

TD n°5

Algorithmes de Tri

L'objet de ce TD est de manipuler différents algorithmes de tri et d'étudier leur complexité. On se donne à chaque fois un tableau d'entiers et le problème consiste à modifier l'ordre des éléments de telle sorte qu'ils soient triés après l'application de l'algorithme.

Exercice 1 *Tri par Dénombrement*

Une méthode de tri applicable à une situation particulière : le tri par dénombrement. Il s'agit de trier une suite de n nombres entiers dont les valeurs sont comprises entre 0 et k .

Si k n'est pas très grand on peut utiliser l'algorithme suivant :

- Soit s le tableau contenant la suite, on calcule (à l'aide d'un autre tableau auxiliaire `compte` de taille $k + 1$), la fréquence de chaque entier dans s .
- On modifie `compte` de façon à avoir dans `compte[i]` le nombre d'éléments de s ayant une valeur inférieure ou égale à i .
- On utilise le contenu de `compte` pour construire la suite triée dans un deuxième tableau auxiliaire t en utilisant l'instruction suivante :

pour tout i allant de n à 1 par pas de -1 faire :

```
t[compte[ s[i] ] ] = s[ i ];
compte[ s[i] ]=compte[s [i] ]-1;
```

finpour

- On recopie la suite triée dans le tableau s .

1. Dérouler l'algorithme de tri par dénombrement sur le tableau :

$A = [7, 1, 3, 1, 2, 4, 5, 7, 2, 4, 3]$

2. Écrire une méthode pour calculer la fréquence des éléments d'une suite de n entiers positifs ayant une valeur inférieure à k .
3. Écrire une méthode qui effectue le tri par dénombrement d'une suite de n entiers positifs ayant une valeur inférieure à k .

Exercice 2 *Tri par Distribution*

Cette méthode s'applique aux tris de nombres et aux tris alphabétiques. Si nous considérons une suite de nombres à nbc chiffres (ou une suite de mots de nbc caractères), le tri s'effectue par distribution successive en considérant d'abord les chiffres des unités puis celui des dizaines, des centaines, etc ... À chaque passage nous formons des groupes de nombres suivant ce critère, puis nous reconstituons à partir de ces groupes une nouvelle suite qui sera utilisée pour la distribution suivante. Le nombre de distributions est égal au nombre de chiffres du plus grand nombre.

Application sur une suite de nombre

Soit une suite de 12 nombres (d'au plus 3 chiffres) :

3 127 37 51 18 172 25 45 7 33 11 131

Pour la première distribution, nous considérons le chiffre des unités et nous obtenons les groupes suivants :

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|-----|-----|----|---|----|---|-----|----|---|
| | 51 | 172 | 3 | | 25 | | 127 | 18 | |
| | 11 | | 33 | | 45 | | 37 | | |
| | 131 | | | | | | 7 | | |

Pour recréer la suite, nous prenons successivement les groupes formés dans l'ordre croissant des chiffres, soit :

51 11 131 172 3 33 25 45 127 37 7 18

En faisant maintenant la distribution suivant le chiffre des dizaines, nous obtenons les groupes suivants :

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|----|----|-----|-----|----|----|---|-----|---|---|
| 03 | 11 | 25 | 131 | 45 | 51 | | 172 | | |
| 07 | 18 | 127 | 33 | | | | | | |
| | | | 37 | | | | | | |

La nouvelle suite a alors la configuration ci-dessous :

03 07 11 18 25 127 131 33 37 45 51 172

Nous faisons la dernière distribution sur le chiffre des centaines :

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|-----|-----|---|---|---|---|---|---|---|---|
| 003 | 127 | | | | | | | | |
| 007 | 131 | | | | | | | | |
| 011 | 172 | | | | | | | | |
| 018 | | | | | | | | | |
| 025 | | | | | | | | | |
| 033 | | | | | | | | | |
| 037 | | | | | | | | | |
| 045 | | | | | | | | | |
| 051 | | | | | | | | | |

Et nous obtenons finalement les nombres triés :

003 007 011 018 025 033 037 045 051 127 131 172

1. Définir une classe `Etudiant` et une classe `ListeEtudiants` avec pour chacune un constructeur initialisant les champs à des valeurs données. La liste des étudiants sera stockée dans un tableau, qui sera privé. On prévoira également un champ contenant le nombre d'éléments de ce tableau.
2. (*Sélection du moins bon*) Ecrire une méthode `minimum(int n)` trouvant la valeur `i` telle que le `i`-ième étudiant de la liste est celui qui, parmi les étudiants des `n` premières entrées du tableau, a obtenu (ou l'un de ceux qui ont obtenu) la moins bonne note.
3. (Tri par sélection) En déduire une procédure ordonnant les étudiants de la liste par ordre de notes décroissantes. On rappelle le pseudo-code du tri par sélection (**par ordre croissant**) :

```

TRI-SELECTION
Pour i <--- 1 a n-1 faire
    imin <--- i
    Pour j <-- i+1 a n faire
        Si A[j] < A[imin] Alors imin <--- j
    Fin Pour
    ECHANGER A[i] et A[imin]
Fin Pour

```

4. Écrire une méthode `toString()` qui affiche le nom, le prénom de chaque étudiant ainsi que sa note.

Exercice 5 optionnel : *Comparaison de Tableaux*

- Deux tableaux sont *égaux* si ils contiennent les mêmes éléments aux mêmes positions.
 - Deux tableaux sont *similaires* si ils contiennent les mêmes éléments mais pas forcément aux mêmes positions. Par exemple : `[9,6,7,6,3,6,1,9]` et `[1,3,6,6,6,7,9,9]` sont similaires alors que `[9,6,7,6,3,6,1,9]` et `[9,9,7,6,3,6,1,9]` ne le sont pas.
 - Deux tableaux sont *comparables* si l'ensemble des valeurs qu'ils contiennent est le même. Par exemple, les tableaux `[1,3,6,7,9]`, `[6,7,3,6,1,9]` et `[9,6,7,6,3,6,1,9]` sont tous les trois comparables.
1. Écrire une méthode qui teste si deux tableaux sont égaux.
 2. Écrire une méthode qui teste si deux tableaux sont similaires.
 3. Écrire une méthode qui teste si deux tableaux sont comparables .
(*idée : écrire une méthode qui efface dans le tableau, toutes les occurrences d'un entier*)