Yann.Secq@univ-lille.fr
Fabien.Delecroix@univ-lille.fr

TP n°8

Objectifs: Exercices plus complexes sur les tableaux et révisions.

1 Nuage de mots

Un nuage de mots (voir la figure) permet de montrer visuellement la fréquence d'apparition de certains mots clés dans un texte.

FIGURE 1 – Le nuage de mot fabriqué depuis le sujet de cet exercice. Les mots les plus grands sont ceux qui apparaissent le plus souvent dans ce sujet.

Pour produire un nuage de mots, on a besoin de connaitre la fréquence d'apparition des différents mots clés. Dans les exercices qui suivent vous allez implémenter un ensemble de fonctions qui permettent de partir d'un texte (une chaine de caractères) et calculer la fréquence d'apparition de chacun des mots dans ce texte.

Vous devez faire les exercices dans l'ordre où ils sont présentés. Chaque exercice demande d'écrire une fonction et contient un test. À la fin vous verrez comment les différentes fonctions permettent de calculer les fréquences d'apparition de chaque mot dans un texte.

Tous les exercices doivent être faits dans le même fichier.

Exercice 1 : Nombre d'occurrences d'un mot dans un tableau [Í]

Ecrire la fonction `int nbFois(String[] tab, String mot)` qui calcule le nombre de fois que `mot` apparait dans le tableau `tab`. La fonction doit passer ce test.

```
void testNbFois () {
  String[] tab = new String[]{"je", "tu", "tu", "elle", "je", "je"};
  // "je" apparait 3 fois dans tab
  assertEquals(3, nbFois(tab, "je"));
  assertEquals(2, nbFois(tab, "tu"));
  assertEquals(0, nbFois(tab, "il"));
}
```

Exercice 2 : Calcul des fréquences [O]

n suppose qu'on dispose de deux tableaux de mots (`String[]`): un tableau `motsCles`, et un tableau `mots`. Pour chaque mot dans `motsCles`, on voudrait calculer combien de fois il apparait dans le tableau `mots`.

Écrire la fonction `int[] frequences (String[] motsCles, String[] mots)` qui produit en résultat un tableau d'entiers `freq` ayant la même taille que `motsCles` et tel que pour chaque indice `i`, la valeur `freq[i]` est

égale au nombre de fois que le mot `motsCles[i]` apparait dans le tableau `mots`. Pensez à utiliser la fonction `nbFois` écrite précédemment.

```
void testFrequences () {
  String[] motsCles = new String[]{"je", "tu", "elle"};
  int[] freq = new int[]{3, 2, 1};
  String[] mots = new String[]{"je", "tu", "tu", "elle", "je", "je"};
  assertArrayEquals(freq, frequences(motsCles, mots));
}
```

Exercice 3 : Extraire les mots différents d'un tableau de mots [Í]

Etant donné un tableau `mots` de String, dans lequel certains mots peuvent être répétés, on voudrait produire un autre tableau dans lequel chaque mot de `mots` apparait exactement une fois. Vous devez écrire la fonction `String[] motsDifferents (String[] mots)` qui passe ce test.

```
void testMotsDifferents () {
  String[] mots = new String[]{"je", "tu", "tu", "elle", "je", "je"};
  String[] motsDiff = motsDifferents(mots);
  for (int idx = 0; idx < length(mots); idx = idx+1){
    assertEquals(1, nbFois(motsDiff,mots[idx]));
    //on s'assure que chaque mot apparaît une et une seule fois dans motsDiff
  }
  //on vérifie que les mots sont bien à la bonne place
  assertEquals("je",motsDiff[0]));
  assertEquals("tu",motsDiff[1]));
  assertEquals("elle",motsDiff[2]));
  for (int idx = 3; idx < length(motsDiff); idx = idx+1){
    assertEquals("", motsDiff[idx]);
  }

}
```

La difficulté vient du fait qu'on ne peut pas savoir par avance quel est le nombre de mots différents qui apparaissent dans le tableau `mots` donné en paramètre. Par conséquent, on ne sait pas quelle va être la taille du tableau de résultat à créer.

