

# TP 8 : tableaux à deux dimensions

Informatique Fondamentale (IF1)

Semaine du 22 novembre 2010

## Préliminaires

**Exercice 1.** Écrivez une fonction

```
public static int[] [] saisie()
```

qui lit deux entiers  $n$  et  $m$ , puis  $n \times m$  entiers, puis retourne un tableau de  $n$  lignes et  $m$  colonnes contenant les entiers saisis.

En quoi cette fonction diffère-t-elle de la fonction `Saisie` vue en cours ?

**Exercice 2.** Écrivez une fonction

```
public static void affiche(int[] [] a)
```

qui affiche le tableau passé en paramètre.

Écrivez une fonction `main` qui saisit un tableau et l'affiche ; comme d'habitude, testez votre programme.

## 1 Recherche de motifs

**Exercice 3.** Écrivez une fonction

```
public static boolean recherche2(int[] [] a, int v)
```

qui retourne `true` s'il existe une paire d'indices  $i, j$  tels que  $a[i][j] = v$ .

Écrivez une fonction `main` qui lit au clavier un tableau ainsi qu'une valeur, et affiche si la valeur se trouve dans le tableau. (Vous pouvez bien-sûr vous servir des fonctions écrites dans la partie précédente.)

**Exercice 4.** Écrivez une fonction

```
public static boolean recherche3(int[] [] a, int[] [] s)
```

qui retourne `true` s'il existe un sous-tableau du tableau `a` qui est égal à `s`. En d'autres termes, s'il existe des indices  $i, j$  tels que pour tout  $k < s.length$  et  $l < s[0].length$ ,

$$a[i + k][j + l] = s[k][l].$$

Écrivez une fonction `main` qui vous permette de tester votre fonction.

## 2 Carrés magiques

Un *carré magique* est une matrice carrée de taille  $n \times n$  telle que la somme de chaque rangée, de chaque colonne et de chaque diagonale ait la même valeur. Un carré magique est dit *normal* s'il contient chaque entier compris entre 1 et  $n^2$  exactement une fois. Par exemple, le tableau suivant est un carré magique normal :

$$\begin{bmatrix} 6 & 7 & 2 \\ 1 & 5 & 9 \\ 8 & 3 & 4 \end{bmatrix}$$

**Exercice 5.** Écrivez une fonction

```
public static boolean carre(int[] [] a)
```

qui retourne `true` si le tableau `a` est une matrice carrée (qui a autant de lignes que de colonnes). Écrivez une fonction `main` et testez votre programme.

**Exercice 6.** Écrivez deux fonctions

```
public static int ligne(int[] [] a, int i)
public static int colonne(int[] [] a, int j)
```

qui retournent la somme de la  $i$ -ème ligne, resp. de la  $j$ -ème colonne, du tableau passé en paramètre. Écrivez une fonction `main` qui vous permette de tester ces fonctions.

**Exercice 7.** Écrivez deux fonctions

```
public static int diagonale1(int[] [] a)
public static int diagonale2(int[] [] a)
```

qui retournent la somme de la diagonale majeure, resp. de la diagonale mineure, du tableau passé en paramètre. Écrivez une fonction `main` qui vous permette de tester ces fonctions.

**Exercice 8.** Écrivez un programme qui demande à l'utilisateur de rentrer un tableau, et affiche s'il s'agit d'un carré magique (pas forcément normal).

**Exercice 9.** Écrivez une fonction

```
public static int[] histogramme(int[] [] a, int n)
```

qui retourne l'histogramme de taille `n` du tableau `a`, c'est-à-dire le tableau `h` de taille `n` tel que pour tout  $i < n$ , `h[i]` contient le nombre d'occurrences de la valeur  $i$  dans le tableau `a`.

**Exercice 10.** Modifiez le programme écrit ci-dessus pour qu'il affiche en outre si le carré magique est normal. (*Indication : on pourra par exemple construire l'histogramme de taille  $n^2 + 1$  du carré magique.*)

### 3 Images noir et blanc

On rappelle qu'une image noir et blanc est représentée en mémoire par un tableau de booléens. L'affichage d'une telle image peut se faire à l'aide de la fonction suivante :

```
public static void dessine(boolean[] [] image) {
    for(int i = 0; i < image.length; i++)
        for(int j = 0; j < image[i].length; i++) {
            if(image[i][j])
                Deug.drawPoint(i, j);
        }
}
```

**Exercice 11.** Écrivez une fonction

```
public static boolean[] [] croix(int l, int h)
```

qui retourne une image de taille  $l \times h$  contenant les segments qui connectent les milieux des bords opposés.

Écrivez un programme qui affiche le contenu de cette image.

**Exercice 12.** Écrivez une fonction

```
public static boolean[] [] croixDiagonale(int l)
```

qui retourne une image de taille  $l \times l$  contenant les segments connectant les angles opposés.

**Exercice 13.** Écrivez une fonction

```
public static boolean[] [] superpose(boolean[] [] im1, boolean[] [] im2)
```

qui retourne une nouvelle image qui contient la superposition des images `im1` et `im2`. Écrivez un programme qui affiche la superposition des deux croix dessinées aux exercices précédents.

### 4 Images en niveaux de gris

On rappelle qu'une image en niveaux de gris est représentée par une matrice d'entiers courts :

```
short[] []
```

contenant des valeurs comprises entre 0, représentant la couleur noire, et 255, représentant la couleur blanche.

**Exercice 14.** Écrivez une fonction

```
public static void dessineGris(short[] [])
```

qui dessine une image en niveaux gris.

En supposant que l'image contient des zones entières de couleur identique, comment peut-on minimiser le nombre d'appels à `setGray` ?

**Exercice 15.** Écrivez une fonction

```
public static short[][] colourExpand(boolean[][] im)
```

qui convertit une image noir et blanc en une image en niveaux de gris identique. Testez votre fonction.

**Exercice 16.** Écrivez une fonction

```
public static short[][] antiAlias(short[][] im)
```

qui implémente un *filtre d'anti-aliasage*. Elle retourne une nouvelle image de la même taille que l'image `im` où les ■ créneaux ■ ont été adoucis en utilisant les niveaux de gris. L'intensité d'un pixel de la nouvelle image sera égale à la somme de  $3/4$  fois la valeur du pixel correspondant de l'ancienne image et  $1/4$  de la moyenne des valeurs des 8 pixels voisins.

Écrivez un programme qui affiche la version filtrée de la superposition des deux croix obtenues ci-dessus.