CONCOURS CENTRALE•SUPÉLEC

Informatique

MP, PC, PSI, TSI

3 heures Calculatrice autorisée

2020

Photomosaïque

Une *photomosaïque* (figure 1) 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 composée de 1600 vignettes — de gauche à droite : l'image source[1], la photomosaïque et 16 vignettes[2] (détail du pied)

De nombreux paramètres régissent la construction d'une photomosaïque et la qualité du résultat :

— la structure du pavage utilisé (nombre, forme et arrangement des vignettes) ;

— le nombre et la diversité des images de la banque ;

— les algorithmes mis en œuvre pour :

- sélectionner les bonnes images dans la banque (partie III) ;
- redimensionner les images (partie II) ;
- placer les vignettes (partie IV).

Dans la suite du sujet, les photomosaïques sont construites sur des pavages rectangulaires réguliers, c'est-à-dire, constitués de vignettes rectangulaires, toutes de mêmes dimensions et juxtaposées bord à bord.

Les seuls langages de programmation autorisés dans cette épreuve sont Python et SQL. Pour répondre à une question, il est possible de faire appel aux fonctions définies dans les questions précédentes. Dans tout le sujet, on suppose que les modules `math`, `numpy`, `matplotlib.pyplot` et `random` ont été rendus accessibles grâce à l'instruction

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

Si les candidats font appel à des fonctions d'autres bibliothèques, ils doivent préciser les instructions d'importation correspondantes.

Dans tout le sujet, le terme « liste » appliqué à un objet Python signifie qu'il s'agit d'une variable de type `list`. Les termes « vecteur » et « tableau » désignent des objets numpy de type `np.ndarray`, respectivement à une dimension ou de dimension quelconque. Enfin le terme « séquence » représente une suite itérable et indiçable, indépendamment de son type Python, ainsi un tuple d'entiers, une liste d'entiers et un vecteur d'entiers sont tous trois des « séquences d'entiers ».

[1] Photo par « Sincerely Media », issue de `https://unsplash.com`.

[2] Vignettes issues de la banque `https://picsum.photos/images`.

Les entêtes des fonctions demandées sont annotés pour préciser les types des paramètres et du résultat. Ainsi,

```
def uneFonction(n:int, X:[float], c:str, u) -> np.ndarray:
```

signifie que la fonction `uneFonction` prend quatre paramètres `n`, `X`, `c` et `u`, où `n` est un entier, `X` une liste de nombres à virgule flottante, `c` une chaine de caractères et le type de `u` n'est pas précisé. Cette fonction renvoie un tableau numpy.

Il n'est pas demandé aux candidats d'annoter leurs fonctions, la rédaction pourra commencer par

```
def uneFonction(n, X, c, u):
    ...
```

De façon générale, une attention particulière sera portée à la lisibilité, la simplicité et la clarté du code proposé. L'utilisation d'identifiants significatifs, l'emploi judicieux de commentaires seront appréciés.

Une liste de fonctions potentiellement utiles est fournie à la fin du sujet.

I Pixels et images

I.A – 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.

Q 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 ?

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.

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

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`.

Q 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` ?

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.

Q 4. Écrire une fonction d'entête

```
def gris(p:pixel) -> np.uint8:
```

qui calcule le niveau de gris correspondant au pixel `p`.

I.B – 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.

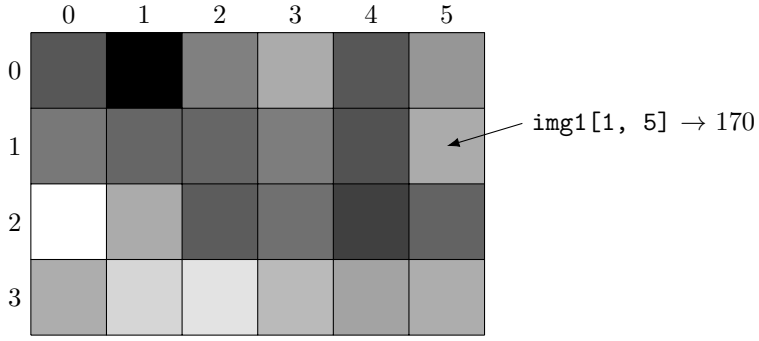

Figure 2 Visualisation de l'image `img1`

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)`.

Q 5. Interpréter ces valeurs.

Q 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`.

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.

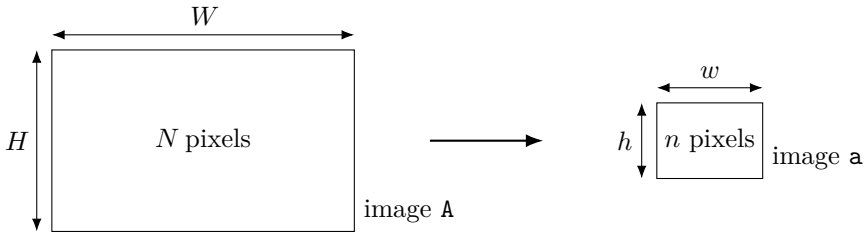

Figure 3 Redimensionnement d'image

II.A – 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 $40 \times 40 = 1600$ 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.

