Chapitre 4

Trier un tableau

4.1 Introduction

Réaliser un tri (ou un classement) est une opération relativement courante dans la vie quotidienne : trier les cartes d'un jeu, trier par ordre alphabétique les livres d'une bibliothèque, classer des concurrents selon leurs performances, etc... Gérer un agenda téléphonique est aussi une forme de tri, puisqu'un ordre (alphabétique selon le nom des individus) est appliqué, même si ce tri est fait petit à petit, au fur et à mesure de l'insertion de nouveaux numéros.

Un tri porte généralement sur un nombre assez important de données. En effet, lorsque l'on a peu de numéros de téléphone à gérer, on se contente souvent de les inscrire dans l'ordre où ils sont connus sur une feuille libre, mais, dès que leur nombre devient trop important[1], on ressent le besoin de les classer et donc de les trier dans un agenda, afin d'accéder plus rapidement à une information.

Effectuer un tri est souvent assez long et fastidieux (du moins quand les données initiales ne sont pas ou peu triées). Cependant, l'intérêt d'une telle opération, une fois réalisée, est de pouvoir facilement accéder aux différentes données en s'appuyant sur le critère du tri. C'est le cas en particulier de l'algorithme efficace de recherche dichotomique (cf **??**).

Un tri est toujours réalisé selon un critère, particulier et arbitraire, portant sur ces données : une *relation d'ordre*, c'est-à-dire une relation binaire $\mathcal{R}$ sur les éléments d'un ensemble E ayant les trois propriétés

- *réflexivité*, *i.e.* $\forall x \in E$, $x\mathcal{R}x$;
- *antisymétrie*, *i.e.* $\forall x, y \in E$, $x\mathcal{R}y$ et $y\mathcal{R}x$ entraînent $x = y$;
- *transitivité*, *i.e.* $\forall x, y, z \in E$, $x\mathcal{R}y$ et $y\mathcal{R}z$ entraînent $x\mathcal{R}z$.

Cette relation est dite *totale* si $\forall x, y \in E$, on a $x\mathcal{R}y$ ou $y\mathcal{R}x$. Elle est dite *partielle* dans le cas contraire.

Ainsi, on peut arbitrairement choisir de classer les livres d'une bibliothèque selon un ordre croissant ou décroissant : alphabétique par auteurs ou par titres ; ou numérique selon les dates de parutions, etc.

Du fait de leur fréquente utilisation, les tris ont été très étudiés en informatique et constituent des exercices classiques lors de l'initiation à la programmation, car ils représentent un excellent support pour l'étude de problèmes plus généraux. De nombreux algorithmes de tri existent, plus ou moins efficaces et plus ou moins faciles à mettre en œuvre.

[1] Il est d'ailleurs difficile d'expliquer à partir de quand ce nombre est trop important, on retrouve ici le problème de savoir combien il faut de grains de sable pour en faire un tas...

4.2 Qu'est-ce que trier ?

On suppose dans ce chapitre que les éléments des tableaux considérés appartiennent à un ensemble E totalement ordonné par une relation notée $\leq$. Cet ensemble peut être par exemple l'ensemble des nombres entiers ou flottants muni de l'ordre numérique usuel, l'ensemble des caractères muni de l'ordre induit par le code ASCII, l'ensemble des chaînes de caractères muni de l'odre lexicographique.

`Trier un tableau` c'est obtenir, à partir d'un tableau t, un tableau contenant les mêmes éléments mais rangés par ordre croissant

Du point de vue du traiement des données, cette définition n'est pas suffisante : doit-on construire un nouveau tableau et laisser le tableau t inchangé ? ou bien doit-on transformer le tableau de sorte qu'il soit trié ?

Le problème du tri d'un tableau peut donc se décliner sous ces deux aspects :

1. à partir d'un tableau t, construire un nouveau tableau t' trié de même longueur que t, contenant les mêmes éléments que t ;
2. transformer un tableau t en un tableau trié, sans globalement changer son contenu, mais en déplaçant les éléments au sein du tableau.

