Université Assane SECK de Ziguinchor

Unité de Formation et de Recherche des Sciences et Technologies

Département d'Informatique

Les algorithmes de tri

Licence 1 en Ingénierie Informatique

Octobre 2021

Papa-alioune.cisse@univ-zig.sn

Résumé : Un tri consiste à ordonner de façon croissante, décroissante, (ou suivant un critère quelconque) une suite d'éléments à partir des clés qui leur sont associées. Pour une suite de n éléments, il s'agit donc de trouver une permutation particulière des éléments parmi les n! permutations possibles.

1 - DÉFINITIONS ET EXEMPLES

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.

- **Exemple 1 : tableau d'entiers**

Donnons-nous un tableau de nombres. Le problème est de trier ces nombres dans l'ordre croissant. En d'autres termes, nous devons modifier le tableau pour qu'il contienne les même nombres qu'auparavant, mais cette fois les nombres sont rangés dans l'ordre croissant, du plus petit au plus grand. Ici la clé de tri des éléments du tableau est le nombre situé dans chaque case du tableau.

Le résultat du tri du tableau :

6	3	7	2	3	5

est le suivant :

2	3	3	5	6	7

- **Exemple 2 : Annuaire téléphonique**

Dans un annuaire téléphonique, les abonnés sont classés par ordre alphabétique, d'abord sur les noms de famille (***clé primaire***), puis en cas d'homonymie, par ordre alphabétique sur les prénoms (***clé secondaire***). Ici, la clé est dite ***complexe*** avec 2 ***niveaux*** (contrairement à l'exemple 1 où la clé est ***simple***).

Remarque : La complexité de la clé ne modifie pas les algorithmes de tri, seule la méthode de comparaison des clés change.

- **Exemple 3. Tableau d'étudiants**

Un tableau contenant l'ensemble des étudiants d'une classe avec leurs notes et les moyennes calculées d'une matière. Pour délibérer dans cette matière, on peut trier le tableau par ordre de mérite selon la moyenne. En cas de moyennes égales, par ordre alphabétique suivant le nom, puis le prénom. Si le tri concerne uniquement la moyenne, la clé est donc simple.

2 - TRI PAR SÉLECTION

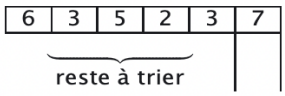
2.1 - Principe du tri par sélection

Soit un tableau ***T*** de ***n*** nombres. Le principe du tri par sélection consiste à sélectionner le maximum des nombres se trouvant entre les positions ***1*** et ***n*** et de l'échanger avec l'élément à la position ***n***. Cela fait, le maximum est en dernière position du tableau qui est sa position effective. Ce procédé est itéré pour chaque k allant de n à 1. On peut faire l'inverse en sélectionnant le minimum, l'échanger avec l'élément à la position 1 et itérer pour k allant de 1 à n.

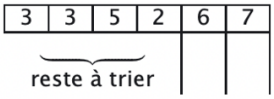
Exemple : soit le tableau suivant de 6 entiers :

6	3	7	2	3	5

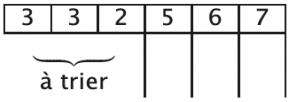
Pour k=6, le maximum du tableau à trier est 7. Il est échangé avec l'élément à la dernière position :



Pour k=5, le maximum du tableau à trier est 6, il est échangé avec le dernier :



Pour k = 4, le maximum du reste à trier est 5, il est échangé avec le dernier :



Pour k=3, le maximum du reste à trier est le premier 3, il est échangé avec le dernier :

2	3	3	5	6	7

Pour k=2 et 1, le tableau est déjà trié.

