teaching sciences UPSTI for innovation

Proposition de corrigé

Concours : Concours Centrale-Supélec

Année : 2020

Filière : MP - PC - PSI - TSI

Épreuve : Informatique

Ceci est une proposition de corrigé des concours de CPGE, réalisée bénévolement par des enseignants de Sciences Industrielles de l'Ingénieur et d'Informatique, membres de l'UPSTI (Union des Professeurs de Sciences et Techniques Industrielles), et publiée sur le site de l'association :

https://www.upsti.fr/espace-etudiants/annales-de-concours

A l'attention des étudiants

Ce document vous apportera des éléments de corrections pour le sujet traité, mais n'est ni un corrigé officiel du concours, ni un corrigé détaillé ou exhaustif de l'épreuve en question.

L'UPSTI ne répondra pas directement aux questions que peuvent soulever ces corrigés : nous vous invitons à vous rapprocher de vos enseignants si vous souhaitez des compléments d'information, et à vous adresser à eux pour nous faire remonter vos éventuelles remarques.

Licence et Copyright

Toute représentation ou reproduction (même partielle) de ce document faite sans l'accord de l'UPSTI est **interdite**. Seuls le téléchargement et la copie privée à usage personnel sont autorisés (protection au titre des droits d'auteur).

En cas de doute, n'hésitez pas à nous contacter à : corrigesconcours@upsti.fr.

Informez-vous !

Retrouvez plus d'information sur les Sciences de l'Ingénieur, l'orientation, les Grandes Ecoles ainsi que sur les Olympiades de Sciences de l'Ingénieur et sur les Sciences de l'Ingénieur au Féminin sur notre site : www.upsti.fr

L'équipe UPSTI

Photomosaïque

Corrigé UPSTI

INFORMATIQUE

I - Pixels et images

I.1 - Pixels

Un pixel (contraction de l'anglais *picture element*) est un élément de couleur homogène utilisé pour représenter une image sous forme numérique. La teinte d'un pixel peut être représentée de plusieurs façons. Une méthode courante, basée sur la synthèse additive, consiste à la décomposer en trois composantes qui correspondent aux couleurs rouge, vert et bleu. On parle de représentation RGB (pour red, green et blue). Chacune des trois composantes donne l'intensité de la couleur correspondante dans la teinte finale, 0 indiquant l'absence de cette couleur. Ainsi, le triplet (0, 0, 0) désigne un pixel noir.

Question 1 On suppose que chacune des trois composantes RGB d'un pixel est représentée par un nombre entier positif ou nul, codé sur 8 bits. Combien de couleurs différentes peut-on représenter avec un tel pixel ?

Avec trois composantes (R, G et B) de chacune huit bits, on peut réaliser $2^8 \times 2^8 \times 2^8 = 2^{24}$, soit 16 777 216 combinaisons.

Dans la suite, on représente un pixel par un vecteur (tableau numpy à une dimension) d'entiers de type `np.uint8` (entier non signé codé sur 8 bits) à trois éléments, correspondant respectivement à chacune des composantes RGB du pixel ; on utilise dans toute la suite le type pixel pour désigner un tel vecteur.

Question 2 Donner une instruction permettant de créer un vecteur correspondant à un pixel blanc.

Une instruction permettant de créer un vecteur correspondant à un pixel blanc peut être la suivante :

```
pixelBlanc = np.zeros((3,), dtype=np.uint8)
```

Remarque : `np.zeros((3,), dtype = np.uint8)` est une version simplifiée de `np.zeros((1,3), dtype = np.uint8)`

Il est rappelé qu'en Python, comme dans beaucoup de langages de programmation, les opérations d'addition, soustraction, multiplication, division entière, modulo et élévation à la puissance (opérateurs `+`, `-`, `*`, `//`, `%`, `**`) appliquées à deux opérandes de même type fournissent un résultat du type de leurs opérandes. Cela peut conduire à un dépassement de capacité et à une erreur de calcul car, les dépassements de capacité étant par défaut « silencieux », ils ne produisent pas d'erreur lors de l'exécution du programme. L'opérateur division (`/`) entre deux entiers produit toujours un résultat sous forme de nombre à virgule flottante même si la division est exacte (12 / 2 → 6.0). Il en est de même pour toute fonction faisant implicitement appel à cet opérateur comme `np.mean`.

© UPSTI - Toute reproduction interdite sans l'autorisation de l'UPSTI

Question 3 On pose `a = np.uint8(280)` et `b = np.uint8(240)`. Que valent `a`, `b`, `a+b`, `a-b`, `a//b` et `a/b` ?