De nombreux algorithmes de tri ont été conçus. Parmi eux, on distingue les tris *comparatifs* qui opèrent par comparaison d'éléments du tableau (tri par sélection, tri par insertion, ...), d'autres tris qui opèrent par d'autres moyens (comme le tri par dénombrement par exemple).

4.3 Le tri par sélection

4.3.1 Sélection du minimum dans une tranche de tableau

Algorithme 4.1 Algorithme de sélection du minimum dans une tranche de tableau

Entrée : t un tableau de longueur n, et a et b deux indices tels que $0 \leq a \leq b < n$.
Sortie : indice d'un élément minimal de la tranche $t(a..b)$

```
1: indice_min := a
2: pour i variant de a + 1 à b faire
3:     si t(i) < t(indice_min) alors
4:         indice_min := i
5:     fin si
6: fin pour
7: renvoyer indice_min
```

4.3.2 L'algorithme de tri par sélection

Le principe de l'algorithme de tri par sélection consiste à construire petit à petit une tranche triée grandissante du tableau en

- sélectionant à chaque étape le plus petit élément de la partie non triée
- et en l'échangeant avec l'élément du début de la tranche non triée.

Exemple 4.1 :
La table 3.1 montre l'évolution de la tranche triée (en vert) au cours des différentes étapes du tri par sélection pour le tableau de longueur 8 :

$t =$

	0	1	2	3	4	5	6	7
$t =$	T	I	M	O	L	E	O	N

i	ind_min	t
0	5	E I M O L T O N
1	1	E I M O L T O N
2	4	E I L O M T O N
3	4	E I L M O T O N
4	7	E I L M N T O O
5	6	E I L M N O T O
6	7	E I L M N O O T

TAB. 4.1 – États successifs du tableau au cours du tri par sélection

Algorithme 4.2 Algorithme de tri par sélection du minimum

Entrée : t un tableau de longueur n.
Sortie : t un tableau trié de longueur n contenant les mêmes éléments.

1: **pour** i variant de 0 à $n-2$ **faire**
2: $indice\_min :=$ sélectionner_min$(t, i, n-1)$
3: échanger $t(indice\_min)$ et $t(i)$
4: {À ce stade, la tranche $t(0..i)$ est triée
et contient les $i+1$ plus petits éléments de t}
5: **fin pour**
6: {Le tableau t est trié.}

4.3.3 Coût du tri par sélection

Notons $c_{\text{tri-select}}(n)$ le nombre de comparaisons d'éléments du tableau à effectuer pour trier un tableau de longueur n par le tri par sélection. Et notons $c_{\text{select-min}}(k)$ le nombre de comparaisons à effectuer pour sélectionner l'indice du minimum d'une tranche de tableau de longueur k selon l'algorithme 3.1.

Exemple 4.2 :
Commençons par examiner le nombre de comparaisons effectuées dans le tri du tableau de l'exemple 3.1.

En reprenant le tableau 3.1, voici le nombre de comparaisons d'éléments de t effectuées à chaque étape.

i	0	1	2	3	4	5	6
$c_{\text{select-min}}(8-i)$	7	6	5	4	3	2	1

Soit au total $7 + 6 + \ldots + 1 = 28$ comparaisons.

L'examen de l'algorithme 3.2 montre que l'on a

$$c_{\text{tri-select}}(n) = \sum_{k=2}^{n} c_{\text{select-min}}(k),$$

et celui de l'algorithme 3.1 montre que

$$c_{\text{select-min}}(k) = k - 1.$$

Par conséquent, on a

$$\begin{aligned} c_{\text{tri-select}}(n) &= \sum_{k=2}^{n} (k-1) \\ &= \frac{n(n-1)}{2}. \end{aligned}$$