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

II.B – 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.

Q 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` × `h` en utilisant l'interpolation au plus proche voisin.

Q 9. Quelle est sa complexité temporelle asymptotique ?

II.C – 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
```

Q 10. Expliquer en quelques lignes son principe de fonctionnement.

Q 11. Donner sa complexité temporelle asymptotique.

II.D – 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.

Q 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.

Q 13. Écrire une fonction, de complexité temporelle asymptotique $O(N)$, d'entête

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

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

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
```

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

Q 15. Donner sa complexité temporelle asymptotique.

Q 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())
```

Q 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 ?

II.E – Synthèse

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

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.

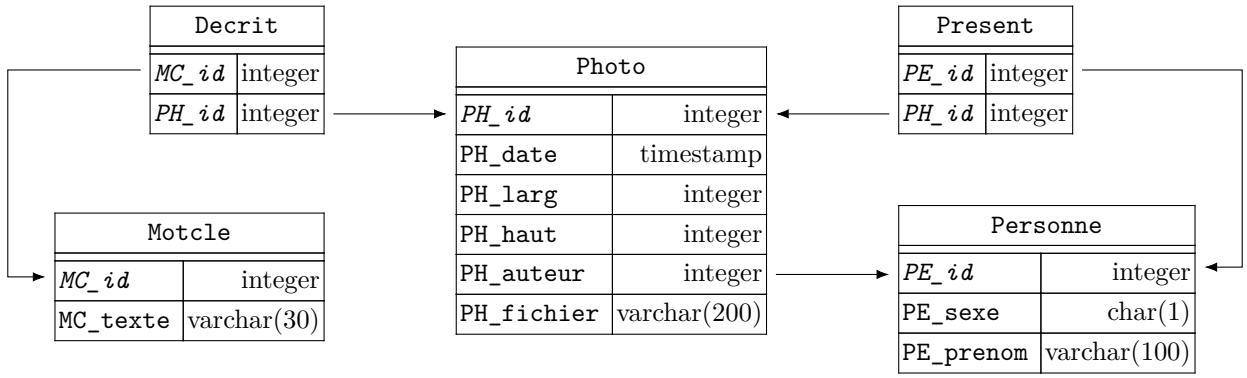

Figure 4 Structure physique de la base de données de photographies.

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.A – 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.

Q 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.

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

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

Q 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.

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

III.B – 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`.

Q 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).

Q 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.

IV Placement des vignettes

IV.A – 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 $r = p^2$.

Q 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.

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 vignettes et les pavés, on définit la distance L_1 entre deux images a et b de même taille $w \times h$ par :

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

Q 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.

Q 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.

IV.B – Méthode sans restriction du choix des vignettes

Q 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.

Q 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`.

IV.C – 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.

Q 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.

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

Opérations et fonctions disponibles en Python et SQL

Fonctions Python diverses

— `range(n)` itérateur sur les `n` premiers entiers ($[\![0, n-1]\!]$).
`list(range(5))` → `[0, 1, 2, 3, 4]`.

— `range(d, f, p)` où `d`, `f` et `p` sont des entiers, itérateur sur les entiers $(r_i = d + ip \mid r_i < f)_{i\in\mathbb{N}}$ si $p > 0$ et $(r_i = d + ip \mid r_i > f)_{i\in\mathbb{N}}$ si $p < 0$. Le paramètre `p` est optionnel avec une valeur par défaut de 1.
`list(range(1, 5))` → `[1, 2, 3, 4]` ; `list(range(20, 10, -2))` → `[20, 18, 16, 14, 12]`.

— `s[d:f:p]` où `s` est une séquence et `d`, `f` et `p` sont des entiers, désigne la séquence des éléments de `s` dont les indices correspondent à `range(d, f, p)`. Si `s` est d'un type de base (liste ou tuple), `s[d:f:p]` effectue une copie, si `s` est un tableau numpy, `s[d:f:p]` est une vue sur les éléments de `s` et peut être utilisé pour modifier `s`.
`[0, 1, 2, 3, 4, 5][2:6:2]` → `[2, 4]` ; `(0, 1, 2, 3, 4, 5)[5:2:-2]` → `(5, 3)`.

— `random.randrange(a, b)` renvoie un entier aléatoire compris entre `a` et `b-1` inclus (`a` et `b` entiers).

— `random.random()` renvoie un nombre flottant tiré aléatoirement dans $[0, 1[$ suivant une distribution uniforme.

— `random.choice(s)` renvoie un élément pris au hasard dans la séquence non vide `s`.

— `random.shuffle(L)` permute aléatoirement les éléments de la liste `L` (modifie `L`).

— `random.sample(s, n)` renvoie une liste constituée de `n` éléments distincts de la séquence `s` choisis aléatoirement, si `n` $\leqslant$ `len(s)` lève l'exception `ValueError`.

— `math.sqrt(x)` calcule la racine carrée du nombre `x`.

— `round(n)` arrondit le nombre `n` à l'entier le plus proche. Le résultat est de type `int` pour les types numériques de base. Pour les types de la bibliothèque numpy, le résultat a le même type que l'argument.

— `math.floor(x)` renvoie le plus grand entier inférieur ou égal à `x`.

— `math.ceil(x)` renvoie le plus petit entier supérieur ou égal à `x`.

Opérations sur les listes

— `len(L)` donne le nombre d'éléments de la liste `L`.

— `L1 + L2` construit une liste constituée de la concaténation des listes `L1` et `L2`.

— `n * L` construit une liste constituée de la liste `L` concaténée `n` fois avec elle-même.

— `e in L` et `e not in L` déterminent si l'objet `e` figure dans la liste `L`. Cette opération a une complexité temporelle en $O($`len`$(L))$.