Un octet est codé sur 256 bits (2^8 = 256, avec des valeurs allant de 0 à 255), lorsque l'on définit a avec une valeur de 280, on a un dépassement de capacité qui génère une erreur de calcul, soit :

$$a = 280 - 256 = 24$$

Il n'y a pas de problèmes de dépassement de capacité avec l'assignation de la variable b :

$$b = 240$$

Pour l'addition, on a également un dépassement de capacité car a + b = 24 + 240 = 264, le résultat ne pouvant être compris qu'entre 0 et 255, l'addition donne :

$$a + b = 264 - 256 = 8$$

Pour la soustraction, on obtient un chiffre négatif (a – b = 24 – 240 = -216) alors que le type de la variable est un octet non signé, il faut alors ajuster le résultat :

$$a - b = -216 + 256 = 40$$

Pour la division entière, il n'y a pas de problèmes de dépassement de capacité :

$$a//b = 24//240 = 0$$

Finalement, pour la division, il n'y a pas de problèmes non plus, mais le résultat obtenu devient un nombre à virgule flottante :

$$a/b = 0.1$$

Les fonctions numpy qui effectuent de manière répétitive des opérations élémentaires, si elles ne garantissent pas l'absence de dépassement de capacité, prennent la précaution d'utiliser pour leurs calculs intermédiaires et leur résultat un type compatible avec le type de base de la plus grande capacité possible. Par exemple le résultat de `np.sum(np.array([100, 200], np.uint8))` est de type `np.uint64` (entier non signé codé sur 64 bits) et vaut bien 300.

Pour représenter une image en niveau de gris, on peut se contenter d'une valeur par pixel, représentant l'intensité du gris entre le noir et le blanc. Pour convertir une image en couleurs en niveaux de gris, on peut remplacer chaque pixel par un seul entier, dont la valeur correspond à la meilleure approximation entière de la moyenne des trois composantes RGB du pixel.

Question 4 Écrire une fonction d'entête
`def gris(p:pixel) -> np.uint8:`
qui calcule le niveau de gris correspondant au pixel p.

Le niveau de gris d'un pixel est définit ici comme étant la meilleure approximation entière de la moyenne des trois composantes RGB du pixel, la fonction calculant le niveau de gris peut donc s'écrire comme suit :

```
def gris(p):
    return np.uint8(round(p.mean()))
```


© UPSTI - Toute reproduction interdite sans l'autorisation de l'UPSTI

I.2 - Images

Une image en niveaux de gris de taille $w \times h$ (w pixels de large, h pixels de haut) est associée à un tableau d'octets (type np.uint8) à deux dimensions, à h lignes et w colonnes. Chaque élément de ce tableau représente le niveau de gris du pixel correspondant. Ainsi le tableau à deux dimensions `img1`, défini par :

```
img1 = np.array([[ 85,   0, 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 Figure 2.

Dans toute la suite, on utilise le type image pour désigner un tableau d'octets à deux dimensions.

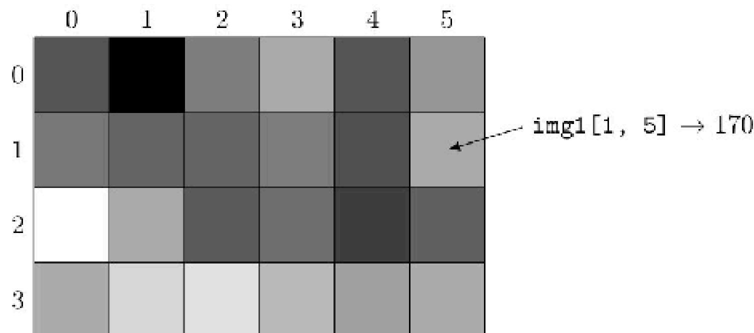

Figure 2 Visualisation de l'image `img1`

Pour les images en couleurs, on ajoute une dimension pour représenter les trois composantes 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`. Les expressions `source.shape` et `source[0,0]` valent alors respectivement :

`(3000, 4000, 3)` et `np.array([144, 191, 221], np.uint8).`

Question 5 Interpréter ces valeurs.

L'instruction `source.shape` renvoie la forme du tableau numpy associé à l'image `surfer.jpg`, soit un tableau de 3000 lignes, 4000 colonnes et comportant 3 informations pour chaque élément du tableau. On en déduit alors que l'image possède 3000 pixels de haut et 4000 pixels de large, chaque pixel contenant les informations de couleur (RGB : *red*, *green* et *blue*).

L'instruction `source[0,0]` renvoie les valeurs de l'élément situé à la position `(0,0)` du tableau (en haut à gauche, *c.f.* Figure 2). Le pixel en haut à gauche de l'image `surfer.jpg` possède une intensité de rouge à 144 (0 étant une absence de couleur), une intensité de vert à 191 et une intensité de bleu à 221. Chaque couleur est codée par un octet et ces trois intensités sont regroupées dans un tableau de type `np.array`.

Question 6 Écrire une fonction d'entête

`def conversion(a:np.ndarray) -> image:`

qui génère une image en niveaux de gris correspondant à la conversion de l'image en couleurs `a`.

Une fonction générant une image en niveaux de gris à partir d'une image en couleur `a` peut s'écrire comme suit :

