Séance 12b: Manipulation des images

Université Paris-Diderot

Objectifs:

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

Exercice 1 (Images en gris, ⋆–⋆⋆)
Dans cet exercice on travaillera avec des images en gris. Dans ce format, une image est représentée par une liste de listes 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.

Pour lire une image (`pgm`*) utilisez fonction* `getGray` *déjà écrite dans le fichier, qui renvoie une liste de listes d'entiers correspondant à l'image en gris.*

Pour sauver une image, utilisez la fonction `saveGray`*, en donnant la liste de listes, et le chemin+nom du fichier de sortie (par exemple "./save.png" pour sauver dans le dossier de travail).*

1. *Écrire une fonction* `negatif` *qui, étant donnée une liste de listes d'entiers, representant une image en gris, renvoie son négatif. Le négatif s'obtient en inversant l'échelle* $intensite \mapsto 255 - intensite$*. Testez votre fonction sur l'image* `image.pgm`

   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 une liste de listes d'entiers, representant une image en gris, inverse la direction de la coordonée horizontal.*

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

3. *Une operation 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*

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

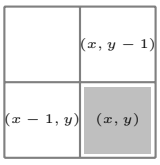


Figure 1 – Pixels qui interviennent dans le calcul de Δ.

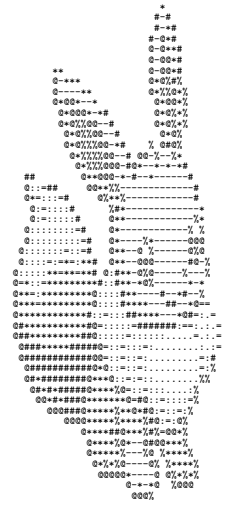
et si $\Delta > 30$ on déclare (x, y) comme un "pixel de bord".

Écrire une fonction renvoyant la liste de listes 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 ?

4. *On voudrait génerer "ASCII art" automatiquement d'après une image en gris. Pour le faire, on va decouper les instensités en 10 niveaux, les intensités 0..25 corresponderont 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.pgm`*, et agrandissez le terminal.*

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

   Contrat:

   Par exemple, étant donné `l = [[0,0],[255,255]]`,

   l'appel `egalise(l)` *renvoie* `[[127,127],[255,255]]`

6. *On crée maintenant une opération de "mélange". On decoupe l'image en blocs de 2×2 avec les pixels $(2x, 2y), (2x + 1, y), (2x, 2y + 1), (2x + 1, 2y + 1)$, et on aplique une operation de permutation de ses intensités, donnée par*

$$gray(2x, 2y) \to gray(2x + 1, 2y) \to gray(2x + 1, 2y + 1) \to gray(2x, 2y + 1) \to gray(2x, 2y)\,,$$

   pour chaque carré de 2×2, 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

```
exemple = [   [0   , 128, 255],
              [127, 255, 255],
              [255, 255, 255]];
```

   l'appel `melange(exemple)` *renvoie :*

```
[[127, 0   , 255],
 [255, 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

```
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]]
```

□