`2 in [1, 2, 3]` → `True` ; `2 not in [1, 2, 3]` → `False`.

— `L.append(e)` ajoute l'élément `e` à la fin de la liste `L`.

— `L.pop(i)` : renvoie l'élément à l'indice `i` de la liste `L` et le supprime de la liste.

— `L.remove(e)` supprime de la liste `L` le premier élément qui a pour valeur `e`, s'il existe. Cette opération a une complexité temporelle en $O($`len`$(L))$.

— `L.insert(i, e)` insère l'élément `e` à la position d'indice `i` dans la liste `L` (en décalant les éléments suivants) ; si `i >= len(L)`, `e` est ajouté en fin de liste.

— `L.sort()` trie en place la liste `L` (qui est donc modifiée) en réordonnant ses éléments dans l'ordre croissant.

Opérations sur les tableaux (`np.ndarray`)

— `np.array(s, dtype)` crée un nouveau tableau contenant les éléments de la séquence `s`. La taille de ce tableau est déduite du contenu de `s`. Le paramètre `dtype` précise le type des éléments du tableau créé.

— `np.empty(n, dtype)`, `np.empty((n, m), dtype)` crée respectivement un tableau à une dimension de `n` éléments et un tableau à `n` lignes et `m` colonnes dont les éléments, de valeurs indéterminées, sont de type `dtype`. Si le paramètre `dtype` n'est pas précisé, il prend la valeur `float`.

— `np.zeros(n, dtype)`, `np.zeros((n, m), dtype)` fonctionne comme `np.empty` en initialisant chaque élément à la valeur zéro pour les types numériques ou `False` pour les types booléens.

— `np.full(n, v, dtype)`, `np.full((n, m), v, dtype)` fonctionne comme `np.empty` en initialisant chaque élément à la valeur `v`.

— `a.ndim` nombre de dimensions du tableau `a`.

— `a.shape` tuple donnant la taille du tableau `a` pour chacune de ses dimensions.

— `len(a)` taille du tableau `a` dans sa première dimension, équivalent à `a.shape[0]`.

— `a.size` nombre total d'éléments du tableau `a`.

— `a.dtype` type des éléments du tableau `a`.

— `a.flat` itérateur sur tous les éléments du tableau `a`.

— `np.ndenumerate(a)` itérateur sur tous les couples `(ind, v)` du tableau `a` où `ind` est un tuple de `a.ndim` entiers donnant les indices de l'élément `v`.

— `a.min()`, `a.max()` renvoie la valeur du plus petit (respectivement plus grand) élément du tableau `a` ; ces opérations ont une complexité temporelle en $O($`a.size`$)$.

— `a.sum()` ou `np.sum(a)` calcule la somme de tous les éléments du tableau `a` ; cette opération a une complexité temporelle en $O($`a.size`$)$.

— `a.sum(d)` ou `np.sum(a, d)` effectue la somme des éléments du tableau `a` suivant la dimension `d` ; le résultat est un nouveau tableau avec une dimension de moins que `a`.
`a.sum(0)` → somme par ligne, `a.sum(1)` → somme par colonne, etc.

— `a.mean()` ou `np.mean(a)` renvoie la valeur moyenne de tous les éléments du tableau `a` ; le résultat est de type `np.float64`. Cette opération a une complexité temporelle en $O($`a.size`$)$.

— `a.mean(d)` ou `np.mean(a, d)` effectue la moyenne des éléments du tableau `a` suivant la dimension `d` ; le résultat est un nouveau tableau avec une dimension de moins que `a`.
`a.mean(0)` → moyenne par ligne, `a.mean(1)` → moyenne par colonne, etc.

— `a.round()`, `np.around(a)` crée un nouveau tableau de même forme et type que `a` en arrondissant ses éléments à l'entier le plus proche.

SQL

— `T1 JOIN T2 USING (c1, c2, …)` joint les deux tables `T1` et `T2` sur les colonnes `c1`, `c2`… qui doivent exister dans les deux tables ; équivalent à `T1 JOIN T2 ON T1.c1 = T2.c1 AND T1.c2 = T2.c2 AND …`, sauf que les colonnes `c1`, `c2`… n'apparaissent qu'une fois dans le résultat.

— Les requêtes
- (SELECT ... FROM ... WHERE ...) INTERSECT (SELECT ... FROM ... WHERE ...)
- (SELECT ... FROM ... WHERE ...) UNION (SELECT ... FROM ... WHERE ...)
- (SELECT ... FROM ... WHERE ...) EXCEPT (SELECT ... FROM ... WHERE ...)

sélectionnent respectivement l'intersection, l'union et la différence des résultats des deux requêtes, qui doivent être compatibles : même nombre de colonnes et mêmes types.

— `EXTRACT(`*part* `FROM t)` extrait un élément de `t`, expression de type `date`, `time`, `timestamp` (jour et heure) ou `interval` (durée). *part* peut prendre les valeurs `year`, `month`, `day` (jour dans le mois), `doy` (jour dans l'année), `dow` (jour de la semaine), `hour`, etc.

— Les fonctions d'agrégation `SUM(e)`, `AVG(e)`, `MAX(e)`, `MIN(e)`, `COUNT(e)`, `COUNT(*)` calculent respectivement la somme, la moyenne arithmétique, le maximum, le minium, le nombre de valeurs non nulles de l'expression `e` et le nombre de lignes pour chaque groupe de lignes défini par la cause `GROUP BY`. Si la requête ne comporte pas de clause `GROUP BY` le calcul est effectué pour l'ensemble des lignes sélectionnées par la requête.

• • • FIN • • •

Informatique

Présentation du sujet

Le sujet est construit autour d'un des thèmes du programmes de seconde année, le traitement des images. Il s'intéresse à la mise en œuvre de méthodes numériques visant à concevoir des photomosaïques, images composées à la manière d'une mosaïque d'une multitude de petites images appelées vignettes. Le sujet comporte 32 questions réparties sur 4 parties et fait largement appel aux connaissances algorithmiques et pratiques du programme de première année :

