Cours MOOC EPFL d'initiation à la programmation illustré en Java

Quatrième devoir noté
Tableaux et chaînes de caractères

J. Sam & J.-C. Chappelier

1 Exercice 1 — Elément le plus fréquent dans un tableau

Le but de cet exercice est d'écrire un programme permettant d'identifier l'élément apparaîssant le plus fréquemment dans un tableau d'entiers.

Ce programme devra également afficher le nombre d'occurences dans le tableau de cet élément le plus fréquent. Par exemple, pour le tableau suivant :

```
{2, 7, 5, 6, 7, 1, 6, 2, 1, 7}
```

votre programme devra indiquer que l'élément le plus fréquent est le 7 et que sa fréquence d'apparition est 3.

Télécharger le programme `MostFrequent.java` fourni sur le site du courset le compléter suivant les instructions données ci-dessous.

ATTENTION : vous ne devez modifier ni le début ni la fin du programme, juste ajouter vos propres lignes à l'endroit indiqué. Il est donc primordial de respecter la procédure suivante (les points 1 et 3 concernent spécifiquement les utilisateurs d'Eclipse) :

1. désactiver le formatage automatique dans Eclipse :
   `Window > Preferences > Java > Editor > Save Actions`
   (et décocher l'option de reformatage si elle est cochée)
2. sauvegarder le fichier téléchargé sous le nom `MostFrequent.java` (avec une majuscule, notamment). Si vous travaillez avec Eclipse vous ferez cette sauvegarde à l'emplacement `dossierDuProjetPourCetExercice/src/` ;

3. rafraîchir le projet Eclipse où est stocké le fichier (clic droit sur le projet > refresh) pour qu'il le prenne en compte ;
4. écrire le code à fournir entre ces deux commentaires :

```
/*****************************************
 * Completez le programme a partir d'ici.
 *****************************************/

/*****************************************
 * Ne rien modifier apres cette ligne.
 *****************************************/
```

5. sauvegarder et tester son programme pour être sûr(e) qu'il fonctionne correctement, par exemple avec les valeurs données plus bas ;
6. rendre le fichier modifié (toujours `MostFrequent.java`) dans « OUTPUT submission » (et non pas dans « Additional ! »).

Exemple d'exécution

Votre programe devra produire un affichage se conformant strictement à l'exemple suivant :

```
Le nombre le plus frequent dans le tableau est le :
7 (3 x)
```

Il y a un retour à la ligne après les « : ».

Le code que vous écrirez devra pouvoir s'appliquer à n'importe quel tableau, mais vous pourrez supposer que ces tableaux sont toujours non vides.

Notez à ce propos que si dans un tableau donné il y a plus d'un nombre ayant le plus grand nombre d'occurences, alors votre programme ne retiendra que celui qui apparaît en premier dans le tableau.

Par exemple, pour le tableau `tab1 = {2, 7, 5, 6, 7, 1, 6, 2, 1, 7, 6}`, où 6 et 7 sont tous deux les nombres les plus fréquents (les deux apparaissant 3 fois), votre programme ne retiendra que le 7 et affichera :

```
Le nombre le plus frequent dans le tableau est le :
7 (3 x)
```

2 Exercice 2 — Cryptage

Jules César utilisait un système de codage très simple, qui consiste à remplacer chaque lettre d'un message par la lettre placée plusieurs rangs après dans l'ordre alphabétique. Par exemple, pour un décalage de 4, A devient E, B devient F, jusque Z qui devient D.

Il s'agit ici d'appliquer cette technique pour coder une chaîne de caractère. Vous écrirez pour cela un programme qui met en oeuvre les traitements décrits ci-dessous.

2.1 Codage

Le programme fourni pour cet exercice met à votre disposition une chaîne de caractère `ALPHABET` contenant toutes les lettres de l'alphabet latin, en minuscule.

Il demande à l'utilisateur de saisir une chaîne de caractères `s`.