```
def conversion(a):
      H, W = a.shape
      image = np.empty((H,W),np.uint8)
      for i in range(0,H):
            for j in range(0,W) :
                  image[i,j]=gris(a[i,j])
                  #ou image[i,j]=np.uint8(round(a[i,j].mean()))
      return image
```

Attention, la fonction range génère des nombres compris entre un entier de start et un entier de stop. Exemple :

```
for i in range(0,10) :
            print(i, end=', ')
```

donne le résultat

```
0, 1, 2, 3, 4, 5, 6, 7, 8, 9
```

II - Redimensionnement d'images

On s'intéresse dans cette partie à plusieurs algorithmes de redimensionnement d'une image A, de taille $W \times H$ (W pixels de large par H pixels de haut, on note $N = HW$ son nombre total de pixels), en une image a de taille $w \times h$ (on pose $n = hw$). Nous nous intéresserons dans la suite uniquement à des images en niveau de gris.

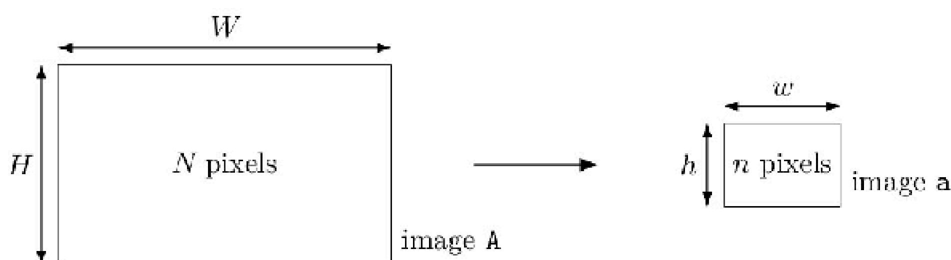

Figure 3 Redimensionnement d'image

II.1 - Le contexte

À l'occasion du mariage d'Alice et de Bernard, leurs amis souhaitent réaliser plusieurs photomosaïques sur des thèmes variés. Ils ont pour cela accumulé un grand nombre de photos au ratio 4:3, ce qui signifie que le rapport W/H vaut exactement 4/3. Les photomosaïques mesureront chacune deux mètres de large et seront constituées de 40 × 40 = 1600 vignettes, toutes de même taille et au même ratio 4:3. Pour garder une bonne qualité d'impression, ils choisissent une résolution de 10 pixels par millimètre.

Question 7 Quelle taille de vignette ($w \times h$, en pixels) faut-il choisir ? Quelle sera alors la taille en pixels de la photomosaïque ?

La photomosaïque fait 40 vignettes de hauteur et 40 vignettes de largeur (40 x 40 = 1600), on sait également qu'elle fait deux mètres de large, donc une vignette fait 2/40 = 0,05 m, soit w = 50 mm de large. La résolution étant de 10 pixels par millimètre, une vignette fait 500 pixels de large.

Le ratio $W/H = w/h = 4/3$, permet d'obtenir la hauteur en pixel d'une vignette, soit $h = 3/4 \ . \ w$, l'application numérique donne une hauteur de vignette de 375 pixels de hauteur.

© UPSTI - Toute reproduction interdite sans l'autorisation de l'UPSTI

h = 375 pixels et w = 500 pixels

La photomosaïque fait 40 vignettes de hauteur et 40 vignettes de largeur, on a donc les dimensions suivantes pour la photomosaïque :

H_photomosaïque = 40 . h et W_photomosaïque = 40 . w

L'application numérique donne les dimensions :

H_photomosaïque = 15000 pixels et W_photomosaïque = 20000 pixels

II.2 - Algorithme d'interpolation au plus proche voisin

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

Question 8 Écrire une fonction d'entête
`def procheVoisin(A:image, w:int, h:int) -> image:`
qui renvoie une nouvelle image correspondant au redimensionnement de l'image `A` à la taille $w \times h$ en utilisant l'interpolation au plus proche voisin.

Une fonction d'entête, renvoyant une image redimensionnée `a`, de l'image initiale `A` par interpolation au plus proche voisin peut s'écrire comme suit :

```
def procheVoisin(A, w, h):
     H, W = A.shape
     a = np.empty((h,w),np.uint8)
     for i in range(0,h):
          for j in range(0,w) :
          a[i,j]=(A[math.floor(i*(H/h)),math.floor(j*(W/w))])
     return a
```

Question 9 Quelle est sa complexité temporelle asymptotique ?

La complexité temporelle asymptotique est le nombre d'opérations élémentaire (affectations, comparaisons, opération arithmétiques) effectuées par l'algorithme. On émet l'hypothèse que toutes les opérations élémentaires sont à égalité de coût. L'analyse asymptotique donne une mesure de l'efficacité d'un algorithme, ne dépendant pas de constantes spécifiques à une machine. Ce nombre s'exprime en fonction de la taille n des données.