- la première partie traite du codage des images en termes de pixels et de codage RGB pour se terminer par l'écriture d'une fonction de conversion d'une image en niveaux de gris ;
- la deuxième partie étudie plusieurs solutions algorithmiques de redimensionnement d'images de complexités temporelles différentes. La partie se termine par une synthèse discutant des usages respectifs de ces solutions ;
- la troisième partie aborde le thème des bases des données par l'écriture de requêtes sélectionnant une image source et des vignettes ;
- la quatrième partie aboutit à la construction d'une photomosaïque. Les deux dernières questions laissent une part importante à l'initiative des candidats.

Outre la maitrise des connaissances informatiques du programme, l'écriture syntaxiquement correcte de codes et l'analyse de leurs performances, le sujet évalue l'aptitude des candidats à porter un regard critique sur des propositions de codes. Ce sujet a très largement permis au jury d'évaluer la qualité et le niveau de compétences de chaque candidat.

Analyse globale des résultats

Ce sujet quelque peu original nécessite de s'approprier correctement la problématique : des indications sont fournies dans l'énoncé ou dans l'annexe en fin de sujet. Un nombre relativement élevé de copies très faibles a été constaté cette année, pour les candidats n'ayant pas fait ce travail d'appropriation. À contrario, quand les principes sont correctement appréhendés, le sujet ne présente pas de difficultés insurmontables ; des notes très correctes étaient à la portée des candidats rigoureux, les meilleurs ayant résolu de façon satisfaisante 90 % des questions.

La partie base de données représente un quart du barème. Là encore, à condition de connaître la syntaxe de base du langage SQL, de nombreuses questions sont très accessibles et ont été dans l'ensemble correctement abordées.

Enfin, trois questions demandent d'expliquer le principe de fonctionnement de quelques lignes de code. Les bons candidats se reconnaissent à la concision avec laquelle ils accomplissent cette tâche. À l'opposé, certains se perdent en de longs commentaires ligne à ligne du code, sans convaincre le correcteur.

En résumé, ce problème, plus axé sur la compréhension que sur la technique de programmation, a perturbé les candidats les plus fragiles, mais a permis, par la diversité des concepts abordés, de mettre en exergue les candidats les mieux préparés.

Commentaires sur les réponses apportées et conseils aux futurs candidats

L'épreuve d'informatique s'appuie sur un corpus assez restreint. Au cours de leurs années de préparation, les candidats doivent s'être suffisamment approprié la syntaxe de base des langages au programme. À cette condition, mais à cette condition seulement, une lecture attentive du texte permet de résoudre une bonne partie des questions. Les mauvaises prestations sont en général le fait de candidats qui n'ont pas cet acquis. Il est difficile de juger une requête SQL qui ne contient pas au moins la structure `SELECT ... FROM ... WHERE ...`, ou une fonction écrite en Python sans indentation.

Passée cette exigence, la qualité de la copie est corrélée à sa concision. Plus les réponses sont courtes, que ce soit pour un code ou pour une explication, plus la réponse est lisible, pertinente, et récompensée. Les candidats sont invités à prendre conscience de ce fait et à se préparer dans cet esprit.

La présentation est aussi un facteur important. Le jury est conscient que la programmation sur la feuille est un exercice un peu artificiel, mais cela n'excuse pas les trop nombreuses ratures de certaines compositions, qui rendent le contenu difficilement évaluable.

Au final, le jury est enclin à être bienveillant envers les petites erreurs de syntaxe, tant que le sens de la réponse reste perceptible. Des points transversaux représentant une partie significative de la note, sont attribués en fin de correction pour récompenser la présentation de la copie (y compris l'ordre des questions), la concision et la clarté, la présence de commentaires pertinents, le respect de la syntaxe.

I Pixels et images

La première partie aborde la représentation en mémoire des images. Beaucoup de candidats ont été perturbés par les entiers non signés sur 8 bits utilisés en traitement d'images. Les tableaux en deux (images en niveaux de gris) ou trois dimensions (images en couleurs) sont mieux maitrisés.

II Redimensionnement d'images

La seconde partie étudie trois algorithmes de changement de résolution d'image, de complexités et de rendus différents. Les meilleurs candidats ont parfaitement perçu les avantages et inconvénients respectifs de chacun d'eux. De nombreuses confusions ont été commises dans l'expression des complexités asymptotiques : dire que la complexité est quadratique n'a de sens que si l'on précise la taille des données. Autre erreur fréquente : une fonction telle que `np.sum(a)` n'est pas `O(1)` comme trop souvent évoqué. De plus, comme déjà mentionné, décrire le fonctionnement d'un segment de code ne consiste pas à énumérer le rôle de chaque ligne, mais demande de caractériser le résultat obtenu en fonction des variables d'entrée, et le moyen d'y arriver. Cela nécessite de prendre un peu de recul.

