Les algorithmes de tri

Le problème du tri

On désigne par "tri" l'opération consistant à ordonner un ensemble d'éléments en fonction de *clés* sur lesquelles est définie une relation d'ordre.

Les algorithmes de tri ont une grande importance pratique. Ils sont fondamentaux dans certains domaines, comme l'informatique de gestion où l'on tri de manière quasi-systématique des données avant de les utiliser.

L'étude du tri est également intéressante en elle-même car il s'agit sans doute du domaine de l'algorithmique qui a été le plus étudié et qui a conduit à des résultats remarquables sur la construction d'algorithmes et l'étude de leur complexité.

8.1 - Introduction

- Un tableau T est dit « **trié en ordre croissant** » si tous les éléments consécutifs du tableau vérifient :

$$T[i-1] \leq T[i]$$

- Il est admis qu'un
  - tableau vide est trié
  - tableau ne contenant qu'un seul élément est trié

8.1 - Introduction

- D'où la définition :
  - Un tableau vide (n=0) est ordonné (trié),
  - Un tableau contenant un seul élément (n=1) est ordonné,
  - Un tableau T[1..n], n>1, est ordonné si

$$\forall\ i \in [2..n],\ T[i-1] \leq T[i]$$

8.1 - Introduction

- Tri d'un tableau
  - Soit un vecteur (tableau à une dimension) T[1..n] à valeurs dans un ensemble E de valeurs muni d'une relation d'ordre notée <
  - Trier le vecteur T consiste à construire un vecteur T'[1..n] tel que :
    - T' soit trié,
    - T' et T contiennent les mêmes éléments.
  - Le plus souvent T et T' sont le même vecteur ; T' est construit en permutant entre eux les éléments de T .

8.2 – Tri par remplacement

- Cette méthode simple et intuitive est malheureusement très peu performante.
- Elle consiste à construire un tableau Ttrié[1..n] à partir de T[1..n] tel que :

$$\text{Ttrié}[i-1] \leq \text{Ttrié}[i] \;,\; \forall\, i \in [2..n]$$

- Principe :
  - Identifier le maximum du tableau
  - Rechercher le minimum du tableau T
  - Recopier ce minimum dans Ttrié à la position i
  - Remplacer le minimum du tableau T par le maximum
  - Recommencer pour i+1

8.2 – Tri par remplacement

Procédure tri_remplacement (D/R T[1..n], R Ttrié[1..n] : tableau de …)
Entier i,j
E max
Début
  max ← maximum(T[1..
  i ← 1
  tant que i<n **faire**
    j ← indice_min(T[1.n])
    Ttrié[i] ← T[j]
    T[j] ← max
    i ← i+1
  ftq
  Ttrié[n] ← max
Fin

Fonction maximum(D T[1..n] : tableau de …):E
Entier i
E max
Début
  max ← T[1]
  Pour i ← 2 à n **faire**
    si T[i]>max **alors**
      max ← T[i]
    fsi
  fpour
  Retour max
Fin

