Ce sujet est adapté des épreuves d'informatique issues des concours (Centrale, MP, 2020) et (Centrale, MP, 2001).

Les deux parties sont indépendantes.

I Photomosaique

Une *photomosaïque* est une image composée à la manière d'une mosaïque, où les fragments sont eux-mêmes des petites images, appelées *vignettes*. Elle est créée à partir d'une image appelée *image source*. Chaque vignette remplace une zone de même forme dans l'image source appelée *pavé*. Les vignettes sont fabriquées à partir d'une collection d'images appelée *banque* d'images.

L'intérêt est essentiellement artistique : vue de loin, une photomosaïque ressemble à l'image source ; en se rapprochant, on reconnait les vignettes.

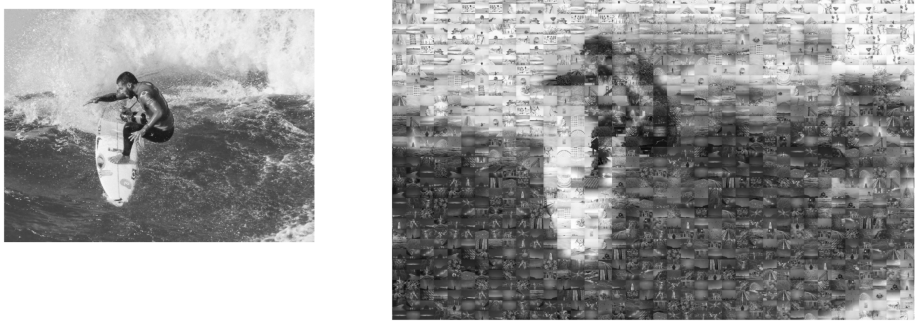

Figure 1: Photomosaïque d'un surfer composé de 1600 vignettes.

Dans ce sujet, on considère des photomosaïques en nuances de gris. Dans un premier temps, on étudie les algorithmes de redimensionnement d'image, permettant de transformer les images de notre banque d'images en vignettes. Dans un second temps, on s'intéresse au placement de ces vignettes pour former de telles mosaïques.
Dans tout le problème, on suppose que les bibliothèques `numpy`, `random`, `math` et `matplolib` ont été importées par

```
import math
import random
import numpy              as np
import matplotlib.pyplot as plt
```

I.1 Images

Une image en niveaux de gris de taille $l \times h$ (pour *largeur* et *hauteur*) est associé à un tableau d'octets (type `np.uint8`) à deux dimensions, à h lignes et l colonnes. Chaque élément de ce tableau représente le niveau de gris du pixel correspondant, 0 correspondant à un pixel noir. Ainsi, le tableau à deux dimensions `img1`, défini par :

```
img1 = np.array([[ 85,   9, 127, 170,  85, 150],
                 [119, 102, 102, 123,  81, 170],
                 [255, 170,  90, 112,  63,  97],
                 [171, 212, 225, 186, 162, 171]], np.uint8)
```

définit une image de taille 6×4, représentée ci-après. Dans toute la suite, on utilise le type `image` pour désigner un tableau d'octets à deux dimensions.

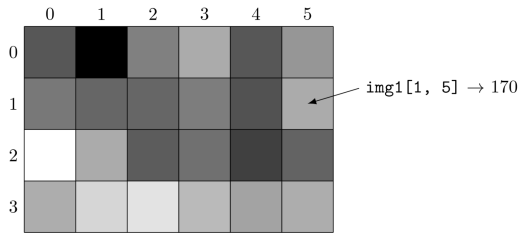

Figure 2: Visualisation de l'image `img1`.

Pour les images en couleurs, on ajoute une dimension supplémentaire, de même type `uint8`, pour représenter les trois composantes (rouge, verte et bleue) d'un pixel. L'instruction

```
source = plt.imread("surfer.jpg")
```

charge dans un tableau `numpy` l'image en couleurs contenue dans le fichier `surfer.jpg`.

Remarque. À l'exception de `/` qui renvoie toujours un élement de type `float`, les opérateurs (`+`, `-`, `*`, `//`, `%`, `**`) renvoient un résultat du même type que leurs deux opérandes. En particulier, cela peut mener à un dépassement de capacité et à une erreur de calcul *invisible* (les dépassements de capacité étant par défaut silencieux).

