TP 8 : IMAGES ET BDD

L'objectif de ce TP est de rappeler quelques technique de traitement des images d'une part, et d'effectuer des révisions sur les bases de données d'autre part.

1 Rappels sur le traitement des images

Une manière de représenter informatiquement une image est de l'échantillonner en petits éléments (picture element : pixel) supposés de couleur uniforme, suivant un quadrillage. L'image est alors représentée informatiquement par une matrice (carte de points : bitmap), chaque case de la matrice donnant la couleur du pixel correspondant du quadrillage. Les formats les plus connus d'images bitmap sont JPEG, BMP, GIF, TIFF et PNG. La résolution de l'image dépend alors de la finesse du quadrillage.
Selon le format de l'image bitmap, la palette de couleurs est encodée sur un certain nombre de bits, appelée profondeur de l'image (exprimée en bpp = bits par pixel). Les principaux cas sont :

- les images en noir et blanc : 1 bit par pixel
- les images en niveaux de gris : chaque nuance est encodée sur 1 octet, par un nombre compris entre 0 (pour le noir) et 255 (pour le blanc).
- les images couleur au format RGB (Red, Green, Blue). Chaque pixel est décrit par une quantité de rouge, une quantité de vert, et une quantité de bleu. La couleur résultante est obtenue par synthèse additive. On utilise 1 octet pour la quantité de rouge, 1 octet pour la quantité de vert, et 1 octet pour la quantité de bleu, si bien qu'une couleur est représentée sur 3 octets=24 bits. En pratique, on donne un triplet de nombres entre 0 et 255 ((255,0,0) : rouge , (0,255,0) : vert , (0,0,255) : bleu).

En pratique, pour transformer notre image en tableau numpy, nous utiliserons le module PIL (aucune connaissance sur ce module ou un autre n'est exigible).

```
from PIL import Image
import numpy as np
Im=Image.open("image.jpeg")
ImTab=np.array(Im)
```

`ImTab` contient alors un tableau numpy à 3 dimensions (dans la cas d'une image en couleur, 2 pour une image en niveau de gris) de format $(h, l, 3)$, h étant la hauteur de l'image et l sa largeur.
Tester les commandes numpy suivantes :

```
print(np.shape(Imtab))
print(ImTab[i,j]) ; print(ImTab[i,j,:])
print(np.mean(ImTab[0,:,0])) ; print(np.mean(ImTab[0:10,0:10,0]))
```

On peut modifier la valeur d'un pixel :

```
ImTab[0,0] = [0,0,0]
```

Pour afficher une image à partir de son tableau représentatif :

```
Im = Image.fromarray(ImTab)
Im.show()
```

Pour la sauvegarder :

```
Im.save('nouvelle_image.png',"PNG")
```

(on peut utiliser d'autres format comme .jpeg).
Pour créer de nouveaux tableaux vides destinés à construire des images :

```
h , l = 100 , 200 # dimensions
ImtabCouleur = np.zeros((h,l,3),dtype=np.uint8)
ImtabNB = np.zeros((h,l),dtype=np.uint8)
```

2 Premières manipulations

On testera les fonctions sur l'image ouverte ci-dessus. Les fonctions seront écrites pour des (tableau représentant des) images en couleurs.

Coin

Écrire une fonction `coin` qui prend en entrée un tableau représentant une image et renvoyant le tableau représentant l'image située dans le coin en bas à droite de l'image (hauteur 200 pixels, largeur 300). On suppose que le tableau initial est assez grand.
Remarque : cette fonction et les suivantes restent au niveau des tableaux numpy. C'est dans l'utilisation de ces fonctions que l'on traitera les images à proprement parler.

Négatif

Écrire une fonction `neg` qui prend en entrée un tableau `Tab` représentant une image et renvoyant un tableau représentant le négatif de cette image.
Par exemple, si un pixel a pour composantes RGB (232,12,120), alors son pixel correspondant dans le négatif a pour composantes RGB (255-232, 255-12, 255-120).

