

# Séance 11b: MANIPULATION DES IMAGES

Université de Paris

## Objectifs:

- Implémenter et utiliser des fonctions étant donnée une spécification
- Manipuler des tableaux de tableaux

### Exercice 1 (Images en gris, ★★)

Dans cet exercice on travaillera avec des images en gris. Dans ce format, une image est représentée par un tableau de tableaux d'entiers, chaque entier représentant un pixel de l'image. Les valeurs des entiers sont comprises entre 0 et 255, qui correspondent aux valeurs d'intensité du gris. La valeur 255 indique donc le blanc, et la valeur 0 le noir. L'entrée  $[i][j]$  représentera l'intensité du pixel qui se trouve  $i$  pixels à droite et  $j$  pixels en bas du pixel supérieur gauche  $[0][0]$  (regarder la figure ci-dessous).

Pour lire une image (jpg, png ou jpeg) utilisez fonction public static `int[][] getGray(String filename)` déjà écrite dans le fichier, qui renvoie un tableau des tableaux d'entiers correspondant à l'image.

Pour afficher l'image, utilisez la fonction `drawGray`, en donnant le tableau de tableaux, et pour la sauve en format png utilisez `saveGray`.

1. Écrire une fonction `negatif` qui, étant donnée un tableau de tableaux d'entiers, représentant une image en gris, renvoie son négatif. Le négatif s'obtient en inversant l'échelle intensité  $\mapsto 255 - \text{intensité}$ . Testez votre fonction sur l'image `image.jpg`

#### Contrat:

Par exemple, `negatif({{0, 128, 255}, {128, 255, 255}, {255, 255, 255}})` renvoie : `{{255, 127, 0}, {127, 0, 0}, {0, 0, 0}}`

2. Écrire une fonction `flip` qui, étant donnée un tableau de tableaux d'entiers, représentant une image en gris, inverse la direction de la coordonnée horizontale.

#### Contrat:

Par exemple, `flip({{0, 128, 255}, {128, 255, 255}, {255, 255, 255}})` renvoie : `{{255, 255, 255}, {128, 255, 255}, {0, 128, 255}}`

3. Une opération classique sur des images correspond à détecter les bords. On va faire ça d'une façon simplifiée comme suit : pour chaque pixel  $(x, y)$  avec  $x, y \geq 1$  on calcule

$$\Delta = |\text{gray}(x-1, y) - \text{gray}(x, y)| + |\text{gray}(x, y-1) - \text{gray}(x, y)|,$$

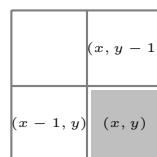


FIGURE 1 – Pixels qui interviennent dans le calcul de  $\Delta$ .

et si  $\Delta > 30$  on déclare  $(x, y)$  comme un "pixel de bord".

Écrire une fonction renvoyant le tableaux de tableaux correspondant, où on met 0 pour un pixel de bord, et 255 autrement. Tester ! Que se passe-t-il si on met 5, 10, 50 au lieu de 30 ?

- On voudrait générer "ASCII art" automatiquement d'après une image en gris. Pour le faire, on va découper les intensités en 10 niveaux, les intensités 0..25 correspondront au caractère '@', les intensités 26..51 à '%', et ainsi suite avec les caractères :

"@", "%", "#", "\*", "+", "=", "-", ":", ";", ".", " "

Par exemple

Écrire une fonction `drawASCII` qui, étant donnée une image dans l'entrée, affiche le dessin correspondant dans le terminal. Utilisez l'image `icon.gif`, et agrandissez le terminal.

- Pour égaliser une image on va remplacer l'intensité chaque pixel  $i = \text{gris}(x, y)$  de chaque pixel  $(x, y)$  par  $\frac{255 \times s_i}{\text{width} \times \text{height}}$ , où  $s_i$  correspond à la quantité de pixels ayant une intensité inférieur ou égal à  $i$ . Testez sur l'image `overexposed.jpg`

**Contrat:**

Par exemple, étant donné `int [][] l = {{0,0},{255,255}};`,

l'appel `egalise(l)` renvoie `{{127,127},{255,255}}`

- On crée maintenant une opération de "mélange". On découpe l'image en blocs de  $2 \times 2$  avec les pixels  $(2x, 2y)$ ,  $(2x + 1, y)$ ,  $(2x, 2y + 1)$ ,  $(2x + 1, 2y + 1)$ , et on applique une opération de permutation de ses intensités, donnée par

$$\text{gray}(2x, 2y) \rightarrow \text{gray}(2x + 1, 2y) \rightarrow \text{gray}(2x + 1, 2y + 1) \rightarrow \text{gray}(2x, 2y + 1) \rightarrow \text{gray}(2x, 2y),$$

pour chaque carré de  $2 \times 2$ , et qui fait rien sur la dernière colonne si `width` est impair, et qui fait rien sur la dernière ligne si `height` est impair.

**Contrat:**

Par exemple, étant donné

```
int [][] exemple = {
    {0, 128, 255},
    {127, 255, 255},
    {255, 255, 255}};
```

l'appel `melange(exemple)` renvoie :

```
{127, 255, 255},
{0, 128, 255},
{255, 255, 255}}
```

7. Pour finir, on crée une fonction de compression d'image. La fonction `compress` prend en argument une liste de liste d'entiers `l` de taille  $n \times m$ , et renvoie une liste de liste `result` de taille  $n/2 \times m/2$ , où `result[i][j]` est la moyenne des cases `l[2i][2j]`, `l[2i+1][2j]`, `l[2i][2j+1]` et `l[2i+1][2j+1]`

**Contrat:**

Par exemple, étant donné

```
int [][] exemple = {
    {0 , 100, 200, 255},
    {0 , 100, 200, 255},
    {0 , 100, 200, 255},
    {0 , 100, 200, 255}};
```

l'appel `compress(exemple)` renvoie :

```
{{50, 127},
 {50, 127}}
```

□