▸ **Question 1.** *On pose* `a = np.uint8(280)` *et* `b=np.uint(240)`*. Que valent* `a`*,* `b`*,* `a+b`*,* `a-b`*,* `a//b` *et* `a/b` *?*

▸ **Question 2.** *Écrire une fonction* `conversion(img: np.ndarray) -> image` *qui génère une image en niveaux de gris correpondant à la conversion de l'image en couleurs* `img`*. On précise que le niveau de gris d'un pixel correspond à la meilleure approximation entière de la moyenne des intensités des trois composantes.*

I.2 Redimensionnement d'images

On s'intéresse dans cette partie à plusieurs algorithmes de redimensionnement d'une image `A`, de taille $L \times H$ (pour *largeur* et *hauteur*), en une image `a` de taille $l \times h$. On note $N = LH$ le nombre de pixels de l'image `A`, et $n = lh$ celui de l'image `a`.

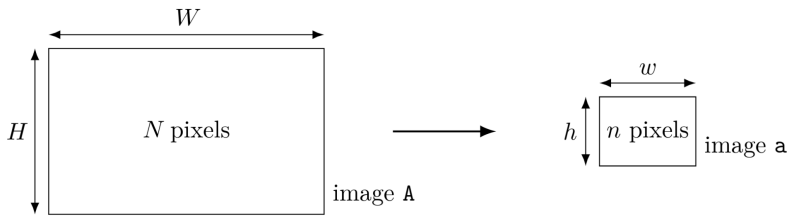

Figure 3: Redimensionnement d'image.

I.2.1 Algorithme d'interpolation au plus proche voisin.

Cette interpolation est définie par la formule $a(i,j) = A\left(\left\lfloor \frac{iH}{h} \right\rfloor, \left\lfloor \frac{jW}{w} \right\rfloor\right)$ où $\lfloor x \rfloor$ désigne la partie entière de x.

▸ **Question 3.** *Écrire une fonction* `procheVoisin(A:image, l:int, h:int) -> image` *qui renvoie une nouvelle image correspondant au redimensionnement de l'image* `A` *à la taille* $l \times h$ *en utilisant l'interpolation au plus proche voisin.*

▸ **Question 4.** *Quelle est sa complexité temporelle asymptotique ?*

I.2.2 Algorithme de réduction par moyenne locale

On suppose ici que les dimensions de l'image `a` divient celles de l'image `A`: H/h et L/l sont des entiers. Afin d'améliorer la qualité de la réduction, on propose la fonction `moyenneLocale`.

```
def moyenneLocale(A:image, l:int, h:int) -> image:
    a = np.empy((h,w), np.uint8)
    H, W = A.shape
    ph, pl = H//h, W//w
    for I in range(0, H, ph):
        for J in range(0, W, pw):
            a[I//ph, J//pw] = round(np.mean(A[I:I+ph, J:J+pw]))
    return a
```

▸ **Question 5.** *Expliquer brièvement son fonctionnement et donner sa complexité temporelle asymptotique.*

I.2.3 Optimisation de la réduction par moyenne locale

Afin d'accélérer le calcul de la moyenne locale, on précalcule pour chaque image sa table de sommation. La table de sommation d'une image `A` de N pixels, représentée par le tableau A à H lignes et L colonnes, est le tableau `S` à $H+1$ lignes et $L+1$ colonnes, défini par

$$\forall l \in [\![0, H]\!]\,,\ \forall c \in [\![0, L]\!]\,,\ S(l,c) = \sum_{\substack{0 \leqslant i < l \\ 0 \leqslant j < c}} A(i,j),$$

la somme ne comportant aucun terme étant prise nulle.

▸ **Question 6.** *Le type* `np.uint32` *(entier non signé codé sur 32 bits) est-il suffisant pour stocker les élements de* S *si l'image* `A` *comporte 50 millions de pixels ? Justifier. Si non, proposer un type python* `np.uint__` *suffisant.*

▸ **Question 7.** *Écrire une fonction* `tableSommation(A:image) -> np.ndarray` *qui calcule la table de sommation de l'image* `A`*, de complexité temporelle asymptotique* $O(N)$.

On suppose à nouveau que les dimensions de `a` divisent celles de `A`.

▸ **Question 8.** *Proposer une fonction* `reductionSommation(A:image, l:int, h:int) -> image`*, dérivée de* `moyenneLocale`*, qui tire parti de la fonction précédente. Comment se compare-t-elle, en temps et en espace, à* `moyenneLocale` *?*

I.3 Placement des vignettes

Dans cette partie, on s'intéresse au cas où l'image source a les même proportions que les vignettes et est constituée de p vignettes de haut sur p vignettes de large. Le nombre total de vignettes est donc $r = p^2$.

