Images PGM

T.P. IV

Les fichiers *portable graymap file format* (ou pgm) sont des fichiers textes permettant de stocker des images. Ces images sont stockées décrites par la couleur, en niveau de gris, de chacun des pixels. Le codage du fichier est en ASCII et le format de la forme suivante :

```
P2
nb_lignes nb_colonnes
255
...
...
```

`P2` décrit le type de fichier, `nb_lignes` × `nb_colonnes` est la taille de l'image, `255` décrit l'ensemble des couleurs disponibles en niveaux de gris (0 blanc à 255 noir) et la suite décrit la couleur de chacun des pixels. L'image est codée ligne par ligne en partant du haut ; chaque ligne est alors codée de gauche à droite. L'image est ainsi parcourue en zigags.
Dans ce T.P., nous allons traiter de telles images à l'aide du module qui permet de définir un type *tableau*. Le module permet quant à lui d'obtenir des sorties graphiques. Dans les premières parties, un pré-traitement a été effectué pour passer de fichiers au format `pgm` en tableaux .
Les images sont chiffrées par votre correspondant à l'aide d'un algorithme qu'il vous a transmis. Nous considérerons ci-dessous deux algorithmes différents. L'objectif de ce T.P. est d'écrire les fonctions permettant de déchiffrer ces images. Les fichiers à déchiffrer, `03_mystere1.txt` et `03_mystere2.txt`, sont à copier dans votre dossier de travail.

Partie I : Chiffrement d'une photo par tri

Manipulation des tableaux. Le module `Numpy` va permettre de transformer les images en matrices (type `ndarray`). Chacun des éléments de cette matrice représente la couleur d'un pixel.

- `import numpy as np` charge le module `numpy` en utilisant le surnom `np`.
- `np.loadtxt("fichier.txt")` charge le contenu du fichier `fichier.txt` sous forme de tableau `Numpy`.
- L'accès à l'élément d'indice `(i,j)` d'un tableau `a` à deux entrées de type `ndarray` s'effectue via `a[i,j]`. L'accès à un sous-tableau s'effectue via `a[i0:i1:pas1,j0:j1:pas2]` en utilisant les règles de tranchage du type `list`. Par exemple, `a[i,:]` renvoie la ligne d'indice `i` du tableau `a`.
- `np.shape` renvoie la taille d'un tableau.
- `np.copy` copie un tableau sans alias.

Chiffrement de l'image. Pour chiffrer l'image (possédant moins de 256 pixels), une colonne de pixels a été ajoutée sur la gauche de l'image. La couleur de ces pixels est décroissante, celui du haut ayant comme valeur 0. Les lignes de l'image ont ensuite été mélangées aléatoirement. Pour reconstituer l'image, il suffit ainsi de trier les lignes de l'image mystère par ordre de premier élément croissant.

1. Écrire une fonction `echange(tab, i, j)` qui échange les lignes d'indice `i` et `j` du tableau `tab`.

2. Écrire une fonction `mini(i0, tab)` qui, parmi les lignes d'indice supérieur ou égal à `i0` du tableau `tab`, renvoie l'indice de la ligne dont le premier élément est minimal.

3. Écrire une fonction `dechiffre_tri(tab)` qui trie les lignes de `tab` par premier élément croissant.

Visualisation de l'image. Le module `matplotlib.pyplot` permet d'obtenir une sortie graphique.

- `import matplotlib.pyplot as plt` charge le module en utilisant le surnom `plt`.
- `f = plt.figure()` crée la figure `f`.
- `plt.imshow(tab, cmap = plt.gray(), origin="lower")` trace le tableau `tab` en niveaux de gris.
- `plt.savefig(nom.png)` enregistre la figure dans le fichier `nom.png`.

4. Déchiffrer l'image `03_mystere1.txt`.

Partie II : Chiffrement par ressemblance

Chiffrement de l'image. La première ligne de l'image initiale n'a pas été modifiée, les suivantes ont été échangées aléatoirement. Pour reconstruire l'image, on cherche successivement la ligne la *plus ressemblante* à la ligne précédente. Pour cela, on dira que deux lignes sont les plus ressemblantes si la différence, en valeur absolue, de leurs pixels situés sur une même colonne est minimale.

5. Écrire une fonction `distance_l1(tab, l1, l2)` qui détermine la distance entre les lignes d'indices `l1` et `l2` du tableau `tab`.

6. Écrire une fonction `mini_l1(i0, tab)` qui, parmi les lignes de `tab` d'indices strictement plus grand que `i0` détermine celle qui est la plus ressemblante à la ligne d'indice `i0`.

7. Écrire une fonction `dechiffre_l1(tab)` qui déchiffre le tableau `tab`.

8. Déchiffrer l'image `03_mystere2.txt`.

Partie III : Lecture & écriture d'un fichier `.pgm`

Jusqu'à présent, nous n'avons pas manipulé de fichier `.pgm` mais leur représentation `Numpy`. Dans cette partie, nous allons manipuler les fichiers pour pouvoir construire un tableau `Numpy` à partir d'une image `PGM` et réciproquement.

Manipulation de fichier.

- `with open(fichier, argt) as f:` permet d'ouvrir (le cas échéant créer) le fichier `fichier` en lecture seule. Si l'argument `argt` prend prend la valeur `"w"`, on peut ensuite écrire dans le fichier par ajout en fin de fichier (si un fichier du même nom existe, il sera écrasé). S'il prend la valeur `"r"` (par défaut), on peut ensuite lire le fichier.
- `f.readline()` permet de lire la chaîne de caractères contenue dans la ligne courante puis de passer à la ligne suivante.
- `f.write(chaine)` permet d'écrire `chaine` à la fin du fichier `f`.
- Le caractère `\n` permet de passer à la ligne.

Compléments `Numpy`.

- `np.concatenate((v1,v2))` permet de concaténer des tableaux, le tableau `v2` s'écrivant à la suite du tableau `v1`.
- `np.reshape(hauteur, largeur)` permet de transformer un tableau ligne en un tableau de nombre de lignes `hauteur` et de nombre de colonnes `largeur`.
- `np.savetxt('fichier.txt',tab)` permet de sauvegarder le tableau `tab` dans le fichier `fichier.txt`.

9. Écrire une fonction `pgm_to_array(fichier_pgm, fichier_txt)` qui prend comme argument le fichier `fichier_pgm` au format PGM et stocke dans le fichier texte `fichier_txt` le tableau correspondant.

10. Écrire une fonction `array_to_pgm(tab, fichier_pgm)` qui permet de transformer le tableau `tab` en un fichier `pgm` nommé `fichier_pgm`.

11. En utilisant les algorithmes précédents, chiffrez et déchiffrez les images de votre choix.
Pour chiffrer des images en mélangeant les lignes, on peut utiliser le module `random`.