III Sélection des images de la banque

La partie base de données a une importance relative élevée. Les candidats maitrisant la syntaxe s'en sont en général correctement sortis. Les dernières questions demandent de modifier la structure existante de la table. Ceux qui s'y sont essayé ont pour beaucoup proposé des solutions satisfaisantes.

IV Placement des vignettes

La dernière partie détaille les étapes de construction d'une photomosaïque. De petites fonctions indépendantes sont demandées, globalement réussies par les candidats qui s'étaient imprégnés des parties précédentes. Les dernières questions demandent de proposer une amélioration de l'algorithme. Seules les réponses concises et en rapport avec le cahier des charges ont été validées. De la même façon, quelques fonctions courtes au rôle facilement identifiable sont préférables à un programme d'une page dont on ne perçoit ni l'intérêt ni la structure.

Conclusion

De par la structure du problème et peut-être également à cause d'une préparation perturbée, la correction a fait ressortir cette année un nombre plus important de candidats en difficulté avec la matière. Pour les autres, les différentes parties ont permis à chacun de s'exprimer en fonction de ses compétences.

Informatique

Présentation du sujet

Le sujet est construit autour d'un des thèmes du programmes de seconde année, le traitement des images. Il s'intéresse à la mise en œuvre de méthodes numériques visant à concevoir des photomosaïques, images composées à la manière d'une mosaïque d'une multitude de petites images appelées vignettes. Le sujet comporte 32 questions réparties sur 4 parties et fait largement appel aux connaissances algorithmiques et pratiques du programme de première année :

— la première partie traite du codage des images en termes de pixels et de codage RGB pour se terminer par l'écriture d'une fonction de conversion d'une image en niveaux de gris ;

— la deuxième partie étudie plusieurs solutions algorithmiques de redimensionnement d'images de complexités temporelles différentes. La partie se termine par une synthèse discutant des usages respectifs de ces solutions ;

— la troisième partie aborde le thème des bases des données par l'écriture de requêtes sélectionnant une image source et des vignettes ;

— la quatrième partie aboutit à la construction d'une photomosaïque. Les deux dernières questions laissent une part importante à l'initiative des candidats.

Outre la maitrise des connaissances informatiques du programme, l'écriture syntaxiquement correcte de codes et l'analyse de leurs performances, le sujet évalue l'aptitude des candidats à porter un regard critique sur des propositions de codes. Ce sujet a très largement permis au jury d'évaluer la qualité et le niveau de compétences de chaque candidat.

Analyse globale des résultats

Au regard de la longueur et de la difficulté de l'épreuve, le jury est satisfait du niveau général des copies. Les connaissances informatiques semblent globalement acquises, les langages Python et SQL convenablement maitrisés. Quelques rares candidats ont visiblement négligé la formation, tentant de répondre aux questions ne relevant pas immédiatement du domaine de l'informatique. Ces copies conduisent à des notes généralement très faibles.

La moitié des candidats de la filière PC traite pratiquement 65 % des questions. Un très faible pourcentage ne traite que moins de 20 % des questions.

De nombreux candidats ont fourni des copies d'excellente qualité. Le jury regrette le niveau parfois très bas d'autres copies. Il serait souhaitable que les candidats mesurent l'importance de la formation initiale en informatique pour la suite de leurs études mais également pour leurs cultures d'ingénieur et de citoyen.

Commentaires sur les réponses apportées et conseils aux futurs candidats

Les compétences en matière de programmation élémentaire semblent acquises chez le plus grand nombre de candidats. Néanmoins, le jury souhaite attirer l'attention des futurs candidats sur les points suivants.

— La notion de *type* est essentielle en informatique. Il convient d'en tenir compte lors de la manipulation d'objets informatiques. En particulier, les candidats doivent s'interroger sur la pertinence et les limites de certaines opérations effectuées sur ou entre objets de même type.

— La *complexité* est souvent estimée au regard du nombre d'opérations effectuées dans tout ou partie d'un code. Un minimum d'explications est attendu pour justifier le résultat qui doit, en outre, être exprimé

avec les notations strictes de l'énoncé. Ainsi, affirmer que la présence de deux boucles imbriquées induit une complexité quadratique, souvent notée $O(n^2)$ sans préciser la nature de n, est insuffisant. Ces questions attendent une argumentation fondée menant à l'écriture de complexités sous la forme, par exemple, $O(h \times w)$ puis $O(n)$ après avoir rappelé que $n = h \times w$ (cf. question 9).