```
def procheVoisin(A, w, h):
 H, W = A.shape                                        affectation     : 2
  a = np.empty((h,w),np.uint8)                         affectation     : 1
  for i in range(0,h):                                 itérations      : h
   for j in range(0,w) :                               itérations      : w
    a[i,j]=(A[math.floor(i*(H/h)),math.floor(j*(W/w))])  affectation   : 7
 return a                                              fonction        : 1
```

Soit une complexité temporelle asymptotique de :

2+1+h.w.7 + 1 = 3 + 7n = O(n) avec : n= h.w

© UPSTI - Toute reproduction interdite sans l'autorisation de l'UPSTI

II.3 - Algorithme de réduction par moyenne locale

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

```
1   def moyenneLocale(A:image, w:int, h:int) -> image:
2       a = np.empty((h, w), np.uint8)
3       H, W = A.shape
4       ph, pw = H // h, W // w
5       for I in range(0, H, ph):
6           for J in range(0, W, pw):
7               a[I // ph, J // pw] = round(np.mean(A[I:I+ph, J:J+pw]))
8   return a
```

Question 10 Expliquer en quelques lignes son principe de fonctionnement.

La fonction `moyenneLocale` prend comme argument une image initiale et les dimensions de l'image réduite renvoyée par la fonction. Elle segmente l'image initiale en fonction de la taille de l'image finale à l'aide de divisions entières et chaque pixel de l'image finale est la moyenne locale du segment de l'image initiale associé.

Question 11 Donner sa complexité temporelle asymptotique.

Sa complexité temporelle asymptotique est de :

1+2+2+h.w.3 + 1 = 6 + 3n = O(n)   avec n = h.w

II.4 - 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 W colonnes, est le tableau `S` à $H+1$ lignes et $W+1$ colonnes, défini par

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

la somme étant prise nulle quand elle ne comporte aucun terme.

Question 12 Le type `np.uint32` (entier non signé codé sur 32 bits) est-il suffisant pour stocker les éléments de `S` si l'image `A` comporte 50 millions de pixels ? Justifier.

L'image `A` est composé d'éléments de type `np.uint8` (soit un octet), la table de sommation ayant chaque élément de son tableau définit comme étant la somme des éléments d'indice strictement inférieur lui, on se met dans le cas le plus défavorable (Si A est composé entièrement d'éléments à une valeur de 255), soit :

$$S(H,W) = A.sum = 50.10^{6} \times 255 = 1,275.10^{10}$$

Pour savoir combien de bits sont nécessaires pour coder ce nombre on résout l'inéquation suivante :

$2^n \geq 1,275.10^{10}$

<=> $\exp(n.\ln(2)) \geq 1,275.10^{10}$

<=> $n.\ln(2) \geq \ln(1,275.10^{10})$

<=> $n \geq \ln(1,275.10^{10}) / \ln(2)$

<=> $n \geq 36,9$

© UPSTI - Toute reproduction interdite sans l'autorisation de l'UPSTI

Il faut au moins 37 bits pour stocker les éléments de `S`, si l'image comporte 50 millions de pixels. Le type `np.uint32` n'est donc pas suffisant.

Question 13 Écrire une fonction, de complexité $O(N)$, d'entête

```
def tableSommation(A:image) -> np.ndarray:
```

qui calcule la table de sommation de l'image `A`.

Une fonction de complexité $O(N)$ calculant la table de sommation de l'image `A` peut s'écrire comme-suit :

```
def tableSommation(A):
    H, W = A.shape
    S = np.empty((H+1,W+1), np.uint64)
    for i in range(1,H+1):
        for j in range(1,W+1):
            S[i,j] = S[i-1,j] + S[i,j-1] - S[i-1,j-1] + A[i-1,j-1]
    return S
```

On suppose à nouveau que les dimensions de l'image `A` divisent celles de l'image `a` : H/h et W/w sont entiers. On propose alors la fonction `réductionSommation1`, qui prend en paramètre l'image `A` et sa table de sommation `S (S = tableSommation(A))`, ainsi que les dimensions de l'image que l'on souhaite obtenir.

```
1    def réductionSommation1(A:image, S:np.ndarray, w:int, h:int) -> image:
2        a = np.empty((h, w), np.uint8)
3        H, W = A.shape
4        ph, pw = H // h, W // w
5        nbp = ph * pw
6        for I in range(0, H, ph):
7            for J in range(0, W, pw):
8                X = (S[I+ph, J+pw] - S[I+ph, J]) - (S[I, J+pw] - S[I, J])
9                a[I // ph, J // pw] = round(X / nbp)
10       return a
```

Question 14 Expliquer en quelques lignes le principe de fonctionnement de `réductionSommation1`.

La fonction `réductionSommation1` prend en compte comme arguments : l'image initiale `A`, la table de sommation `S` et les dimensions de la nouvelle image `w` et `h`, et renvoie l'image finale réduite `a`. Elle segmente l'image initiale en fonction de la taille de l'image finale à l'aide de divisions entières et chaque pixel de l'image finale est la moyenne locale du segment de l'image initiale associé.

Cette moyenne est réalisée à l'aide de la table de sommation qui est manipulée de sorte à obtenir X : la somme des valeurs comprises dans la zone (de `ph*pw` pixel de l'image initiale) à réduire en un pixel de l'image finale. Cette somme est finalement moyennée (en la divisant par `nbp` : le nombre de pixel compris dans cette zone) puis affectée à la nouvelle image réduite.

Question 15 Donner sa complexité temporelle asymptotique.

La complexité temporelle asymptotique de la fonction `réductionSommation1` est de :

$$6 + h.w.(4 + 3) + 1 = 7 + 7n = O(n) \qquad \text{avec : } n = h.w$$

© UPSTI - Toute reproduction interdite sans l'autorisation de l'UPSTI

Question 16 Montrer que la fonction `réductionSommation2` dont le code est fourni ci-dessous donne le même résultat que `réductionSommation1`.