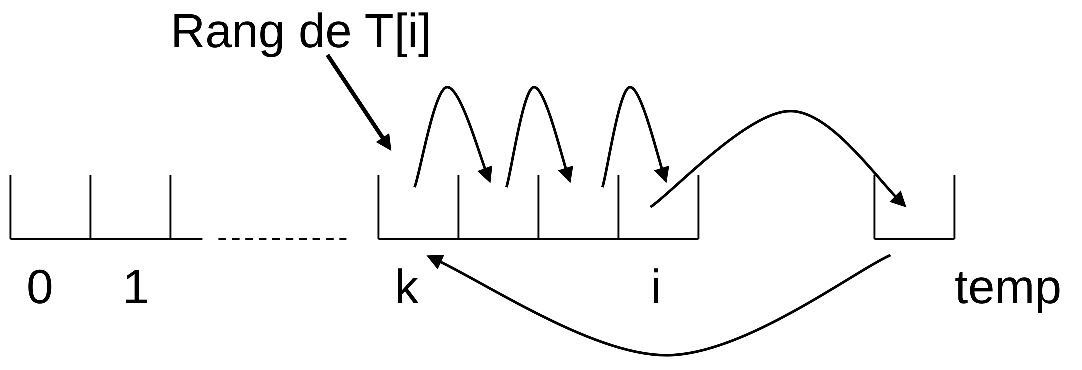
8.2 – Tri par remplacement

- Pour chaque élément rangé dans le tableau Ttrié, il faut parcourir tout le tableau T et non une partie du tableau T
- Nécessite un 2ème tableau, or si le nombre d'éléments à trier est important, cet algorithme requiert donc un espace mémoire double.

8.3 - Tri par insertion

- Cette méthode de tri insère (au $i^{ème}$ passage) le $i^{ème}$ élément T[i] à la bonne place parmi T[1],T[2]...T[i-1].
- Après l'étape i, tous les éléments entre la première et la $i^{ème}$ position sont triés.
- Il existe plusieurs méthode de tri par insertion selon le principe qui est utilisé pour rechercher le rang de l'élément à insérer parmi les éléments du début de la liste déjà triés

8.3 - Tri par insertion



- Principe de l'algorithme :
  - pour i ← 2 à n faire

    déplacer T[i] vers le début du tableau jusqu'à

    la position j<=i telle que

    T[j] < T[k] pour j<=k<i et (ou bien T[j]>=T[j-1] ou bien j=1).

8.3 - Tri par insertion

4	2	0	5	3	Vecteur de départ
2	4	0	5	3	Les cellules 1 à 2 sont triées
0	2	4	5	3	Les cellules 1 à 3 sont triées
0	2	4	5	3	Les cellules 1 à 4 sont triées
0	2	3	4	5	Les cellules 1 à 5 sont triées

8.3 - Tri par insertion

Procédure tri_insertion (D/R T[1..n], : tableau de …)
Entier i,j
Début
  pour i← 2 à n **faire**
    j ← i-1
    tant que j ≥ 1 **et** T[j] > T[j+1] **faire**
      échange(T,,j+1,,j)
      j ← j-1
    ftq
  fpour
Fin

8.4 – Tri par sélection

- Le principe du tri par sélection d'un vecteur est d'aller chercher le plus petit élément du vecteur pour le mettre en premier, puis de repartir du second, d'aller chercher le plus petit élément pour le mettre en second etc.
- Au $i^{ème}$ passage, on sélectionne l'élément ayant la plus petite valeur parmi les positions i..n et on l'échange avec T[i].

8.4 – Tri par sélection

4	2	0	5	3	Vecteur de départ
0	2	4	5	3	Le plus petit élément est à sa place
0	2	4	5	3	Les 2 plus petits éléments sont à leur place
0	2	3	5	4	Les 3 plus petits éléments sont à leur place
0	2	3	4	5	Les 4 plus petits éléments sont à leur place

8.4 – Tri par sélection

```
Procédure tri_sélection (D/R T[1..n], : tableau de …)
Entier i,j
Début
  pour i← 1 à n faire
    pour j← i+1 à n faire
     si T[i] > T[j] alors
       échange(T,i,j)
     fsi
    fpour
  fpour
Fin
```

8.5 – Tri à bulles

- Le principe du tri à bulles *(bubble sort)* est de comparer deux à deux les éléments e_1 et e_2 consécutifs d'un tableau et d'effecteur une permutation si $e_1 > e_2$.
- On continue de trier jusqu'à ce qu'il n'y ait plus de permutation.