— La lecture et l'*analyse de codes* comptent parmi les activités de tout futur ingénieur. Elles requièrent plus qu'un simple survol du code et plus encore qu'un commentaire de type *paraphrase*. Il convient d'abord de préciser le rôle et les objectifs d'une fonction ou d'un bout de code puis d'identifier des blocs structurels importants du code et d'en expliquer leur fonction.

— Les *requêtes SQL* doivent faire l'objet d'une attention particulière. Il s'agit de répondre exactement à la question, sans oublier d'attributs dans la réponse, sans oublier les jointures, etc. Si les candidats maitrisent l'écriture de requêtes très élémentaires, de nombreuses réponses sont souvent incomplètes, voire mal écrites, en raison d'une lecture incomplète ou erronée des questions.

— Les codes sont globalement syntaxiquement corrects et lisibles. Leur lecture révèle toutefois une écriture au fil de l'eau. Il serait souhaitable qu'avant même d'écrire une fonction, chaque candidat s'interroge sur l'*organisation* et la *structure logiques des codes* qu'il propose. À cela s'ajoute la présence de commentaires parfois inutiles dans le corps du code. Un commentaire n'a de sens que s'il apporte une information utile et pertinente pour comprendre le code. Les docstrings (documentation placée immédiatement après la définition d'une fonction) sont toujours utiles en pratique mais généralement pas attendues sur une copie dans le cadre d'une épreuve de concours en temps limité.

— Des points dits *transversaux* ont été attribués pour valoriser la clarté des explications, la qualité rédactionnelle, le respect de la syntaxe de Python, la lisibilité des codes et les commentaires pertinents. Le jury est particulièrement attentif à ces compétences transversales.

Signalons par ailleurs quelques erreurs de syntaxe générales :

— écriture *à l'envers* des affectations, `10 = a` ;

— mauvaise gestion des `range` en ajoutant 1 à la valeur finale pour parcourir toute la liste ;

— l'incrémentation avec `+=` devient parfois `±` ;

— le symbole `*` de la multiplication est souvent omis.

I Pixels et images

Q1. Cette première question a mené à des réponses de qualité variable. Près d'une fois sur deux, le décompte du nombre de couleurs est incorrect et l'application numérique simple est souvent omise.

Q2. De nombreuses réponses sont erronées en raison essentiellement d'un manque de rigueur. La réponse attendait d'une part un objet de type clairement défini, d'autre part une proposition qui respecte les contraintes liées au codage RGB.

Q3. Très peu de candidat ont traité convenablement cette question. La principale erreur est liée à l'absence de prise en compte du type des objets manipulés, conduisant inévitablement à des erreurs dans les calculs demandés. Beaucoup de copies tentent d'écrire les opérations sous forme binaire.

Q4. Cette question est globalement bien traitée. Une attention particulière doit, là encore, être portée sur le type du résultat renvoyé. Beaucoup d'erreurs sont liées à un manque de rigueur dans le suivi des consignes. L'énoncé demandait *la meilleure approximation entière* (qui n'est ni la partie entière, ni le quotient euclidien d'une division) d'une moyenne, retournée sous le type `np.uint8`.

Q5. Cette question est comprise par l'ensemble des candidats mais les réponses sont souvent imprécises ou incomplètes. Deux points structuraient la réponse : un premier point détaillait le contenu de `source.shape`, un second point précisait la signification et le contenu de `source[0,0]`. La rédaction est souvent trop vague. Parler de *largeur* et de *longueur* d'une image est ambigu.

Q6. Cette question de codage est bien réussie par les candidats. Certaines réponses utilisant `a.shape` oublient parfois qu'en raison de la nature même de `a`, cette instruction renvoie un triplet. Le type des éléments du nouveau tableau renvoyé par la fonction est parfois oublié. Enfin, l'énoncé demandant de retourner une image en niveaux de gris qui est un tableau à deux dimensions, il est incorrect de modifier le tableau `a` passé en argument.

II Redimensionnement d'images

Q7. Dans cette question globalement comprise par les candidats, les explications sont parfois confuses même si les résultats sont corrects. Le jury est attentif à la qualité rédactionnelle et aux explications fournies. Même s'il fait preuve de bienveillance, les réponses succinctes, de type *steno*, sont pénalisées.

Q8. Cette question est globalement bien traitée. Une erreur récurrente est observée dans les arguments calculés qui permettent de sélectionner les éléments du tableau `A`. Certaines réponses ont tendance à utiliser `W` et `H` sans les définir dans la fonction. Des confusions sont également faites dans l'utilisation des dimensions `w` et `h`, parfois interverties.

Q9. Bien qu'à priori relativement simple, cette question a révélé la difficulté de nombreux candidats à argumenter leurs calculs de complexité. Une telle question attend une réponse détaillée : opérations prises en compte, décomptes du nombre de ces opérations, expressions du résultat final en respectant les notations strictes de l'énoncé. La seule réponse à la question ne suffit pas à obtenir tous les points.