Théorème 4.1 (*Coût du tri par sélection*)**.** *Pour tout tableau de longueur n, le nombre de comparaisons d'éléments du tableau pour le trier par le tri par sélection est donné par*

$$c_{\text{tri-select}}(n) = \frac{n(n-1)}{2} \sim \frac{n^2}{2}.$$

4.4 Le tri par insertion

4.4.1 Insertion d'un élément dans une tranche triée

Algorithme 4.3 Insertion d'un élément dans une tranche triée de tableau

Entrée : t un tableau de longueur n et $1 \leq i < n$ un indice tel que la tranche $t(0..i-1)$ soit triée

Sortie : insertion de l'élément $t(i)$ dans $t(0..i)$ de sorte que la tranche $t(0..i)$ soit triée

1: $aux := t(i)$
2: $k := i$
3: **tant que** $(k \geq 1)$ et $(aux < t(k-1))$ **faire**
4: $\quad t(k) := t(k-1)$
5: $\quad k := k - 1$
6: **fin tant que**
7: $t(k) := aux$
8: {la tranche $t(0..i)$ est triée}

4.4.2 L'algorithme du tri par insertion

Le principe de l'algorithme du tri par insertion consiste à contruire une tranche triée de longueur croissante en insérant successivement le premier élément non considéré dans la tranche triée en construction.

Exemple 4.3 :

i	t							
1	I	T	M	O	L	E	O	N
2	I	M	T	O	L	E	O	N
3	I	M	O	T	L	E	O	N
4	I	L	M	O	T	E	O	N
5	E	I	L	M	O	T	O	N
6	E	I	L	M	O	O	T	N
7	E	I	L	M	N	O	O	T

TAB. 4.2 – États successifs du tableau au cours du tri par insertion

Algorithme 4.4 Algorithme de tri par insertion

Entrée : t un tableau de longueur n.
Sortie : t un tableau trié de longueur n contenant les mêmes éléments.

1: **pour** i variant de 1 à $n-1$ **faire**
2: inserer $t(i)$ dans $t(0..i)$
3: {À ce stade, la tranche $t(0..i)$ est triée}
4: **fin pour**

4.4.3 Coût du tri par insertion

Notons $c_{\text{tri-insert}}(n)$ le nombre de comparaisons d'éléments du tableau à effectuer pour trier un tableau de longueur n par le tri par insertion. Et notons $c_{\text{inserer}}(k)$ le nombre de comparaisons à effectuer pour inserer un élément dans une tranche triée de tableau de longueur k selon l'algorithme 3.3.

Exemple 4.4 :

Commençons par examiner le nombre de comparaisons effectuées dans le tri du tableau de l'exemple 3.3.

En reprenant le tableau 3.2, voici le nombre de comparaisons d'éléments de t effectuées à chaque étape.

i	1	2	3	4	5	6	7
$c_{\text{inserer}}(i+1)$	1	2	2	4	5	2	4

Soit au total $1+2+\ldots+2+4=20$ comparaisons.

Théorème 4.2 (*Coût du tri par insertion*)**.** *Pour tout tableau de longueur n, le nombre de comparaisons d'éléments du tableau pour le trier par le tri par insertion est encadré par*

$$n-1 \leq c_{\text{tri-select}}(n) \leq \frac{(n-1)(n-2)}{2} \sim \frac{n^2}{2}.$$

La minoration (meilleur des cas) est atteinte pour un tableau trié. Et la majoration (pire des cas) est atteinte pour un tableau trié dans l'ordre inverse.

On peut prouver que le coût moyen du tri par insertion est équivalent à $\frac{n^2}{4}$.

4.5 Le tri par dénombrement

Le tri qui va être présenté ici diffère des tris précédents car il n'opère pas par comparaison des éléments du tableau à trier. Il consiste essentiellement à compter le nombre d'occurrences de chacun des éléments du tableau.