```
1     def réductionSommation2(A:image, S:np.ndarray, w:int, h:int) -> image:
2         H, W = A.shape
3         ph, pw = H // h, W // w
4         sred = S[0:H+1:ph, 0:W+1:pw]
5         dc = sred[:, 1:] - sred[:, :-1]
6         dl = dc[1:, :] - dc[:-1, :]
7         d = dl / (ph * pw)
8         return np.uint8(d.round())
```

La fonction `réductionSommation1` utilise une double boucle `for` pour accéder à chaque élément de la matrice `a` (qui est l'image réduite de `A`) et à l'aide d'opérations sur la matrice de sommation `S`, permet d'obtenir la moyenne sur un intervalle de `ph` (sur la hauteur) et `pw` (sur la largeur). On moyenne donc `ph x pw` pixels de l'image originale `A` en un pixel de l'image réduite `a`.

La fonction `réductionSommation2` n'utilise pas de boucle `for`, mais définit une nouvelle matrice `sred`, qui permet d'accéder aux valeurs de la matrice `S` avec un « pas » de `ph` sur les lignes et de `pw` sur les colonnes. Des opérations sur cette matrice `sred`, permettent d'obtenir la valeur moyenne des pixels de l'image originale `A` sur un intervalle de `ph` (sur la hauteur) et de `pw` (sur la largeur) et d'obtenir directement l'image `a`, après avoir pris la valeur arrondie et l'avoir convertie en une valeur sur huit bits.

Question 17 Comparer les complexités asymptotiques en temps et en mémoire des deux versions de la fonction `réductionSommation`. Quel est l'avantage de la seconde version ?

Soit CAT la complexité asymptotique en temps et CAM la complexité asymptotique en mémoire (qui est également appelée complexité en espace), et les indices 1 et 2 représentant respectivement les fonctions `réductionSommation1` et `réductionSommation2`.

$CAT_1 = O(n)$
$CAT_2 = O(n)$ *(les boucles sont implicites)*
$CAM_1 = size(a) + (size(H)+size(W)) + (size(ph)+size(pw)) + size(nbp) + size(X)$
$\quad = n + 2 + 2 + 1 + n = 2n + 5 = O(n)$
$CAM_2 = (size(H)+size(W)) + (size(ph)+size(pw)) + size(sred) + size(dc) + size(dl) + size(d)$
$\quad = 2 + 2 + n + n + n + n = 4n + 4 = O(n)$

La seconde version possède une complexité asymptotique temporelle plus faible que celle de la première version ($O(1) < O(n)$) pour une complexité asymptotique en mémoire équivalente entre les deux versions. L'avantage de la seconde version est qu'elle est plus rapide que la première (on n'utilise plus une double boucle `for`, mais des opérations sur une matrice de réduction).

II.5 - Synthèse

Question 18 Discuter des cas d'usage respectifs de `procheVoisin`, `moyenneLocale` et `réductionSommation` pour redimensionner une image.

La fonction `procheVoisin` perd de l'information (on ne prend que quelques pixels parmi l'image initiale `A`). La fonction `moyenneLocale` ne fonctionne que si les dimensions de l'image `a` divisent celles de l'image `A`. Finalement, la fonction `réductionSommation` permet de réduire la taille de l'image initiale en prenant en compte l'intégralité de ses éléments, tout fonctionnant même si les dimensions de l'image `a` ne divisent pas celles de l'image `A`.

© UPSTI - Toute reproduction interdite sans l'autorisation de l'UPSTI

III - Sélection des images de la banque

Une première étape dans la conception d'une photomosaïque est le choix d'une image source et de vignettes. Cette partie est consacrée à la sélection d'images dans la banque. Les images de la banque sont répertoriées dans une base de données dont le modèle physique est présenté Figure 4, dans laquelle les clés primaires sont notées en italique.

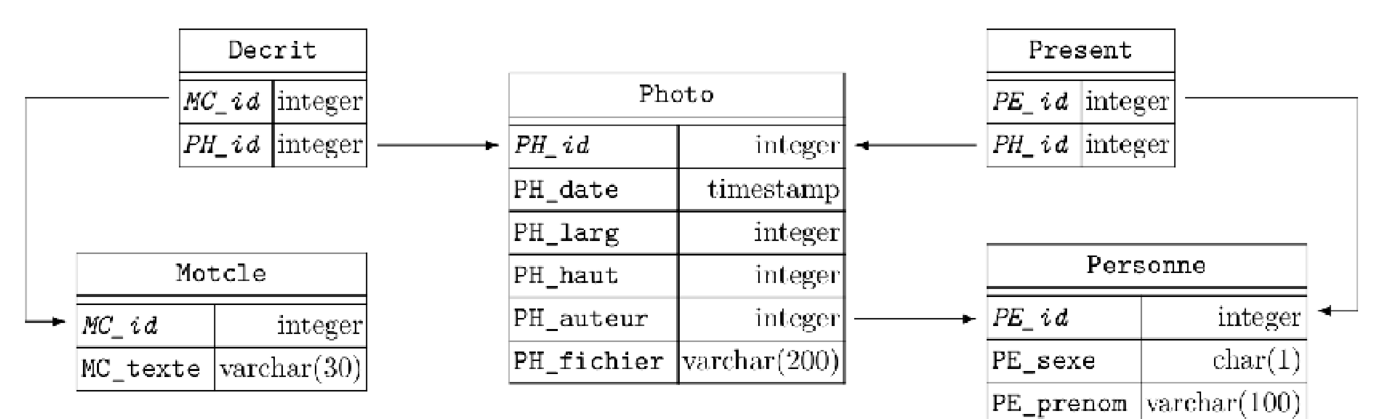

Figure 4 Structure physique de la base de données de photographies.

Cette base comporte les cinq tables listées ci-dessous avec la description de leurs colonnes :

— la table `Photo` répertorie les photographies

- `PH_id` identifiant (entier arbitraire) de la photographie (clé primaire)
- `PH_date` date et heure de la prise de vue
- `PH_larg`, `PH_haut` largeur et hauteur de la photographie en pixels
- `PH_auteur` identifiant de l'auteur de la photographie
- `PH_fichier` nom du fichier contenant la photographie

— la table `Personne` des modèles et des photographes

- `PE_id` identifiant (entier arbitraire) de la personne (clé primaire)
- `PE_sexe` sexe de la personne ('M' ou 'F')
- `PE_prenom` prénom de la personne

— la table `Motcle` des mots-clés utilisés pour décrire une photographie

- `MC_id` identifiant (entier arbitraire) du mot-clé (clé primaire)
- `MC_texte` le mot-clé lui-même

— la table `Decrit` fait le lien entre les photographies et les mots-clés qui les décrivent, ses deux colonnes constituent sa clé primaire

- `MC_id` identifiant du mot-clé (décrivant la photographie)
- `PH_id` identifiant de la photographie (décrite par le mot-clé)

— la table `Present` fait le lien entre les photographies et les personnes qui y figurent, ses deux colonnes constituent sa clé primaire

- `PE_id` identifiant de la personne (figurant sur la photographie)
- `PH_id` identifiant de la photographie (représentant la personne)

III.1 - Quelques requêtes

Pour réaliser les photomosaïques du mariage d'Alice et Bernard, on dispose de plus de 20 000 photographies répertoriées dans une base de données dont le modèle est celui de la Figure 4.

Question 19 Écrire une requête SQL donnant les identifiants de toutes les photographies au ratio 4:3, c'est-à-dire dont le rapport largeur sur hauteur vaut exactement 4/3.