Q10. Trop souvent l'explication se résume à une simple paraphrase des lignes du code, sans montrer une compréhension de celles-ci. Parler des valeurs « autour de `A[i,j]` » est bien trop vague. L'exercice de lecture et d'analyse d'un code attend bien évidemment plus qu'une lecture ligne à ligne. Comme le signale le début de ce rapport, le rôle du code analysé doit être précisé. S'agissant dans le cas présent d'une fonction, étant données des informations en entrée, le résultat renvoyé et son type doivent être indiqués. Ensuite, il convient d'identifier les blocs structurels importants et les étapes clés des calculs effectués dans le corps de la fonction. Ainsi, par son argumentation, le candidat montre sa capacité à prendre du recul par rapport à la question.

Q11. Comme pour la question 9, cette question a révélé les difficultés d'une présentation claire et rigoureuse des nombres d'opérations menant à l'établissement d'une complexité temporelle. Cette question est peu réussie.

Q12. Cette question a été soit très bien réussie, soit pas réussie du tout ou non traitée. Elle s'appuie sur des connaissances de cours simples.

Q13. Cette question plus délicate nécessitait une réflexion préalable à l'écriture de la fonction. Malheureusement, ce travail de préparation, trop souvent clairement absent au vu des productions, a abouti à l'écriture de fonctions incomplètes ou fausses. Quelques trop rares copies ont proposé de bonnes solutions. Une lecture attentive de l'énoncé aurait également pu éviter certaines erreurs comme par exemple le dimensionnement incorrect `H*W` d'un tableau alors qu'il était attendu `(H+1)*(W+1)`.

Q14. Cette question a fait l'objet d'un traitement variable. Les réponses présentent les mêmes défauts que ceux énoncés pour la question 10.

Q15. Cette question amène les mêmes commentaires que ceux des questions 9 et 11.

Q16. Cette question amène les mêmes commentaires que ceux de la question 10.

Q17. Quand elle traitée, cette question est peu réussie. Elle nécessitait de prendre du recul par rapport à l'ensemble des questions de la deuxième partie.

Q18. Cette question marquant la fin de la deuxième partie, quelques candidats ont fourni des réponses partielles en discutant la qualité des images obtenues.

III Sélection des images de la banque

Q19. Cette question est globalement très bien traitée par l'ensemble des candidats.

Q20. Cette question est bien traitée par l'ensemble des candidats. De nombreuses copies utilisent `USING` proposé en annexe. Signalons malgré tout quelques erreurs liées à une mauvaise écriture de la jointure. Attention également aux erreurs de syntaxe comme l'utilisation de `==` dans la clause `WHERE`.

Q21. Cette question nécessitait l'écriture de deux jointures. À ce sujet, les réponses sont inégales. Un nombre non négligeable de candidats ne maitrise pas l'écriture de jointures multiples. Par ailleurs, certaines clauses `WHERE` de fin de requête sont maladroitement écrites, avec un `OR` parfois hasardeux. Plusieurs candidats ont pensé que `PH_auteur` était le prénom de l'auteur alors que le type `integer` est clairement mentionné dans la table `Photo`. Néanmoins, plus de la moitié des candidats apporte une réponse tout à fait satisfaisante à cette question.

Q22. Cette question nécessitait de jointer convenablement trois tables avec deux conditions entre les tables `Photo` et `Present` par exemple. Une erreur fréquente réside dans l'oubli d'une de ces deux conditions. L'extraction de l'année, expliquée dans l'annexe, est souvent fausse.

Q23. Cette question a été peu traitée. Une réponse pouvait être formulée en un `INTERSECT` avec un `EXCEPT` ou un `INTERSECT` avec un `COUNT`.

Q24. Cette question laissait une place importante à la prise d'initiative. Néanmoins, la réponse proposée devait respecter le cahier des charges exprimé par les points a. et b. de l'énoncé, ce qui a conduit à des réponses parfois incomplètes.

Q25. Suite naturelle de la question 24, cette question est convenablement traitée par moins de la moitié des candidats. Peu de copies proposent une réponse satisfaisante et rigoureuse.

IV Placement des vignettes

Q26. Le plus simple était d'utiliser convenablement la fonction `procheVoisin` définie à la question 8. Les arguments passés à cette fonction sont parfois incorrects. Le traitement global de cette question est donc très variable.

Q27. Cette question est peu réussie en raison essentiellement de la non prise en compte du dépassement de capacité. Les entiers utilisés n'étant pas signés, `abs(a-b)` est généralement différent de `abs(b-a)`. Une solution consistait à redéfinir les images `a` et `b` en tableaux de type `np.int64` ou bien à calculer des différences de la forme `int(a[i,j])-int(b[i,j])`.

Q28. Cette question a globalement été réussie.

Q29. Cette question abordée par beaucoup de candidats nécessitait l'écriture d'un code organisé, appelant des fonctions définies aux questions précédentes. Les réponses sont variables en raison d'un manque d'organisation préliminaire des idées qui aurait permis une écriture plus aisée du code.

Q30. Peu de candidats ont abordé cette question avec succès.