Pour contourner ce problème, vous allez d'abord créer un tableau temporaire `motsDiff` suffisamment grand. Pour cela, pensez que le cas le plus large est celui où `mots` contient des mots tous différents. Commencez par initialiser chaque case par le mot vide `""`, pour vous assurez qu'aucune ne restera sans mot. Ajouter à présent les nouveaux mots rencontrés au fur et à mesure. Pour savoir si un mot donné est déjà contenu dans `motsDiff`, on peut utiliser la fonction `nbFois`.

Voici une illustration des calculs qui doivent être effectués par la fonction `motsDifferents` si le paramètre `mots` est le tableau

"je"	"tu"	"tu"	"elle"	"je"	"je"

.
Au départ, on initialise l'ensemble des cases de `motsDiff` avec le mot vide.

""	""	""	""	""	""

Après avoir visité `mots[0]` (càd "je"), le tableau `motsDiff` devient

"je"	""	""	""	""	""

Après avoir visité `mots[1]` (càd "tu"), le tableau `motsDiff` devient

"je"	"tu"	""	""	""	""

Après avoir visité `mots[2]` (càd la deuxième occurrence de "tu"), le tableau `motsDiff` reste inchangé car le mot "tu" apparait déjà dans la partie significative de `motsDiff`. Et ainsi de suite. Après avoir visité tout le tableau `mots`, le tableau `motsDiff` sera égal à

"je"	"tu"	"elle"	""	""	""

Ainsi, à la fin, seules les n premières cases de `motsDiff` contiendront des informations utiles, où n est le nombre de mots différents qui apparaissent dans `motsDiff`, les suivantes contiendront le mot vide.

Exercice 4 : Copie du sous-tableau [L]

a solution consistant à garder des mots vides dans notre tableau n'est pas tout à fait satisfaisante. Pour y remédier, nous allons créer une fonction `String[] sousTableau(String[] tab, int taille)` permettant de copier le sous-tableau des `taille` premières cases d'un tableau `tab` donné en paramètre. Si `taille` n'est pas une taille correcte, càd taille est un nombre négatif, ou taille est un nombre plus grand que `length(tab)`, alors la fonction va retourner une copie exacte de `tab`.

```
void testSousTableau () {
  assertArrayEquals(new String[]{"a", "b", "c"},
        sousTableau(new String[]{"a", "b", "c", "d", "e"}, 3));
  assertArrayEquals(new String[]{"a", "b"},
        sousTableau(new String[]{"a", "b"}, 5));
  assertArrayEquals(new String[]{},
        sousTableau(new String[]{"a", "b", "c"}, 0));
  assertArrayEquals(new String[]{"a", "b"},
        sousTableau(new String[]{"a", "b"}, -5));

  // L'assertion suivante sert à tester qu'on retourne bien une copie du tableau
  // et non le tableau lui-même
  String[] tab = new String[]{"a", "b"};
  assertNotEquals(tab, sousTableau(tab, 2));
  assertNotEquals(tab, sousTableau(tab, 3));
  assertNotEquals(tab, sousTableau(tab, -1));
}
```

On modifie désormais notre fonction `motsDifferents` de manière à ne conserver que les mots non-vides. Pour ce faire, il suffit de retourner le sous-tableau de `motsDiff`, en utilisant la fonction `sousTableau` écrite précédemment. La fonction devra donc désormais passer le test suivant.

```
  void testMotsDifferentsV2 () {
    String[] mots = new String[]{"je", "tu", "tu", "elle", "je", "je"};
    String[] mu = new String[]{"je", "tu", "elle"};
    assertArrayEquals(mu,motsDifferents(mots));
  }
```

Exercice 5 : Extraire dans un tableau les mots d'un texte [L]

e dernier ingrédient pour le calcul des fréquences est de stocker dans un tableau les mots contenus dans un texte (ici, une `String`). Dans la suite, une lettre signifie un caractère alphabétique. Un mot est une suite consécutive de lettres précédée et suivie d'un caractère qui n'est pas une lettre. Vous devez écrire la fonction `String[] lesMots(String texte)` qui passe ce test.

```
void testLesMots () {
  assertArrayEquals(new String[]{"je", "tu", "elle", "je", "tu"},
                    lesMots("je␣tu␣elle␣je␣tu"));
  assertArrayEquals(new String[]{"je", "tu", "elle", "je", "tu"},
                    lesMots("␣je␣tu,␣␣␣elle␣.␣je␣tu␣!␣"));
}
```