Le but de l'exercice est de construire une nouvelle chaîne `aCoder` comme suit :

- `aCoder` contient, dans l'ordre, tous les caractères de `s` qui sont présents dans `ALPHABET` ou qui correspondent au caractères espace (`' '`). **Tous les autres caractères sont ignorés**.

La chaîne `aCoder` sera codée au moyen de la technique de Jules César. **Elle devra être affichée**. Si elle ne contient aucun caractère, un message indiquant que la chaîne à coder est vide sera affiché.

Lorsque la chaîne à coder est non vide, votre programme devra la coder avec un décalage fixe donné par la constante fournie `DECALAGE`.

Codez la chaîne `aCoder` en remplaçant chaque caractère par celui situé `DECALAGE` lettres plus loin dans la chaîne de caractères `ALPHABET`, et cela de façon cyclique ('z' sera remplacé par 'd' par exemple). Les espaces seront maintenus tels quels. Il ne feront l'objet d'aucun codage mais resteront présents à l'endroit qu'ils occupaient.

La chaîne ainsi codée sera affichée.

L'exécution de votre programme devra strictement se conformer aux exemples suivants :

```
Veuillez entrer une chaine de caracteres :
fuyez manants
La chaine initiale etait : 'fuyez manants'
La chaine a coder est : 'fuyez manants'
```

```
La chaine codee est : 'jycid qererxw'

Veuillez entrer une chaine de caracteres :
avez-vous vu mes 3 chats et mes 2 chiens ?
La chaine initiale etait : 'avez-vous vu mes 3 chats et mes 2 chiens
La chaine a coder est : 'avezvous vu mes  chats et mes  chiens '
La chaine codee est : 'ezidzsyw zy qiw  glexw ix qiw  glmirw '

Veuillez entrer une chaine de caracteres :
93589()çç%&=+12AD
La chaine initiale etait : '93589()çç%&=+12AD'
La chaine a coder est vide.
```

2.2 Marche à suivre

Télécharger le programme `Crypto.java` fourni sur le site du courset le compléter suivant les instructions données ci-dessous.

ATTENTION : vous ne devez modifier ni le début ni la fin du programme, juste ajouter vos propres lignes à l'endroit indiqué. Il est donc primordial de respecter la procédure suivante (les points 1 et 3 concernent spécifiquement les utilisateurs d'Eclipse) :

1. désactiver le formatage automatique dans Eclipse :
   `Window > Preferences > Java > Editor > Save Actions`
   (et décocher l'option de reformatage si elle est cochée)
2. sauvegarder le fichier téléchargé sous le nom `Crypto.java` (avec une majuscule, notamment). Si vous travaillez avec Eclipse vous ferez cette sauvegarde à l'emplacement `dossierDuProjetPourCetExercice/src/` ;
3. rafraîchir le projet Eclipse où est stocké le fichier (clic droit sur le projet > refresh) pour qu'il le prenne en compte ;
4. écrire le code à fournir entre ces deux commentaires :

```
/*****************************************
 * Completez le programme a partir d'ici.
 *****************************************/

/*****************************************
 * Ne rien modifier apres cette ligne.
 *****************************************/
```

5. sauvegarder et tester son programme pour être sûr(e) qu'il fonctionne correctement, par exemple avec les valeurs données plus bas ;
6. rendre le fichier modifié (toujours `Crypto.java`) dans « OUTPUT submission » (et non pas dans « Additional ! »).

3 Exercice 3 — « Sens de la propriété »

3.1 Introduction

Un riche propriétaire souhaite clôturer ses terrains. Vous devez écrire un programme pour l'aider à calculer le nombre de mètres de clôture nécessaires.

3.2 Description

Télécharger le programme `Cloture.java` fourni sur le site du courset le compléter suivant les instructions données ci-dessous.