```
SELECT PH_id
FROM Photo
WHERE 3*PH_larg = 4*PH_haut;
```

Question 20 Écrire une requête qui compte le nombre de photos prises par « Alice » ou « Bernard ».

```
SELECT COUNT(*)
FROM Photo
WHERE PE_prenom in ('Alice', 'Bernard') ;
```

Question 21 Écrire une requête qui fournit l'identifiant et la date des photographies prises avant 2006 et associées au mot-clé « surf ».

```
SELECT Photo.PH_id, PH_date
FROM Photo
        JOIN Decrit
    ON Photo.PH_id = Decrit.PH_id
        JOIN Motcle
        ON Decrit.MC_id = Motcle.MC_id
        WHERE EXTRACT(year FROM PH_date) < '2006-01-01' AND MC_texte = 'surf' ;
```

Remarque : au niveau de SELECT, on précise Photo.PH_id, car PH_id existe dans Photo et Decrit

Question 22 Écrire une requête qui donne le prénom de l'auteur et l'identifiant de tous les selfies, c'est-à-dire les photographies sur lesquelles l'auteur est présent.

```
SELECT PE_Prenom, PH_id
FROM Photo
    JOIN Present
    ON Photo.PH_id = Present.PH_id
    JOIN Personne
    ON Photo.PH_auteur = Personne.PE_id;
```


© UPSTI - Toute reproduction interdite sans l'autorisation de l'UPSTI

Question 23 Écrire une requête qui sélectionne toutes les photographies sur lesquelles sont présents « Alice » et « Bernard », à l'exclusion de toute autre personne.