Cet algorithme nécessite que soit connu et fixé par avance les éléments susceptibles d'appartenir au tableau, et que le nombre de ces éléments soit relativement limité. Il n'est pas adapté au tri de tableaux de nombres (entiers ou flottants), ou bien de chaînes de caractères, si aucune restriction sur ces nombres ou ces chaînes pouvant figurer dans le tableau. En revanche, il convient très bien dans le cas où le nombre des valeurs possibles pour les éléments est limité. Par exemple, un tableau ne pouvant contenir que des entiers de 1 à 10, ou bien un tableau ne pouvant contenir que les lettres de l'alphabet latin (caractères compris entre `A` et `Z`).

Désignons par m le nombre des valeurs possibles contenues dans un tel tableau ($m = \text{card}(E)$).

Exemple 4.5 :
Prenons pour exemple l'ensemble des lettres de l'alphabet latin.

$$E = \{\texttt{A}, \texttt{B}, \ldots, \texttt{Z}\}$$
$$m = 26.$$

On veut trier le tableau de lettres

$t =$

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30
T	I	M	O	L	E	O	N	E	S	T	U	N	H	O	M	M	E	P	O	L	I	T	I	Q	U	E	G	R	E	C

.

On commence par compter le nombre d'occurrences dans t de chacune des 26 lettres de l'alphabet. Pour cela on initialise un tableau de 26 compteurs (un par lettre).

$cpt =$

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

.

Puis en parcourant séquentiellement le tableau t, on compte chacune des 26 lettres. On obtient

$cpt =$

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
0	0	1	0	5	0	1	1	3	0	0	2	3	2	4	1	1	1	1	3	2	0	0	0	0	0

.

Enfin on transforme ce tableau en remplaçant les effectifs en effectifs cumulés.

$cpt =$

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
0	0	1	1	6	6	7	8	11	11	11	13	16	18	22	23	24	25	26	29	31	31	31	31	31	31

.

Ceci fait on construit en parcourant séquentiellement le tableau t, on place l'élément $t(i)$ de ce tableau dans un tableau t' à l'indice donné par le tableau cpt que l'on aura préalablement décrémenté.

$t' =$

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30
C	E	E	E	E	E	G	H	I	I	I	L	L	M	M	M	N	N	O	O	O	O	P	Q	R	S	T	T	T	U	U

.

Algorithme 4.5 Algorithme de tri par dénombrement

Entrée : t un tableau de longueur n dont les éléments sont pris dans un ensemble qui en contient m.
Sortie : t' un tableau trié de longueur n contenant les mêmes éléments.

1: {Compter les éléments de t}
2: initialiser à 0 un tableau cpt de compteurs de longueur m
3: **pour** chaque élément e de t **faire**
4: incrémenter le compteur qui lui correspond dans le tableau cpt
5: **fin pour**
6: {Transformer cpt pour obtenir les effectifs croissants}
7: **pour** k variant de 1 à $m-1$ **faire**
8: $cpt(k) := cpt(k-1) + cpt(k)$
9: **fin pour**
10: Soit t' un tableau de longueur n
11: **pour** i variant de 0 à $n-1$ **faire**
12: Soit k le rang de $t(i)$
13: décrémenter $cpt(k)$
14: $t'(cpt(k)) := t(i)$
15: **fin pour**

4.6 Implantations en CAML

Dans ce qui suit, sont données les implantations en CAML des fonctions construisant une version triée du tableau passé en paramètre. Toutes ces fonctions vérifient la même spécification que nous donnons ci-dessous, et qui ne sera pas rappelée à chaque fois.

```
(*
  fonction tri_XXX : 'a array -> 'a array
  parametre
    t : 'a array
  valeur renvoyee : 'a array = une version triee du tableau t
  CU : aucune
*)
```

Il est très facile d'adapter l'implantation de ces fonctions pour en faire des procédures de tri qui transforment le tableau passé en paramètre en un tableau trié. La spécification commune de toutes ces procédures est alors

```
(*
  procedure trier_XXX : 'a array -> unit
  parametre
    t : 'a array
  action :  trie le tableau t
  CU : aucune
*)
```