2.2 - Algorithme du tri par sélection

```
Procédure tri_sélection(var t : tab, n : Entier)
            Var i, k, imax, tmp : Entier
            Début
              Pour i de n à 1 pas -1 faire
                Début
                  {Recherche de la position du maximum (imax)}
                  imax ← 1
                  pour k de 1 à i faire
                    si (t[imax] < t[k] alors
                      imax ← k
                  {échange du max avec le dernier}
                  tmp ← t[i]
```

```
            t[i] ← t[imax]
            t[imax] ← tmp
        Fin pour
    Fin
```

3 - TRIE PAR INSERTION

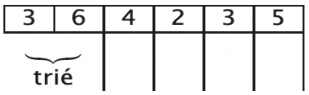
3.1 - Principe du tri par insertion

Soit ***T*** un tableau de ***n*** nombres. Le principe du tri par insertion consiste à supposer que le sous tableau constitué du premier élément du tableau (tableaux à gauche) est trié et que le sous tableau constitué du reste (tableau à droite) ne l'est pas. Il convient maintenant de prendre le premier élément du sous tableau à droite (ce qui le réduit) et de l'insérer à sa bonne place dans le sous tableau à gauche qui est trié (ce qui le grossit). Ce procédé est répété jusqu'à insérer tous les éléments du sous tableau à droite dans leurs bonnes places à gauche. Pour insérer un élément dans un tableau à sa bonne place, il faut déplacer les éléments vers la droite jusqu'à arriver à la bonne place de l'élément et de l'y insérer.

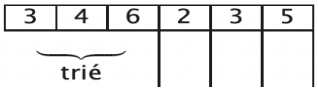
Exemple : soit le tableau suivant de 6 entiers, dont le premier élément (6) est supposé trié :

6	3	4	2	3	5

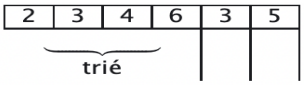
En insérant le deuxième élément à sa bonne place, on obtient :



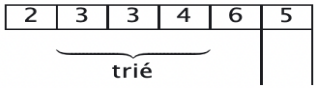
En insérant le 3ème élément à sa bonne place, on obtient :



En insérant le 4ème élément à sa bonne place, on obtient :



En insérant le 5ème élément à sa bonne place, on obtient :



En insérant le 6ème élément à sa bonne place, on obtient :

3.2 - Algorithme du tri par insertion

```
Procédure tri_insertion(var t : tab, n : Entier)
          Var i, j, v : Entier
```

```
Début
  Pour i de 2 à n pas 1 faire
    Début
      {insérertion t[i] à sa bonne place}
      v ← t[i]
      j ← i - 1
      Tant que (j >=0 ET v < t[j]) faire
        Début
          t[j+1] ← t[j]
          j ← j - 1
        Fin Tant que
      t[j+1] ← v
    Fin pour
Fin
```

4 - TRI PAR BULLES

4.1 - Principe du tri par bulles

Dans le tri par bulles, on échange deux éléments successifs T[i-1] et T[i] s'ils ne sont pas dans l'ordre. Évidemment, il faut faire cela un grand nombre de fois pour obtenir un tableau trié. Plus précisément, on fait remonter le maximum à sa place définitive par échanges successifs avec ses voisins.

Exemple : avec le tableau

6	3	4	2	3	5

Voici la succession des échanges d'éléments successifs effectués.

6	3	4	2	3	5

3	6	4	2	3	5

3	4	6	2	3	5

3	4	2	6	3	5

3	4	2	3	6	5

3	4	2	3	5	6

Un deuxième passage est fait pour obtenir :

3	2	4	3	5	6

3	2	3	4	5	6

Un troisième passage est effectué pour obtenir :

2	3	3	4	5	6

Pour les autres passages, le tableau est déjà trié.

4.2 - Algorithme du tri par bulles

```
Procédure tri_bulles(var t : tab, n : Entier)
            Var i, j, tmp : Entier
            Début
              Pour i de n à 1 pas -1 faire
                Début
                  Pour j de 2 à i pas 1 faire
                    Si (t[j] < t[j-1]) alors
                      Début
                        tmp ← t[j]
                        t[j] ← t[j-1]
                        t[j-1] ← tmp
                      Fin Si
                Fin pour
            Fin
```

5 - Le tri rapide (ou quicksort)