```
( SELECT PH_id
FROM Photo
    JOIN Present
    ON Photo.PH_id = Present.PH_id
        JOIN Personne
        ON Present.PE_id = Personne.PE_id
        WHERE PE_prenom = 'Alice' )
INTERSECT
( SELECT PH_id
FROM Photo
    JOIN Present
    ON Photo.PH_id = Present.PH_id
        JOIN Personne
        ON Present.PE_id = Personne.PE_id
        WHERE PE_prenom = 'Bernard' )
EXCEPT
(SELECT PH_id
FROM Photo JOIN Present USING (PH_id)
GROUP BY PH_id HAVING COUNT(*)>2);
```

III.2 - Internationalisation des mots-clés

Afin de partager et d'enrichir la banque d'images, il a été décidé de faire évoluer la structure de la base de données afin de gérer les mots-clés dans différentes langues. Le cahier des charges de cette évolution stipule :

a) l'ensemble des photographies sélectionnées à l'aide de mots-clés ne doit pas dépendre de la langue utilisée pour exprimer ces mots-clés ; autrement dit, les photographies décrites par le mot-clé « montagne » exprimé en français doivent être les mêmes que celles sélectionnées par les mots-clés « mountain » si la langue choisie est l'anglais, « Berg » pour l'allemand, « montaña » pour l'espagnol, etc. ;

b) il doit être possible, pour cette nouvelle base de données, d'écrire une requête de recherche de photographies par mot-clef en spécifiant la langue utilisée pour exprimer le mot-clé de telle sorte que changer de langue se fasse en modifiant uniquement des constantes dans la clause `WHERE`.

Question 24 Proposer un nouveau modèle de base de données répondant à cette évolution du cahier des charges en ne détaillant que ce qui change (tables modifiées, nouvelles tables).

On modifie la table `Motcle`, en ajoutant une caractéristique « langue » : `MC_lang` (*c.f.* Figure 5 ci-dessous) :

© UPSTI - Toute reproduction interdite sans l'autorisation de l'UPSTI

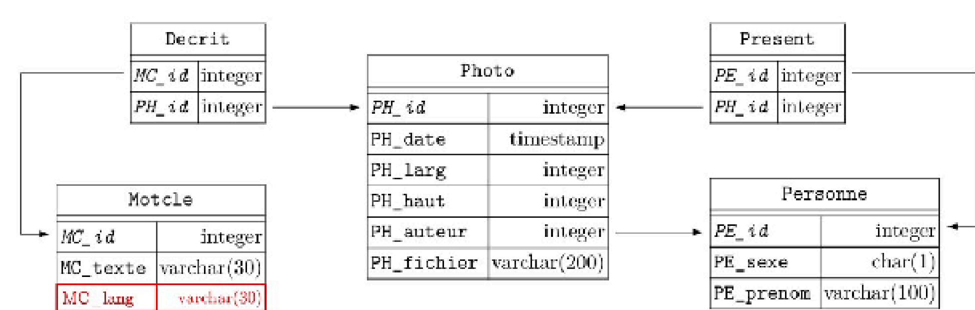

Figure 5 Structure physique de la base de données de photographies.

Question 25 Avec cette nouvelle base de données, écrire une requête qui permet de sélectionner les identifiants des photographies associées au mot-clé « mountain » exprimé en anglais.

```
SELECT Photo.PH_id
FROM Photo
    JOIN Decrit
    ON Photo.PH_id = Decrit.PH_id
        JOIN Motcle
        ON Decrit.MC_id = Motcle.MC_id
        WHERE MC_lang = 'english' AND MC_texte = 'mountain' ;
```

A chaque identifiant lié à une photo (PH_id), on associe un mot-clé (via la table Decrit). La table Motcle permet pour un identifiant de mot-clé donné (MC_id) d'avoir le mot-clé associé (MC_texte) et la langue dans laquelle il est exprimé (MC_lang).

IV - Placement des vignettes

IV.1 - Préparatifs

On envisage ici le cas où la photomosaïque est homothétique de l'image source et constituée de p vignettes de haut sur p vignettes de large. Le nombre total de vignettes est donc $r = p^2$.

Question 26 Écrire une fonction d'entête

```
def initMosaïque(source:image, w:int, h:int, p:int) -> image:
```

qui prend en paramètre l'image source, les dimensions w et h d'une vignette et le nombre p de vignettes par coté. Cette fonction renvoie une version redimensionnée de source, de même taille que la photomosaïque finale.
On rappelle qu'il est possible d'utiliser les fonctions définies précédemment.

Une fonction renvoyant une version redimensionnée de source, de même taille que la photomosaïque, peut s'écrire comme suit :

```
def initMosaïque(source, w, h, p):
    H, W = source.shape
    S = tableSommation(source)
    mosaique = réductionSommation2(source, S, w, h)
    return mosaique
```


© UPSTI - Toute reproduction interdite sans l'autorisation de l'UPSTI

On appelle désormais *pavé* chaque zone de cette image source redimensionnée, de taille $w \times h$, qui doit être remplacé par une vignette. Afin de comparer les vinettes et les pavés, on définit la distance L_1 entre deux images a et b de même taille $w \times h$ par :

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

Question 27 Écrire une fonction d'entête

```
def 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.

