PYTHON. **Compression des images**

Partie I. Compression des couleurs

On considère une image définie par une matrice `img` de dimension $n \times m \times 3$:

Chaque pixel est codé par un couple (i, j), avec $0 \leq i < n$, $0 \leq j < m$. Les couleurs de chacun des $n \times p$ pixels (i, j) sont représentés par des entiers r, g et b sont des entiers codés sur 8 bits (donc des entiers positifs entre 0 et 255) représentant les quantités de rouge, vert et bleu du pixel.

Une matrice de dimension $n \times m \times 3$ est une liste de listes de triplets (un triplet étant une 3-liste).

Un vecteur couleur est une liste `[r,g,b]` de longueur 3.

L'image `img` est représentée en machine par une matrice de dimension $n \times mp \times 3$

Pour tous $0 \leq i < n$ et $0 \leq j < m$, `img[i][j]` est un vecteur couleur `[r,g,b]`

Pour tous $0 \leq i < n$, $0 \leq j \leq m$ et $k \in \{0, 1, 2\}$, `img[i][j][k]` est un entier compris entre 0 et 255.

1) [Écrire une fonction `nb_couleurs(img)` qui renvoie le nombre de vecteurs couleurs différents utilisées dans l'image `img`, c'est-à-dire le nombre de `img[i][j]` distincts

Conseil : Utiliser un dictionnaire pour compter les couleurs distinctes. Attention au fait qu'on ne peut pas placer une liste comme clé d'un dictionnaire. Utiliser `str(liste)` pour convertir une liste en chaîne de caractères.

2) On obtient en général un très grand nombre de couleurs différentes.

L'objectif est de réduire ce nombre à seulement 16 couleurs en utilisant l'algorithme des k-moyennes (avec ici $k = 16$) pour regrouper les différents pixels en 16 classes.

Écrire une fonction `dist(p,q)` qui prend pour arguments deux vecteurs couleurs `p` et `q`, et renvoie la distance euclidienne dans $\mathbb{R}^3$ entre ces deux vecteurs couleurs.

3) Écrire une fonction `initialise(img,k)` qui prend pour arguments une image `img`, un entier k et renvoie une liste de longueur k dont les éléments sont des vecteurs couleurs **distincts** choisis au hasard dans l'image.

Remarque :

- Utiliser la fonction `random.randint(0,n)` qui renvoie un entier aléatoire entre 0 et $n - 1$

- Utiliser un dictionnaire pour s'assurer que les éléments sont distincts.

4) On définit la valeur moyenne entière d'une liste triplets d'entiers comme le triplet formé des parties entières des moyennes des trois coordonnées (la fonction `int(x)` renvoie la partie entière d'un réel positif).

Écrire une fonction `moyCouleur(L)` qui étant donnée une liste de triplets d'entiers naturels renvoie la valeur moyenne entière des éléments de `L`.

En déduire une fonction `moyPixel(img,s)` qui prend pour argument une image `img` et une liste `s` de pixels (i, j) et renvoie la moyenne entière des vecteurs couleurs des pixels de l'image `img` associés aux éléments de `s`.

5) Écrire une fonction `PlusProchePixel(p,mu)` qui prend pour argument un pixel p et une liste de pixels $[\mu_0, ..., \mu_{k-1}]$ et qui renvoie l'indice j qui minimise la distance euclidienne $\|p - \mu_j\|$.

6) Une partition en k classes des pixels est définie par une liste de k listes de pixels, de sorte que chaque pixel (i, j), avec $0 \leq i < n$ et $0 \leq j < m$, apparaisse dans une et une seule liste.

En déduire une fonction `kmoyennes(img,k)` qui prend pour arguments une image `img` et un entier k et qui renvoie une partition de longueur k obtenu par l'algorithme des k-moyennes dont le principe est le suivant :

(I) On choisit k vecteurs couleurs $\mu_0, ..., \mu_{k-1}$ qui représentent les futures positions moyennes (cf `initialise`)

(II) On itère jusqu'à ce qu'il y ait convergence de la partition (c'est-à-dire une partition stationnaire) :

(II-A) On associe à chaque pixel (i, j) le vecteur couleur parmi $\mu_0, ..., \mu_{k-1}$ dont sa couleur est le plus proche

On définit ainsi une partition $A_0, ..., A_{k-1}$ de l'ensemble des pixels (i, j)

(II-B) Pour $1 \leq r \leq k$, on calcule le vecteur couleur μ_i qui correspond la moyenne des vecteurs couleurs des pixels appartenant à A_r

7) Une fois la partition obtenue, il reste à calculer la couleur moyenne de chacune des classes et attribuer cette couleur à chacun des pixels de la classe.

Écrire une fonction `reduire(img,k)` qui prend pour argument une image et renvoie une nouvelle image dans laquelle seules k couleurs sont utilisées.

Partie II. Compression par secteurs

On s'intéresse ici à des images carrées en noir et blanc de taille $2^k \times 2^k$.

Autrement dit, une image est donnée par une matrice M telle que pour tous $0 \leq i < 2^k$ et $0 \leq j < 2^k$, $M[i, j]$ est un entier compris entre 0 et 255 donnant l'intensité de noir du pixel (i, j).

Pour simplifier la terminologie, on dira que $M[i, j]$ est la couleur du pixel (i, j).