On définit la norme L_1 entre deux images a et b de même taille $w \times h$ par :

$$L_1(a,b) = \sum_{\substack{0 \leqslant i < j \\ 0 \leqslant j < w}} |a(i,j) - b(i,j)| \,.$$

▸ **Question 9.** *Écrire une fonction* `L1(a:image, b:image) -> int` *qui calcule la distance L_1 entre deux images de même taille, en prenant garde aux dépassements de capacité.*

▸ **Question 10.** *Écrire une fonction* `choixVignette(obj:image, vignettes:[image]) -> int` *qui renvoie l'indice* `i` *tel que* `L1(obj, vignettes[i])` *soit minimal. L'image* objectif *(*`obj`*) est de même taille que les vignettes. Cette fonction ne doit pas modifier la liste des vignettes.*

Pour construire une photomosaïque, l'image source est redimensionnée aux dimensions de la photomosaïque, qui dépend des dimensions d'une vignettes et du nombre de vignettes sur la photomosaïque. Ensuite, chacun des p^2 blocs de l'image source redimensionnée est remplacé par la vignette qui minimise la norme L_1 entre la vignette et le bloc.

▸ **Question 11.** *En déduire une fonction qui construit une mosaïque de $p \times p$ vignettes, d'en tête* `construireMosaique(source:image, vignettes:[image], p:int) -> image`*. On suppose que toutes les vignettes sont même taille. Quelle est sa complexité temporelle en fonction de la taille $n = hw$ des vignettes, du nombre r de vignettes dans la mosaïque et de la longueur q de la liste vignettes.*

I.4 Améliorations

Remarque. Cette dernière question demande l'initiative de la part du candidat, qui peut-être être amené à définir de nouvelles variables, structures de données et fonctions. Il est demandé d'expliciter clairement la démarche utilisée, de préciser le rôle de chaque nouvelle fonction et variable introduite et de les illlustrer par un schéma le cas échéant. Toute démarche pertinente, même non aboutie, sera valorisée. Le barème prend en compte le temps nécessaire à la résolution de cette question.

La méthode proposée dans la partie précédente peut conduire à sélectionner répititivement les mêmes vignettes et à mal les répartir. En particulier, une plage uniforme de l'image source conduit à l'accumulation de la même vignette dans cette zone de la photomosaïque.

▸ **Question 12.** *Proposer une stratégie de construction de photomoaïque permettant de sélectionner un maximum de vignettes différentes et, au cas où une vignette serait réutilisée, d'éviter que les différentes apparitions de la même vignette se retrouvent trop proches. Il n'est pas nécessaire d'implanter effectivement cette stratégie, un pseudo-code rendant compte des difficultés techniques suffisant.*

II Algorithmes pour la sélection

Dans cette partie, on s'intéresse suivant, appelé *problème de la sélection* : on a un ensemble E de n entiers positifs distincs, un entier i inférieur ou égal à n, et on recherche le i-ème élément de E classé dans l'ordre croissant. Le *médian* d'un ensemble de n nombres est le $\lceil n/2 \rceil$-ème lorsqu'ils sont rangés en ordre croissant.

II.1 Recherche des deux plus grands éléments

Les éléments de E sont donnés non classées dans un tableau T. Soit $p \in \mathbb{N}^*$. On organise un *tournoi* entre les $n = 2^p$ entiers $e_1, \cdots, e_n$ au sens suivant : on consititue un arbre binaire parfait de hauteur p dont les 2^p feuilles sont étiquetées par les e_i. En suite, pour h allant de $p-1$ à 0, les étiquettes des noeuds de profondeur h sont égales au maximum des étiquettes de leurs deux enfants.

▸ **Question 13.** *Donner le nombre de comparaisons nécessaires pour réaliser un tel tournoi. En déduire qu'à l'aide de ce procédé, on peut trouver le maximum et le second plus grand éléments des e_i en moins de $n + \log_2(n)$ comparaisons.*

▸ **Question 14.** *Appliquer la méthode précédente à `[3, 6, 8, 20, 2022, 1515, 17, 1]`.*

▸ **Question 15.** *Comment généraliser cette méthode lorsque n n'est pas une puissance de 2 ? Appliquer à l'entrée `[1, 1024, 1515, 5, 13, 4]`.*

II.2 Une borne inférieure de comparaisons pour un extremum

On considère un algorithme $\mathcal{A}$ prenant en entrée n entiers, et retournant le plus grand de ces entiers. Cet algorithme exécute des comparaisons entre des élémetns du tableau, et que le résultat retourné ne dépend que de l'ensemble des résultats des comparaisons effectuées.