Une fonction calculant la distance L_1 entre deux images de même taille, peut s'écrire comme suit :

```
def L1(a,b)
      return np.uint64(np.sum(abs(a-b)))
```

Question 28 Écrire une fonction d'entête

```
def choixVignette(pavé:image, vignettes:[image]) -> int:
```

qui prend en paramètre une image correspondant à un pavé et une liste de vignettes et qui renvoie l'indice `i` tel que `L1(pavé, vignettes[i])` est minimal (ou l'un d'entre eux si plusieurs vignettes conviennent). Cette fonction ne doit pas modifier la liste des vignettes.

Une fonction renvoyant l'indice `i` tel que `L1` est minimal, peut s'écrire comme suit :

```
def choixVignette(pavé, vignettes)
      I = len(vignettes)
      L1_res = np.array(1,I)
      for i in range(0,I)
            L1_res[i]= L1(pavé,vignettes[i])
      return np.argmin(L1_res)
```

IV.2 - Méthode sans restriction du choix des vignettes

Question 29 Écrire, à l'aide de ce qui précède, une fonction d'entête

```
def construireMosaïque(source:image, vignettes:[image], p:int) -> image:
```

qui construit une photomosaïque homothétique de `source` comportant `p` vignettes par côté.

Une fonction renvoyant une photomosaïque homothétique de `source` comportant `p` vignettes par côté, peut s'écrire comme suit :

```
def construireMosaïque(source, vignettes, p)
   h, w = vignettes[0].shape
   m_f = np.empty((h*p,w*p),np.uint8)
   m_i = initMosaïque(source,w,h,p)
   for i in range(0,h*p,h)
     for j in range(0,w*p,w)
       m_f(i:i+h,j:j+w) = vignettes[choixVignette(m_i(i:i+h,j:j+w),vignettes)]
   return m_f
```

Union des Professeurs de Sciences et Techniques Industrielles
© UPSTI - Toute reproduction interdite sans l'autorisation de l'UPSTI

Question 30 Déterminer sa complexité temporelle asymptotique 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`.

La complexité temporelle asymptotique de la fonction `construireMosaïque` est de :

CAT_construireMosaïque = CAT_initMosaïque + O(r) * CAT_choixVignette

or CAT_initMosaïque = CAT_tableSommation + CAT_réductionSommation2,

et CAT_choixVignettes = O(q)

De plus, CAT_tableSommation = O(N) = O(n*r), et CAT_réductionSommation2 = O(1)

Finalement, on a :

CAT_construireMosaïque = O(n*r) + O(1) + O(r*q)

CAT_construireMosaïque = O(r*(n+q))

IV.3 - Améliorations

Cette sous-partie demande de l'initiative de la part du candidat, qui peut 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 illustrer, le cas échéant, par un schéma. Toute démarche pertinente, même non aboutie, sera valorisée. Le barème prend en compte le temps nécessaire à la résolution de cette sous-partie.

La méthode sans restriction proposée précédemment peut conduire à sélectionner répétitivement 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 31 Proposer une stratégie de construction de photomosaïque 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.

Pour sélectionner un maximum de vignettes différentes, on peut définir des groupes de vignettes qui minimisent la fonction L1 qui définit la distance entre deux images, puis lorsque l'on doit affecter une vignette sur la photomosaïque, prendre la vignette minimisant L1 sauf si un des proches voisin possède la même vignette, dans ce cas, piocher dans le groupe de vignettes minimisant L1.

On peut supprimer de la liste la vignette après l'avoir affectée dans la matrice de la mosaïque.

On peut modifier une autre vignette que celle minimisant la fonction L1 (augmenter la transparence, accentuer la luminosité) dans le cas où un des proches voisin possède la même vignette.

Question 32 Implanter cette stratégie sous la forme d'une fonction `belleMosaïque`, version améliorée de la fonction `construireMosaïque`, dont on définira les éventuels paramètres supplémentaires.

L'implémentation la plus simple est de supprimer de la liste des vignettes, la vignette utilisée dans la matrice finale.

© UPSTI - Toute reproduction interdite sans l'autorisation de l'UPSTI

```
def belleMosaïque(source, vignettes, p)
   h, w = vignettes[0].shape
   m_f = np.empty((h*p,w*p),np.uint8)
   m_i = initMosaïque(source,w,h,p)
   for i in range(0,h*p,h)
     for j in range(0,w*p,w)
       i_vignette = choixVignette(m_i(i:i+h,j:j+w),vignettes)
       m_f(i:i+h,j:j+w) = vignettes[i_vignette]
       vignettes = vignettes.pop(i_vignette)
   return m_f
```