pris pour exécuter une séquence `p q` est la somme des temps pris pour exécuter les instructions `p` puis `q`.

- Le temps pris pour exécuter *un test* `Si (b) Alors p Sinon q FinSi` est inférieur ou égal au maximum des temps pris pour exécuter les instructions `p` et `q`, plus une unité qui correspond au temps d'évaluation de l'expression `b`.

- Le temps pris pour exécuter une boucle `Pour i variant de 1 à m par pas de 1 Faire p FinPour` est `m` fois le temps pris pour exécuter l'instruction `p` si ce temps ne dépend pas de la valeur de `i`.

  En particulier, quand deux boucles sont imbriquées, le corps de la boucle interne est répété à cause de cette boucle, mais aussi parce qu'elle-même est répétée dans son intégralité. Ainsi, si les deux boucles sont répétées respectivement `m` et `m'` fois, alors le corps de la boucle interne est exécuté `m * m'` fois en tout.

  Quand le temps d'exécution du corps de la boucle dépend de la valeur de l'indice `i`, le temps total d'exécution de la boucle est la somme des temps d'exécution du corps de la boucle pour chaque valeur de `i`.

- Le cas des boucles `Tantque` est plus complexe puisque le nombre d'itérations n'est en général pas connu à priori.

5 L'efficacité des algorithmes de tri par sélection et par fusion

5.1 Tri par sélection

La complexité de cet algorithme est en $O(N^2/2)$ puisque, pour chaque valeur i du compteur de la première boucle (qui varie de 1 à N), on effectue $N-i$ comparaisons. Le nombre total de comparaisons est donc :

$$(N-1)+(N-2)+\ldots+2+1=\frac{N(N-1)}{2}$$

Or

$$\frac{N(N-1)}{2}=\frac{N^2}{2}-\frac{N}{2}$$

Le Terme $N/2$ devient négligeable devant $N^2/2$ lorsque N devient très grand.

De même, on peut négliger le temps pris pour les permutations des éléments d'indices i et j, car cette opération est effectuée un nombre de fois proportionnel à N.

5.2 Tri par fusion

Dans cette partie, nous allons poser sans démonstration que le tri par fusion est en $O(N\log N)$[2]. Ainsi, le tri d'un tableau d'un million d'éléments demande un temps :

- de $N^2=10^{12}$ pour l'algorithme de tri par sélection ;

- de $N\log N=10^6\times\log 10^6=10^6\times 20$ pour l'algorithme de tri par fusion.

Si un ordinateur exécute le corps de la boucle interne en une milliseconde, l'implémentation du second algorithme mettra 20 secondes à trier la liste alors que celle du premier mettra 10 jours environ !

2. On rappelle que le logarithme utilisé ici est le logarithme en base 2, soit le nombre x tel que, pour un nombre y donné, $y=2^x$.

Remarque. *Le tri par fusion ne se fait pas en place.* Il est donc beaucoup plus gourmand en ressources mémoire que le tri par sélection.

6 Exercices

Exercice 1. Tris par sélection à la main

Trier à la main les quatre tableaux suivants, par sélection :

1. $A =$ | 57 | 8 | 14 | 26 | 41 | 98 | 3 |
2. $B =$ | 97 | 83 | 55 | 81 | 20 | 73 | 0 |
3. $C =$ | 7 | 6 | 5 | 4 | 3 | 2 | 1 |
4. $D =$ | 2 | 3 | 4 | 5 | 6 | 7 | 1 |

Exercice 2. Implémentation de l'algorithme de tri par sélection

Écrire un programme en Python qui implémente l'algorithme de tri par sélection. Tester ce programme.

Exercice 3. Fusion de deux tableaux à la main

Réaliser à la main la fusion des deux tableaux suivants : $C =$ | 7 | 6 | 5 | 4 | 3 | 2 | 1 | et $D =$ | 2 | 3 | 4 | 5 | 6 | 7 | 1 |.

Exercice 4. Fusion d'un tableau à la main

Réaliser à la main la fusion du tableau $B =$ | 97 | 83 | 55 | 81 | 20 | 73 | 0 |.

Exercice 5. Implémentation de l'algorithme de fusion de deux listes triées

Écrire un programme en Python qui implémente l'algorithme de fusion de deux listes triées de même taille. Tester ce programme.

Exercice 6. Fusion de trois listes triées de même taille

Modifier le programme précédent de façon à pouvoir réaliser la fusion de trois listes triées de même taille. Tester ce programme.

Exercice 7. Tri par fusion à la main

Effectuer à la main un tri par fusion des tableaux :

1. $A =$ | 66 | 11 | 20 | 43 | 80 | 22 | 17 | 61 |
2. $B =$ | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 |
3. $C =$ | 28 | 10 | 31 | 38 | 12 | 66 | 44 | 62 |
4. $D =$ | 5 | 1 | 7 | 3 | 6 | 2 | 8 | 4 |

Exercice 8. Tri par fusion

Construire l'algorithme du tri par fusion et écrire un programme en Python réalisant l'implémentation de cet algorithme.