

## Séance 6: TABLEAUX D'ENTRIERS ALÉATOIRES; COMPRESSION DE TEXTES

Université Paris-Diderot

Objectifs:

- Manipuler des tableaux unidimensionnels d'entiers et de chaînes de caractères.
- Trier un tableau d'entiers par comptage.

### Exercice 1 (Tableaux d'entiers aléatoires, ☆)

Le but de cet exercice est de mettre en œuvre des opérations simples sur des tableaux d'entiers, aboutissant à une méthode de tri particulièrement simple et efficace pour des tableaux contenant des entiers "pas trop grands". Vous allez compléter le fichier `RandomArray.java`. Les tests seront à placer dans la fonction `main` de ce fichier. Vous disposez de la fonction `boolean IntArrayEquals(int[] a, int[] b)` qui vérifie si ses deux arguments sont égaux (même longueur, mêmes éléments rang par rang), et de la procédure `void printIntArray (int[] a)` qui affiche le contenu du tableau `a`.

1. Écrire une fonction `int[] createRandomArray(int n)` qui renvoie un tableau d'entiers de taille `n`, dont les cases sont remplies par des entiers aléatoires compris entre 0 et  $(n-1)$ . L'appel de fonction `rand.nextInt(n)` renvoie de tels entiers. Pour tester la fonction, supprimez les commentaires des quatre premières lignes du `main`.
2. Écrire une fonction `int[] minMaxAverage(int[] a)` qui renvoie un tableau de taille 3 contenant dans sa première case le minimum, dans sa deuxième le maximum et dans sa troisième la partie entière de la moyenne des éléments de `a`.

**Contrat:**

L'appel

```
printIntArray(minMaxAverage(createRandomArray(100)))
```

peut afficher, par exemple,

```
0 96 46
```

Les valeurs exactes sont imprévisibles pour un tel appel, à cause du remplissage aléatoire.

3. Écrire une fonction `int[] occurrences(int[] a)` qui renvoie un tableau contenant, à la case d'indice `i`, le nombre d'occurrences de `i` dans `a`. On supposera que les entiers contenus dans `a` sont positifs ou nuls. La taille du tableau `occurrences(a)` est  $1+(\text{minMaxAverage}(a))[1]$ .

**Contrat:**

Si `a` est le tableau `{1,3,0,0,0,1}`, `occurrences(a)` renvoie `{3,2,0,1}`,

4. Une fois obtenu le tableau `occurrences(a)`, il est possible de trier<sup>1</sup> `a` très simplement : en commençant à l'indice 0, on insère autant de 0 qu'indiqué dans la première case du tableau des occurrences,

---

1. Trier un tableau d'entiers `a` consiste en la construction d'un tableau qui contient les mêmes éléments que `a`, disposés en ordre croissant : l'entier contenu dans la case d'indice `i` n'est pas plus grand que l'entier contenu dans la case d'indice  $(i+1)$ , pour tout indice.

puis autant de 1 qu'indiqué dans sa deuxième case et ainsi de suite. Appliquée à l'exemple précédente, cette méthode produit le tableau {0,0,0,1,1,3} (d'abord 3 0, puis 2 1, puis 0 2, puis 1 3), qui est bien ce qu'on voulait.

- (a) Écrire une fonction `int[] countingSort(int[] a)`, qui utilise `occurrences`, et renvoie un tableau trié contenant les mêmes éléments que `a` en utilisant l'algorithme décrit ci-dessus<sup>2</sup>.
- (b) Écrire une procédure `void countingSort2(int[] a)`, qui utilise le même procédé que `countingSort` pour trier `a` sur place : le contenu de `a` change suite à l'appel `countingSort2(int[] a)`.

**Contrat:**

L'exécution de :

```
int[] a = createRandomArray(100);
int[] b = countingSort(a);
countingSort2(a);
System.out.println(intArrayEquals(a,b));
affiche true.
```

□

## Exercice 2 (Compression de textes, ★)

Dans cet exercice, vous allez mettre en œuvre un algorithme simple de compression de textes. Un texte est un tableau de chaînes de caractères `tab`, contenant un mot par case. La compression se fait en deux étapes : on construit d'abord un tableau de chaînes de caractères `lex` qui contient tous les mots du texte, sans répétitions. On appellera ce tableau le *lexique* du texte. Ensuite, le texte est encodé par un tableau d'entiers, de même longueur que `tab`, contenant à la case d'indice `i` l'indice du mot `tab[i]` dans le tableau `lex`. Par exemple, le texte {"être", "ou", "ne", "pas", "être"} a comme lexique le tableau {"être", "ou", "ne", "pas"}, et comme code le tableau {0,1,2,3,0}.

Vous allez compléter le fichier `Compression.java`. Les tests seront à placer dans la fonction `main` de ce fichier. Vous disposez de la fonction `boolean stringArrayEquals(String[] a, String[] b)` qui vérifie si ses deux arguments sont égaux (même longueur, mêmes éléments rang par rang), et de la procédure `void printStringArray(String[] a)` qui affiche le contenu du tableau `a`. Vous disposez aussi de la fonction `void loadText()` qui copie le contenu du fichier `text.txt` dans le tableau `text`, un mot par case. Vous allez l'utiliser pour tester vos fonctions.

### Construction du lexique

1. Écrire une fonction `boolean isIn(String s, String[] lex)`, qui vérifie si la chaîne `s` est dans le tableau `lex`.
2. Écrire une fonction `String[] extendLexicon (String s, String[] lex)`, qui renvoie un tableau de taille `lex.length+1`, qui contient `s` dans sa dernière case, et qui contient la chaîne `lex[i]` dans sa case d'indice `i`, pour  $0 \leq i < \text{lex.length}$ .
3. Pour construire le lexique d'un texte `tab`, on procède comme suit :
  - On déclare un tableau `String[] lex` de taille 0.
  - Pour chaque case `i` de `tab` on vérifie si `tab[i]` est dans `lex`. Si ce n'est pas le cas, on étend `lex`, par une affectation `lex=extendLexicon(tab[i],lex)`;
  - On renvoie `lex`Écrire une fonction `static String[] buildLexicon (String[] tab)`, qui construit le lexique du texte contenu dans `tab`, en mettant en œuvre l'algorithme donné ci-dessus.

**Contrat:**

---

2. Pour les tableaux engendrés par `createRandomArray(n)` ce procédé est efficace car ils ne contiennent que des entiers allant de 0 à (n-1). Pour des tableaux quelconques, la taille potentiellement très grande du tableau des occurrences peut rendre cette méthode très inefficace.

L'exécution de :

```
loadText();  
System.out.println(text.length+" "+(buildLexicon(text)).length);  
affiche  
1002 521
```

### Codage et Décodage

1. Écrire une fonction `int` `getCode(String s, String[] lex)`, qui renvoie le premier indice de `s` dans `lex`, ou `-1` si `s` n'est pas dans `lex`.
2. Écrire une fonction `int[]` `code(String[] tab, String[] lex)`, qui renvoie l'encodage du texte contenu dans `tab` relativement au lexique `lex`, c'est à dire un tableau d'entiers de même taille que `tab`, contenant à la case d'indice `i` l'indice du mot `tab[i]` dans `lex`.
3. Écrire une fonction `String[]` `decode(int[] code, String[] lex)`, qui reconstruit un texte à partir de son code et du lexique.

#### **Contrat:**

Encodez `text`, puis décodez-le. Vérifier que vous obtenez un texte identique à l'original, en utilisant `stringArrayEquals`.

□