8.5 – Tri à bulles

4	2	0	5	3	Vecteur de départ
4	0	2	3	5	Fin du premier passage
0	2	3	4	5	Fin du deuxième et dernier passage

8.5 – Tri à bulles

```
Procédure tri_bulles (D/R T[1..n], : tableau de …)
Entier i,j
Début
  pour i← 1 à n-1 faire
    pour j← n à i+1 faire {décroissant}

     si T[j-1] > T[j] alors
       échange(T,j-1,j)
     fsi
    fpour
  fpour
Fin
```

8.5 – Tri à bulles

- Évaluation du coût de l’algorithme
  - Les comparaisons sont effectuées à l’intérieur de la boucle la plus interne
  - Pour i=1, n-1 comparaisons sont effectuées
  - Pour i=2, n-2 comparaisons sont effectuées
  - … … … …
  - Pour i=n-1, 1 comparaison est effectuée
  - Le nombre total de comparaisons effectuées est donc :

$$\sum_{k=1}^{n-1} k = \frac{n(n-1)}{2}$$

8.5 – Tri à bulles

- Optimisation de l'algorithme
  - Après avoir traité i-1 éléments ($1 \leq i \leq n$)
    - Les éléments de 1..i-1 sont triés
    - Tous les éléments compris entre 1..i-1 sont inférieurs aux éléments compris entre i..n

$$T[1..i-1] \leq T[i..n]$$

    - Les éléments compris entre i..n ne sont pas traités
  - Si les éléments compris entre i..n sont triés à la suite des permutations effectuées, il est inutile de poursuivre car :
    - $T[1..i-1]$ trié ; $T[1..i-1] \leq T[i..n]$; $T[i..n]$ trié $\Rightarrow$ $T[1..n]$ trié
  - Exemple de BATEAUX

```
Procédure tri_bulles2 (D/R T[1..n], : tableau de …)
Entier i,j
Booléen onapermuté
Début
  i← 1
 onapermuté ← VRAI
  Tant que onapermuté faire
    onapermuté ← FAUX
    pour j← n à i+1 faire {décroissant}
     si T[j-1] > T[j] alors
       échange(T,j-1,j)
       onapermuté ← VRAI
     fsi
    fpour
    i← i+1
  ftq
Fin
```

```
PRODECURE Tri_bulle (D/R T[1..n], : tableau de …)
Booleen permut
Début
  REPETER
    permut ← FAUX
    POUR i ← 1 à N-1 FAIRE
      SI T[i] > T[i+1] ALORS
        échange(T,i,i+1)
        permut ← VRAI
      FSi
    Fpour
  Jusqu'à permut = FAUX
FIN
```

8.6 – Tri indirect

- Ce tri utilise un tableau auxiliaire qui indique, pour chaque élément du tableau à trier, le rang que celui-ci devrait occuper dans le tableau trié.
- Le principe consiste à compter, pour chaque élément du tableau à trier, le nombre d'éléments qui lui sont inférieurs ou égaux. Le nombre trouvé donnera la place (l'indice) de cet élément dans le tableau trié
- Le tri se fait donc ensuite par l'intermédiaire de cet index

8.6 – Tri indirect

<u>Procédure</u> tri_indirect (D/R T[1..n], : tableau de …)
<u>Entier</u> i
Tableau d'entier ind
<u>Début</u>
  <u>pour</u> i← 1 à n **<u>faire</u>**
    ind[i] ← 1
  <u>fpour</u>

COMPTAGE DES ELEMENTS

  <u>pour</u> i← 1 à n **<u>faire</u>**
    Ttrié[ind[i]] ← T[i]
  <u>fpour</u>
<u>Fin</u>

8.6 – Tri indirect

COMPTAGE DES ELEMENTS

```
pour i← 1 à n-1 faire
  pour k← 1 à n faire
    si T[k] < T[i] alors
      ind[i] ← ind[i] + 1
    fsi
  fpour
fpour
```

8.6 – Tri indirect

AMELIORATION DU COMPTAGE DES ELEMENTS

```
pour i← 1 à n-1 faire
  pour k← i+1 à n faire
    si T[i] < T[k] alors
      ind[i] ← ind[i] + 1
    sinon
      ind[k] ← ind[k] + 1
    fsi
  fpour
fpour
```

Fin