Voici quelques indications pour écrire cette fonction:

1. Comme pour la fonction `motsDifferents`, nous ne pouvons pas connaitre le taille du tableau à produire en résultat avant d'avoir visité le texte entier. Pour gérer ce problème on peut utiliser la même technique que dans `motsDifferents`.
2. Remarquer que pour détecter la fin d'un mot dans texte, il suffit de détecter quand on passe d'une lettre à un autre caractère et pour détecter le début d'un mot, quand on passe d'autre chose qu'une lettre à une lettre.
3. L'algorithme est plus facile à écrire si on suppose que la dernier caractère de `texte` n'est pas alphabétique. Par ailleurs, on peut toujours ajouter un espace à la fin de `texte` si cela nous facilite le travail.
4. Il peut être utile de disposer d'une fonction `boolean estLettre(char c)` qui teste si un caractère est une lettre.
5. Vous pouvez commencer par écrire une fonction qui affiche les mots un à un, et la modifier ensuite pour que les mots soient stockés dans un tableau.

Exercice 6 : Prêts pour le calcul des fréquences [N]

ous avons maintenant tous les ingrédients pour calculer les fréquences de chaque mot dans un texte. Compléter la fonction `void algorithm()` ci-dessous pour qu'elle affiche la liste des mots de `texte` et la fréquence de chaque mot.

```
void algorithm() {
  String texte =
  "Patience, patience,\n" +
  "Patience dans l'azur !\n" +
  "Chaque atome de silence\n" +
  "Est la chance d'un fruit mur !";
  println(texte);
  println();
  String[] mots = lesMots(texte);
  // Compléter ici en utilisant les fonctions écrites précédemment pour afficher les mots
du texte
}
```

Exercice 7 : (Prolongement) Extraction de mots améliorée [L]

a fonction de l'exercice précédent ne peut pas détecter qu'il s'agit du même mot lorsque celui-ci est écrit une fois avec une majuscule, et l'autre fois avec une minuscule. Pour améliorer cette situation, modifiez votre fonction pour qu'elle transforme chaque mot en sa version en minuscules seulement avant de l'ajouter au résultat. Il peut être utile d'écrire une fonction auxiliaire qui permet de donner la version minuscules seulement d'un mot.

Votre fonction modifiée doit aussi passer ce test.

```
void testLesMotsV2 () {
  assertArrayEquals(new String[]{"je", "tu", "elle", "je", "tu"},
                    lesMots(" jE tu,   Elle je tu  "));
}
```

Exercice 8 : (Prolongement) Trier les mots d'après leur fréquence [a]

ns l'algorithme de l'exercice précédent, les tableaux `motsDifferents` et `freq` sont liés dans le sens où pour chaque indice `i`, `freq[i]` est le nombre d'occurrences dans `texte` du mot `motsDifferents[i]`. Ce serait bien de pouvoir présenter les mots les plus fréquents en premier. Pour cela, il faudrait trier le tableau `motsDifferents` en fonction des valeurs contenues dans le tableau `freq`.

Écrire la fonction `void triSimultané(int[] critereTri, String[] tab)` qui trie le tableau `tab` en fonction des valeurs données dans le tableau `critereTri`. Concrètement, cette fonction va effectuer un tri de `critereTri`, mais à chaque fois que les valeurs de deux cases sont échangées dans `critereTri`, le même changement doit être fait dans `tab`. Vous ferez un tri dans l'ordre décroissant.

```
void testTriSimultane () {
  int[] critere = new int[]{1,5,3,2,4};
  String[] tab = new String[]{"un", "cinq", "trois", "deux", "quatre"};
  triSimultane(critere, tab);
  assertArrayEquals(new int[]{5,4,3,2,1}, critere);
  assertArrayEquals(new String[]{"cinq","quatre","trois","deux","un"}, tab);
}
```

Ajoutez maintenant un appel à `triSimultane` au bon endroit dans la fonction `void algorithm()` pour que les mots les plus fréquents soient affichés en premier.