ATTENTION : vous ne devez modifier ni le début ni la fin du programme, juste ajouter vos propres lignes à l'endroit indiqué. Il est donc primordial de respecter la procédure suivante (les points 1 et 3 concernent spécifiquement les utilisateurs d'Eclipse) :

1. désactiver le formatage automatique dans Eclipse :
   `Window > Preferences > Java > Editor > Save Actions`
   (et décocher l'option de reformatage si elle est cochée)
2. sauvegarder le fichier téléchargé sous le nom `Cloture.java` (avec une majuscule, notamment). Si vous travaillez avec Eclipse vous ferez cette sauvegarde à l'emplacement `dossierDuProjetPourCetExercice/src/` ;
3. rafraîchir le projet Eclipse où est stocké le fichier (clic droit sur le projet > refresh) pour qu'il le prenne en compte ;
4. écrire le code à fournir entre ces deux commentaires :

```
/*****************************************
 * Completez le programme a partir d'ici.
 *****************************************/

/*****************************************
 * Ne rien modifier apres cette ligne.
 *****************************************/
```

5. sauvegarder et tester son programme pour être sûr(e) qu'il fonctionne correctement, par exemple avec les valeurs données plus bas ;
6. rendre le fichier modifié (toujours `Cloture.java`) dans « OUTPUT submission » (et non pas dans « Additional ! »).

3.3 Représentation des terrains

Le propriétaire dispose d'une représentation de ses terrains sous forme digitalisée : un terrain y est représenté par le biais d'un tableau binaire à deux dimensions. Les `1` y représentent des plaques carrées faisant partie du terrain et les `0` celles n'en faisant pas partie.

Il s'agit donc dans cet exercice de calculer le nombre de mètres de clôture nécessaire pour entourer un terrain donné selon ce format. Pour cela, il faudra compter le nombre de `1` constituant le pourtour du terrain. Chaque `1` du pourtour correspond à 2.50 mètres de clôture *pour chacun de ses cotés qui sont au bord* du terrain.

Par exemple, le terrain

```
1
```

(tout seul) qui représente un tout petit terrain carré de 2.5 m de coté, nécessitera 10 (= quatre fois 2.5) mètres de clôture.

Le terrain

```
0110
0110
```

aura besoin de 20 m de clôture, et le terrain

```
0111110
0111110
0111110
```

de 40 m.

Le problème est un peu compliqué par le fait que le terrain peut contenir des étangs. L'intérieur des étangs est aussi représenté par des `0`, mais le pourtour d'un étang ne doit pas être comptabilisé dans le périmètre du terrain.

Exemple de terrain digitalisé avec des étangs à l'intérieur :

```
000000000000000000000000000000111111100000
000000000000000000000000000011111111111110
000000000000111111111111111111111111111111
001111111111111111111111111111111111111111
011111111111111111111111111111111111111111
111111111111111111111111111111111111111111
111111111111111111111111111111111111111111
111111111111111111111111111111111111111111
011111111111111001111111111111111111111111
011111111111111001111111111111000111111100
011111111111111001111111111110000111111100
001111111111111001111111111100001111111100
000111111111110001111111111100011111110000
000111111111000000111111111111111111110000
000111111110000000000111111111111111110000
000001111100000000000011111111111110000000
000000111111000000000111111111111100000000
000001111111100000001111111111111110000000
000111111111110000111111111111111111000000
000111111111111101111111111111111100000000
000011111111111110111111111111111000000000
000001111111111111011111111111100000000000
000001111111111111111111111111000000000000
000000111111111111111111111110000000000000
000000111111111111111111111100000000000000
000000111111111111111111111100000000000000
000000111111000000000000000000000000000000
000000110000000000000000000000000000000000
```

qui correspondrait au terrain suivant :

Pour simplifier, on supposera :

— que le terrain est « en un seul morceau » : il n'existe pas de zones du terrain déconnectées les unes des autres ;

— qu'il n'y a pas de ligne ne contenant que des `0` ;
— que le pourtour extérieur du terrain est « convexe par ligne », c'est-à-dire que pour chaque ligne de l'image du terrain [1], les seuls `1` du pourtour extérieur sont le premier et le dernier présents sur la ligne [2] ; on ne peut pas avoir une ligne comme cela : « `0011110001111` » où les `0` du milieu seraient des `0` extérieurs ; ce sont dans ce cas forcément des `0` d'un étang.