4.6.1 Tri par sélection

Voici une implantation en CAML d'une fonction de tri suivant l'algorithme de tri par sélection présenté page 3. Elle est précédée d'une implantation d'une fonction de sélection de l'indice du minimum d'une tranche de tableau.

```
(* fonction indice_minimum : 'a array -> int -> int -> int
   parametres
     t : 'a array
     a : int
     b : int
   valeur renvoyee : int = indice du plus petit element de
           la tranche t(a..b)
   CU : 0 ≤ a ≤ b < n
*)
let indice_minimum t a b =
  let ind_min = ref a
  in
    for i = a + 1 to b do
      if t.(i) < t.(!ind_min) then
        ind_min := i
    done ;
    !ind_min ;;
```

```
let tri_selection t =
  let t' = Array.copy t
  and n = Array.length t
  in
    for i = 0 to n - 2 do
      (* recherche du plus petit element de la tranche t'.(i..n-1) *)
      let ind_min = indice_minimum t' i (n - 1)
      in
        (* echanger t'.(i) et t'.(ind_min) si necessaire *)
        if i <> ind_min then begin
          let aux = t'.(i)
          in
              t'.(i) <- t'.(ind_min) ;
              t'.(ind_min) <- aux
        end ;
    done ;
    t' ;;
```

4.6.2 Tri par insertion

Voici une implantation en CAML d'une fonction de tri suivant l'algorithme de tri par insertion présenté page 5. Cette fonction utilise la procédure `inserer` que voici

```
(*
  procedure inserer : 'a array -> int -> unit
  parametres
```

```
    t : 'a array
    i : int
  action : insere l'element t(i) a sa place dans
  la tranche t(0..i) de sorte qu'elle soit triee
  CU : 1 ≤ i < n et t(0..i − 1) triee
*)
let inserer t i =
  let k = ref i
  and aux = t.(i)
  in
    while (!k >= 1) && (aux < t.(!k - 1)) do
      t.(!k) <- t.(!k - 1) ;
      k := !k - 1
    done ;
    t.(!k) <- aux ;;
```

```
let tri_insertion t =
  let t' = Array.copy t
  and n = Array.length t
  in
    for i = 1 to n - 1 do
      inserer t' i
    done ;
    t' ;;
```

On peut aussi déclarer la fonction `inserer` localement dans la fonction `tri_insertion`. Il n'est alors plus nécessaire de passer le tableau en paramètre de la fonction `inserer` si on la déclare sous la portée de la déclaration de la variable locale `t'`.

```
let tri_insertion t =
  let t' = Array.copy t
  and n = Array.length t
  in
  let inserer i =
    let k = ref i
    and aux = t'.(i)
    in
      while (!k >= 1) && (aux < t'.(!k - 1)) do
        t'.(!k) <- t'.(!k - 1) ;
        k := !k - 1
      done ;
      t'.(!k) <- aux
  in
    for i = 1 to n - 1 do
      inserer i
    done ;
    t' ;;
```

4.6.3 Tri par dénombrement

Voici une implantation en CAML d'une fonction de tri suivant l'algorithme de tri par dénombrement présenté page 7. Conformément à ce qui a été dit cette fonction nécessite deux fonctions de conversion des éléments à trier en indices entiers, et réciproquement. Ces fonctions sont

1. `element_en_indice` : `'a → int`
2. et `indice_en_element` : `int → 'a`

et sont réciproques l'une de l'autre.

```
let tri_denombrement t =
 let n = Array.length t
 in
   let cpt = Array.make m 0
   and t' = Array.make n t.(0)
   in
     for i = 0 to n - 1 do
       let k = element_en_indice t.(i)
       in
         cpt.(k) <- cpt.(k) + 1
     done ;
     for k = 1 to m - 1 do
       cpt.(k) <- cpt.(k-1) + cpt.(k)
     done ;

     for i = 0 to n - 1 do
       let k = element_en_indice t.(i)
       in
         cpt.(k) <- cpt.(k) - 1 ;
         t'.(cpt.(k)) <- t.(i)
     done ;
     t' ;;
```