L'idée pour compresser l'image est de regrouper ensemble les pixels de même couleur qui sont proches.

On va d'abord construire une représentation dite sectorisée de l'image :

On partage l'image en **quatre quadrants** numérotés de 0 à 3 dans l'ordre suivant :

0	1
3	2

Ainsi, le quadrant 0 correspond aux pixels (i, j) vérifiant $0 \leq i < 2^{k-1}$ et $0 \leq j < 2^{k-1}$.

Le quadrant 1 correspond aux pixels (i, j) vérifiant $0 \leq i < 2^{k-1}$ et $2^{k-1} \leq j < 2^k$.

Le quadrant 2 correspond aux pixels (i, j) vérifiant $2^{k-1} \leq i < 2^k$ et $2^{k-1} \leq j < 2^k$.

Et le quadrant 3 correspond aux pixels (i, j) vérifiant $2^{k-1} \leq i < 2^k$ et $0 \leq j < 2^{k-1}$.

On définit alors **la représentation sectorisée** $f(M)$ de l'image M par récurrence sur k :

Si $k = 0$, il y a un seul pixel et $f(M)$ est la liste $[r]$, où r est la couleur du pixel.

Si $k \geq 1$, $f(M)$ est la 4-liste $[f(M_0), f(M_1), f(M_2), f(M_3)]$, où M_0, M_1, M_2 et M_3 sont les sous-images qui correspondent aux quatre quadrants.

Exemple : Si $M = \begin{pmatrix} 100 & 45 & 48 & 5 \\ 240 & 10 & 49 & 3 \\ 27 & 99 & 0 & 6 \\ 12 & 0 & 57 & 23 \end{pmatrix}$, alors

```
f(M)=[[[100],[45],[10],[240]],[[48],[5],[3],[49]],[[0],[6],[23],[57]],[[27],[99],[0],[12]]]
```

8) Écrire une fonction `sectoriser(M,k)` qui étant donnée une matrice M de taille $2^k \times 2^k$ renvoie sa représentation sectorisée `f(M)`.

9) Écrire une fonction `couleur(p,F,k)` qui étant donnés un pixel $p = (i, j)$ et la représentation sectorisée F d'une image de taille $2^k \times 2^k$ renvoie la couleur du pixel (i, j).

Remarque : Utiliser `assert` pour s'assurer que (i, j) correspond bien à un pixel de l'image.

L'intérêt de la représentation sectorisée est de pouvoir compresser l'image lorsque des quadrants entiers sont monocolores.

On définit donc $g(M)$ la **représentation sectorisée compressée** de M de la façon suivante :

- Si tous les pixels ont la même couleur r, $g(M)$ est la liste $[r]$.

C'est naturellement le cas si l'image contient un unique pixel.

- Sinon, $g(M)$ est la 4-liste $[g(M_0), g(M_1), g(M_2), g(M_3)]$, où M_0, M_1, M_2 et M_3 sont les sous-images qui correspondent aux quatre quadrants.

Exemple : Si $M = \begin{pmatrix} 100 & 45 & 48 & 48 \\ 240 & 10 & 48 & 48 \\ 27 & 99 & 0 & 6 \\ 12 & 0 & 57 & 23 \end{pmatrix}$, alors

```
g(M)=[[[100],[45],[10],[240]],[48],[[0],[6],[23],[57]],[[27],[99],[0],[12]]]
```

En pratique, la condition d'égalité des couleurs pour tous les pixels d'un quadrant est trop restrictive.

C'est pourquoi on va considérer qu'un quadrant est monocolore lorsque les couleurs des différents pixels sont proches les unes des autres (dans un sens à préciser).

10) Écrire une fonction `test(L)` qui étant donnée une liste L renvoie `True` ssi L est une liste de 4 singletons, c'est-à-dire L de la forme $[[r_0],[r_1],[r_2],[r_3]]$, où les r_i sont des entiers.

On utilisera les fonctions booléennes `isinstance(x,int) et isinstance(x,list)` qui renvoie `True` ssi x est un entier, respectivement une liste.

11) Écrire une fonction `ecart(L)` qui étant donnée une liste L de la forme $[[r_0],[r_1],[r_2],[r_3]]$ calcule la moyenne μ de ces quatre couleurs et l'écart-type $\sigma = \sqrt{\sum_{i=1}^{4}(r_i-\mu)^2}$, et renvoie le couple (σ, r), où r est l'entier le plus proche de μ.

Remarque : La fonction `round` renvoie l'entier le plus proche ; par exemple `round(1.9)` renvoie 2.

L'algorithme de compression est le suivant :

- On choisit une valeur seuil $\delta > 0$.

- On part de la représentation sectorisée F d'une image.

- On parcourt cette représentation en profondeur de sorte à traiter d'abord les petits quadrants. Un quadrant dont les 4 sous-quadrants sont monocolores est donc représenté par $[[r_0],[r_1],[r_2],[r_3]]$; le quadrant est alors considéré comme monocolore ssi $\sigma \leq \delta$, et dans ce cas, la représentation du compressée du quadrant est donc $[r]$.

- *Important* : Ces compressions s'effectuent en cascade, de bas en haut : des petits quadrants vers les grands quadrants.

12) Écrire une fonction `comprime(F,delta)`
qui renvoie la représentation compressée d'une représentation sectorisée F.