Pixellisation

On veut alléger le poids d'une image en regroupant les pixels. Pour cela, on va remplacer chaque carré de $p \times p$ pixels par un seul pixel contenant la moyenne des pixels du carré (la nouvelle image sera donc p^2 fois plus légère). Si une des dimensions n'est pas un multiple de p, on tronque l'image.
Écrire une fonction `pixellisation(tab,p)` qui, à partir d'un tableau représentant une image, crée un nouveau tableau selon ce principe.

3 Tracés de segments et couronnes

On travaille ici avec des images en niveau de gris.

1. Ecrire une fonction `TraceTrajet` prenant en entrée quatre entiers x1, y1, x2 et y2 ainsi qu'une matrice `tab` représentant une image et transformant `tab` de sorte à blanchir (c'est à dire mis à la valeur 255) le segment reliant (x1,y1) et (x2, y2). Pour obtenir quelque chose de plus visible, on prendra des "traits" d'épaisseur 5.
2. Ecrire une fonction `TraceCouronne` prenant en entrée quatre entiers x, y, r et E et une matrice `tab` représentant une image qui traçe la couronne de centre (x,y) et de rayon de r à r+E dans la matrice `tab` (toujours en blanc).

4 Base de données

4.1 Données

Les données nécessaires à la suite de ce TP sont sur votre espace de travail : ScenariosSDA.db pour la base de donnée et JCSA-map.png pour l'image.
L'image contient la carte des terres du milieu.
La base de données ScenariosSDA.db est constituée de cinq relations : regions, coordonneescentre, scenario1, scenario2 et scenario3 avec les schémas relationnels suivants :

- regions : ((Numero, entier), (Nom, chaîne de caractères), (Type, chaîne de caractères))
- coordonneescentres : ((Numero , entier), (Abscisse, entier), (Ordonnee, entier))
- scenario1,scenario2, scenario3 : ((Etape, entier), (Region, entier))

La relation regions contient les régions des terres du milieu, que l'on a numérotées à l'aide d'entiers, et dont on donne le nom et le type (Foret, Terres Libres ou autre...). La relation coordonneescentres contient les numéros de régions avec les abscisses et ordonnées de leur centre sur la carte des terres du milieu. Les relations scenario1, scenario2 et scenario3 contiennent des itinéraires données par un entier correspondant au numéro de l'étape et un autre désignant la région traversée.
Notre objectif est de créer un programme représentant sur la carte des terres du milieu le trajet suivi par un scénario parmi les trois de la base de données.

4.2 Manipulation de la base de données

On peut utiliser python pour effectuer les requêtes SQL. Le résultat sera alors la liste des m-uplets contenant les attributs de la relation obtenue par la requête. Vous pouvez ainsi tester la suite d'instruction suivante (on ne cherchera pas à expliquer les détails) :

```
import sqlite3 as lite
con=lite.connect('ScenariosSDA.db')
cur=con.cursor()
cur.execute('SELECT * FROM scenario2;')
data = cur.fetchall()
print(data)
```

Déterminer les requêtes SQL à faire pour obtenir des relations donnant :

1. tous les noms de regions ainsi que les coordonnées de leur centre,
2. l'ensemble des régions traversées au cours des scénarios 1, 2 et 3,
3. l'ensemble des régions n'étant pas traversées dans le scénario1,
4. le nombre total de régions,
5. les noms des régions de forêt.

Tester au fur et à mesure chacune de ces requêtes à l'aide de la procédure décrite précédemment.

4.3 Tracé d'un trajet suivant un scénario

On en vient maintenant à notre objectif, qui est de créer un programme représentant sur la carte des terres du milieu le trajet suivi par un scénario parmi les trois de la base de données (après avoir demandé un numéro de scénarion). On travaillera avec la carte en noir et blanc. Pour cela, on transforme en niveau de gris par :

```
plan = Image.open("JCSA-map.png")
plan = plan.convert('L')
imtab = np.array(plan)
```

Il faut ensuite charger la base de données, demander un numéro de scénario, puis tracer les étapes de ce scénario sur la carte et enfin afficher celle-ci.