Tout cela pour garantir qu'un `0` est donc à l'intérieur du terrain (étang) si, sur sa ligne [3], il y a au moins un `1` qui le précède et au moins un `1` qui le suit.

3.4 Le code à produire

Le code que vous écrirez commencera par vérifier que le tableau fourni (`carte`) contient bien uniquement des `0` et des `1`. Si tel n'est pas le cas, un message d'erreur respectant *strictement* le format suivant sera affiché (avec les sauts de ligne) :

```
Votre carte du terrain n'a pas le bon format :
valeur '2' trouvée en position [8][7]
```

Le programme arrêtera dans ce cas son exécution. Notez que le tableau que nous donnons dans le code fourni sera remplacé par d'autres tableaux pour tester votre code.

Si le tableau `carte` représentant le terrain est correct, votre programme affichera alors le nombre de mètres de clôture nécessaires en respectant *strictement* le format d'affichage donné dans les exemples d'exécution donnés plus bas (y compris le saut de ligne en fin de message).

Indications :

— Une façon simple de procéder consiste à d'abord « effacer » les étangs (en remplaçant les `0` des étangs par des `1`), puis procéder ensuite au comptage des `1` situés sur le pourtour.
— Un point du pourtour peut avoir plusieurs `0` comme voisins (par exemple un `0` au dessus et un `0` à gauche). Il doit dans ce cas être comptabilisé autant de fois dans la clôture (deux fois dans l'exemple).

1. mais on ne fait pas cette hypothèse pour les colonnes ! Voir l'exemple ci-dessus.
2. mais il peut n'y en avoir qu'un seul lorsque le premier et le dernier sont le même : « `000010000` ».
3. mais pas forcément sur sa colonne ! Voir l'exemple ci-dessus.

— Pour forcer la sortie de la méthode `main`, si vous estimez que l'exécution doit s'arrêter à un certain point, vous pouvez utiliser l'instruction `return;`.

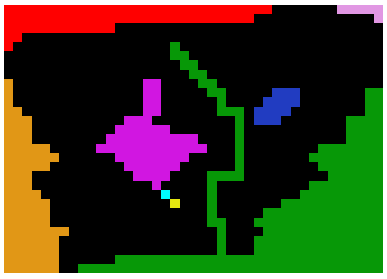
3.5 Vérification de la convexité de ligne

Pour finir, nous vous demandons d'ajouter, entre la vérification que la carte contient bien uniquement des `0` et des `1` et le calcul de la longueur de la clôture, une vérification supplémentaire que la carte soit bien « convexe par ligne ».

Noter que cette partie, plus difficile, est totalement indépendante du reste et que vous pouvez tout à fait calculer la longueur de la clôture, et avoir quelques points sur l'exercice, sans avoir fait cette dernière partie.

La vérification demandée devra donc rejeter toute carte dans laquelle des `0` de l'extérieur du terrain sont présents entre deux `1` d'une même ligne, comme par exemple dans cette carte :

L'algorithme que nous vous proposons pour détecter ces cartes erronées est le suivant :

1. trouver toutes les zones de `0` (on parle de « composantes connexes ») ;
   par exemple, les différentes zones de `0` de l'image précédente sont ici représentées de différentes couleurs :
   
2. trouver parmi ces zones, celles qui sont à l'extérieur du terrain (les autres étant des étangs) ;

3. parcourir l'image ligne à ligne pour voir si une zone de `0` extérieurs est comprise entre deux `1`.

   A noter que vous pourrez regrouper cette dernière étape avec votre étape « d'effacement » des étangs (décrite dans la section précédente).