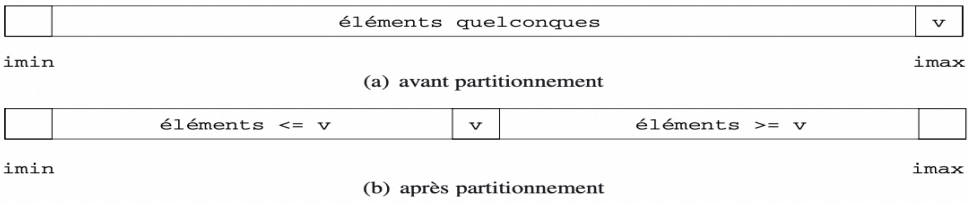
5.1 - Principe du tri rapide

L'algorithme de tri rapide, ("quick sort" en anglais), est un algorithme de type dichotomique. Son principe consiste à séparer l'ensemble des éléments en deux parties (on appelle cela un ***partitionnement***). Pour effectuer le partitionnement, une valeur ***pivot*** est choisie. Les valeurs sont réparties en deux ensembles suivant qu'elles sont plus grandes ou plus petites que le pivot. Ensuite, les deux ensembles sont triés séparément, suivant la même méthode. L'algorithme, est récursif, mais il n'est pas nécessaire de fusionner les deux ensembles. Le résultat du tri est égal au tri de l'ensemble dont les valeurs sont inférieures au pivot concaténé à l'ensemble des valeurs supérieures au pivot.

5.2 - Le principe du partitionnement dans le tri rapide

Le choix du pivot est le problème central de cet algorithme. En effet, l'idéal serait de pouvoir répartir les deux ensembles en deux parties de taille à peu près égales. Cependant, la recherche du pivot qui permettrait une partition parfaite de l'ensemble en deux parties égales aurait un coût trop important. C'est pour cela que le pivot est choisi de façon aléatoire parmi les valeurs de l'ensemble. Dans la pratique, le pivot est le premier ou le dernier élément de l'ensemble à fractionner. En moyenne, les deux ensembles seront donc de taille sensiblement égale.

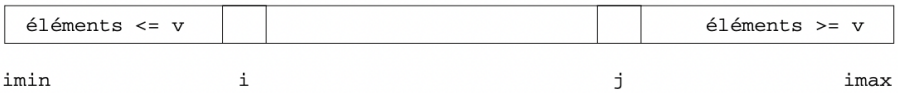
Supposons que l'on souhaite trier une partie d'un tableau T entre les indices ***imin*** et ***imax***. On choisit une valeur pivot, par exemple ***v = T[imax]***. On veut séparer dans le tableau les éléments inférieurs à v et les éléments supérieurs à ***v***.



(a) avant partitionnement

(b) après partitionnement

Pour faire cela, on introduit un indice ***i***, initialisé à ***imin***, et un indice ***j***, initialisé à ***imax-1***. On fait ensuite remonter l'indice ***i*** jusqu'au premier élément supérieur à ***v***, et on fait redescendre ***j*** jusqu'au premier élément inférieur à ***v***. On échange alors les éléments d'indices ***i*** et ***j***. On itère ce procédé tant que $i \leq j$. À la fin, on place la valeur ***pivot*** en position ***i***.



5.3 - Algorithme du partitionnement

```
Fonction partitionner (var t : tab, imin : Entier, imax : Entier) : Entier
            Var i, j, v, tmp : Entier
            Début
              v ← t[imax]
              i ← imain
              j ← imax - 1
              Tant que (i <= j) Faire
                Début
                  Tant que (i< imax ET t[i] <= v) Faire
                      i ← i + 1
                  Tant que (j >= imin ET t[j] >= v) Faire
                      j ← j - 1
                  Si (i < j) alors
                    Début
                      tmp ← t[i]
                      t[i]← t[j]
                      t[j]← tmp
                    Fin SI
                Fin Tant que
                t[imax] ← t[i]
                t[i] ← v
                partitionner ← i
            Fin
```

5.4 - Algorithme du tri rapide

```
Procédure tri_rapide(var t : tab, imin : Entier, imax : Entier)
            Var i: Entier
            Début
              Si (imin < imax) alors
                Début
                  i ← partitionner (t, imin, imax)
                  tri_rapide(t, imin, i-1)
                  tri_rapide(t, i+1, imax)
                Fin Si
            Fin

Procédure quicksort(var t : tab ; n : Entier) ;
            Début
              Tri_rapide(t, 1, n)
            Fin
```