On suppose qu'il existe une entrée $e = (e_1, \cdots, e_n)$ telle que $\mathcal{A}$ exécute strictement moins de $n-1$ comparaions, lorsqu'il est exécuté sur l'entrée e.

▸ **Question 16.** *Montrer que si (S, A) est un graphe, donnée de l'ensemble de ses sommets et de l'ensemble de ses arêtes, connexe, alors $|A| \geqslant |S| - 1$ (où $|X|$ désigne le cardinal de l'ensemble X).*

▸ **Question 17.** *Construire une entrée $e' = (e'_1, \cdots, e'_n)$ telle que l'algorithme $\mathcal{A}$ ne retourne pas le plus grand élément de e'. On pourra considérer le graphe $(\llbracket 1, n \rrbracket, C)$, où C est l'ensemble des (i, j) telles que e_i et $_j$ ont été comparés lors de l'exécution de $\mathcal{A}$ sur e. Conclure.*

II.3 Tri rapide sur liste chaînée

Dans cette sous-partie, on restreint le langage `Python` afin de réaliser le tri rapide comme il le serait avec des listes chaînées. On rappelle qu'une liste chaînée est constituée de noeuds chaînés entre eux. Chaque noeud contient une donnée, ainsi qu'un pointeur vers le noeud suivant. Ce pointeur point vers `None` pour le dernier élément de la liste. Ce qu'on appelle liste est en fait un pointeur vers le premier élément de cette chaîne.

Pour imiter ce comportement, on défini une classe *jouet* `LinkedList`.

```
class LinkedList:
    def __init__(self, li:list):
        self.li = li

    def get_list(self) -> list:
        'Renvoie la liste'

    def get_head(self):
        'Renvoie le premier élément de la liste'

    def get_tail(self):
        'Renvoie la queue de la liste, sous forme de LinkedList'

    def prepend(self, element) -> None:
        'Ajoute un element en tête de liste'
```

▸ **Question 18.** *Écrire une fonction* `partition` *qui, étant donné une liste chaïnée d'entiers liste, et un entier pivot (appartenant ou non à la liste) constitue deux listes* `avant` *et* `après` *constituées des éléments de* `liste` *et telles que*

$$\forall (x, y) \in \texttt{avant} \times \texttt{après},\ x \leqslant \texttt{pivot} < y,$$

et les retourne.

▸ **Question 19.** *Écrire une fonction* `recherche` *récursive qui, pour un entier i et une liste d'entiers* `liste`*, retourne le i-ème élément de la liste. On utilisera la fonction* `partition` *précédente, en prenant comme pivot le premier élément de la liste. Vous justifierez la terminaison et la correction de cette dernière.*

▸ **Question 20.** *Donner des indications sur son temps de calcul, en exhibant notamment les "cas limites". Si la distribution des entiers en E est uniforme dans un intervalle, comment choisir l'entier pivot ?*

II.4 Utilisation d'arbres binaires de recherche

Afin de pouvoir insérer puis rechercher rapidement n'importe quel élément de l'ensemble E, on organise les données selon une structure d'arbres binaires de recherche. On définit pour cela une classe `Node`.

```
class Node:
   def __init__(self, data):
      self.left  = None
      self.right = None
      self.data  = data
      self.size  = 1
```

Chaque noeud de la structure d'arbre binaire contient la valeur d'un entier de E, la taille (comptée en nombre de noeud non vides) de l'arbre de racine N, et des pointeurs vers les deux fils.

▸ **Question 21.** *Donner un schéma de la structure de l'arbre binaire obtenu après insertion des éléments (dans cet ordre)*

```
[3, 5, 1, 13, 12, 14, 9, 4, 6, 1, 8, 10, 0, 2]
```

dans l'arbre `Node(7)`*.*

▸ **Question 22.** *Écrire une procédure* `insère` *récursive, réalisant l'insertion d'un nouvel entier x dans l'arbre binaire de recherche a.*

▸ **Question 23.** *En supposant que l'arbre est bien équilibré (en un sens que l'on précisera), indiquer un ordre de grandeur du temps d'exécution de la fonction* `insère` *en fonction de la taille de l'arbre dans lequel on insère le nouvel élément. Que se passe-t-il si l'arbre n'est pas équilibré ? Citer un cas où cela peut se produire.*

▸ **Question 24.** *Écrire une fonction* `recherche`*, récursive, déterminant le k-ème élément de l'ensemble des étiquettes d'un arbre binaire de recherche.*

••• FIN •••