Pour la première étape (trouver les composantes connexes), avec les connaissances de programmation à ce stade du cours, nous vous proposons de procéder comme suit :

1. déclarez deux tableaux dynamiques d'entiers ; ils serviront à stocker les coordonnées des points de la carte en cours de traitement, l'un pour les ordonnées `i`, l'autre pour les abscisses `j` ;
2. déclarez une variable de type entier et initialisez là à la valeur 1 ; cette variable nous servira à compter et à étiqueter les différentes zone de `0` ; pour simplifier, appelons-la « `composante` » ; elle sera augmentée de 1 à chaque nouvelle zone ;
3. bouclez sur toutes les positions de la carte ;
   si la valeur à la position courante est `0` :
   — augmentez de 1 la variable de compte des zones (`composante`) ;
   — ajoutez l'ordonnée `i` et l'abscisse `j` de la positions courante à vos tableaux dynamiques ;
   — tant que ces tableaux dynamiques ne sont pas vides :
   — récupérez et supprimez les valeurs du premier élément de chaque tableau (une abscisse et une ordonnée) ;
   — si la valeur de la carte à cette position nouvellement récupérée est 0 :
   — mettre la valeur de zone (`composante`) à cette position de la carte ;
   — pour chacun des voisins de la position récupère (voisins NORD, SUD, EST et OUEST, s'ils existent) : si la valeur du voisin est `0`, ajoutez ses coordonnées (abscisse et ordonnée) à vos tableaux dynamiques.

Si vous affichez la carte précédente suite à cette étape, vous devriez obtenir :

```
22222222222222222222222222211111133333
22222222222222222222222222111111111113
22222222222111111111111111111111111111
22111111111111111111111111111111111111
21111111111111111141111111111111111111
11111111111111111441111111111111111111
11111111111111111441111111111111111111
```

```
11111111111111111111441111111111111111111
51111111111111166111144111111111111111111
51111111111111166111114411111777111111144
51111111111111166111111411117777111111144
55111111111111166111111444177771111111144
55511111111116661111111114177711111114444
55511111111166666611111141111111111114444
55511111111666666666611114111111111114444
55555111116666666666661114111111114444444
55555511111166666666111114111111144444444
55555111111116666666111114111111114444444
55511111111111666611114444111111111444444
55511111111111116111141111111111144444444
55551111111111111811114111111111444444444
55555111111111111191114111111144444444444
55555111111111111111114111114444444444444
55555111111111111111144111444444444444444
55555551111111111111114111144444444444444
55555511111111111111114111444444444444444
55555511111111111111114111444444444444444
55555511111444444444444444444444444444444
55555511444444444444444444444444444444444
```

Pour la seconde étape, il suffit simplement de faire le tour du bord de la carte et stocker dans un tableau dynamique toutes les valeurs rencontrées qui ne sont pas `1`. Même si ce n'est pas optimal, ce n'est pas grave ici de répéter plusieurs fois la même valeur dans ce tableau. [4]

Pour la troisième étape enfin, il suffit de parcourir chaque ligne et rechercher toute valeur non `1` comprise entre le premier et le dernier `1` de cette ligne. Si cette valeur est présente dans le tableau construit à l'étape précédente (tableaux des composantes du bord), alors c'est que la carte est erronée. Il faudra alors afficher un message d'erreur respectant *strictement* le format suivant (avec les sauts de ligne) :

```
Votre carte du terrain n'a pas le bon format :
bord extérieur entrant trouvé en position [4][18]
```

4. Avec des notions supplémentaires présentées plus tard dans le cours, on pourrait de façon efficace n'ajouter qu'une seule fois chaque valeur. Mais ce n'est pas fondamental.

3.6 Exemples de déroulement

Avec la carte donnée dans le code fourni et illustrée plus haut :

```
Il vous faut 385.0 mètres de clôture pour votre terrain.
```

Avec une carte contenant un `2` en position `i=8`, `j=7` :

```
Votre carte du terrain n'a pas le bon format :
valeur '2' trouvée en position [8][7]
```

Avec cette carte :

```
01110
01010
01110
```

vous devriez avoir :

```
Il vous faut 30.0 mètres de clôture pour votre terrain.
```

Et avec celle-ci :

```
111
001
111
```

vous devriez avoir :

```
Il vous faut 40.0 mètres de clôture pour votre terrain.
```

Avec une carte n'étant pas « convexe par ligne » comme par exemple celle illustrée plus haut :

```
Votre carte du terrain n'a pas le bon format :
bord extérieur entrant trouvé en position [4][18]
```