Q31. et **Q32**. Les deux dernières questions, relativement ouvertes, laissaient la part belle aux propositions des candidats. Si la question 31 a permis de lire quelques propositions pertinentes de stratégie, son implantation dans la question 32 n'a fait l'objet que de très rares réponses correctes et complètes.

Conclusion

Les résultats à cette épreuve montrent que les étudiants, soutenus par leurs professeurs, ont acquis des compétences certaines en informatique. Le jury encourage les futurs candidats à travailler l'informatique en alliant réflexion sur feuille de papier et mise en œuvre des algorithmes sur ordinateur.

Informatique

Présentation du sujet

Le sujet est construit autour d'un des thèmes du programmes de seconde année, le traitement des images. Il s'intéresse à la mise en œuvre de méthodes numériques visant à concevoir des photomosaïques, images composées à la manière d'une mosaïque d'une multitude de petites images appelées vignettes. Le sujet comporte 32 questions réparties sur 4 parties et fait largement appel aux connaissances algorithmiques et pratiques du programme de première année :

- la première partie traite du codage des images en termes de pixels et de codage RGB pour se terminer par l'écriture d'une fonction de conversion d'une image en niveaux de gris ;
- la deuxième partie étudie plusieurs solutions algorithmiques de redimensionnement d'images de complexités temporelles différentes. La partie se termine par une synthèse discutant des usages respectifs de ces solutions ;
- la troisième partie aborde le thème des bases des données par l'écriture de requêtes sélectionnant une image source et des vignettes ;
- la quatrième partie aboutit à la construction d'une photomosaïque. Les deux dernières questions laissent une part importante à l'initiative des candidats.

Outre la maitrise des connaissances informatiques du programme, l'écriture syntaxiquement correcte de codes et l'analyse de leurs performances, le sujet évalue l'aptitude des candidats à porter un regard critique sur des propositions de codes. Ce sujet a très largement permis au jury d'évaluer la qualité et le niveau de compétences de chaque candidat.

Analyse globale des résultats

Chacune des parties du sujet est progressive, ce qui a permis d'étaler les notes des candidats.

Les candidats ont pu montrer leurs compétences en informatique. Certaines compétences semblent assez bien maitrisées, comme par exemple la programmation en Python, qui ne pose pas de problème à la majorité d'entre eux. D'autres sont plus discriminantes, comme la compréhension de la complexité, qui a permis à certains candidats de tirer leur épingle du jeu.

La partie sur les bases de données a été globalement bien réussie, voire mieux réussie que les autres parties. Pour 14 % des candidats, cette partie représente même la majorité des points. Le jury recommande aux candidats d'utiliser la liberté qu'offre SQL de sauter des lignes ou d'indenter pour rendre la structure de leur code plus facilement lisible.

Les questions d'analyse de programme et de compréhension ont permis de mesurer la capacité à prendre du recul face au problème posé. Les questions ouvertes ont permis d'évaluer la créativité des candidats et leur capacité à défricher un problème.

Le jury est globalement satisfait du niveau en informatique atteint par les candidats et encourage les futurs candidats à travailler cette matière importante dans le cursus d'un ingénieur.

Commentaires sur les réponses apportées et conseils aux futurs candidats

Programmation en Python

Le sujet comporte un certain nombre de questions de programmation en Python (**Q4**, **Q6**, **Q8**, **Q13**, **Q26**, **Q27**, **Q28**, **Q29**) où il est demandé aux candidats d'écrire une fonction satisfaisant une certaine spécification. Cette spécification contient plusieurs exigences (type d'entrée, type de sortie, ce que doit calculer la fonction, gestion des problèmes de débordements d'entiers, complexité...). Les candidats sont invités à faire attention à toutes ces exigences. Idéalement, la fonction écrite par le candidat les satisfait toutes.

Ces questions ont globalement été assez bien réussies. Certains candidats, n'arrivant pas à satisfaire toutes les exigences, ont parfois proposé des fonctions ne satisfaisant qu'une partie de ces exigences, que le jury a évaluées comme valant une partie des points de la question. Malheureusement, dans un certain nombre de copies, certaines exigences n'ont pas été satisfaites par simple négligence, par exemple en renvoyant un entier au lieu d'un `uint8`.

La plupart des candidats n'ont pas hésité à utiliser les fonctions précédemment définies, c'est une bonne chose. Quelques candidats préfèrent reprogrammer les fonctions précédentes au lieu de les utiliser. Le jury ne peut que décourager cette pratique qui est source d'erreurs et qui n'est pas une bonne manière de programmer.

Lorsque les candidats ont le choix entre plusieurs manières de programmer, ils ne doivent pas hésiter à préférer les outils qu'ils maitrisent, fussent-ils plus simples. Utiliser une fonctionnalité avancée de Python qu'on connait mal alors qu'une fonctionnalité plus basique peut convenir n'est pas une bonne stratégie.

Le jury a remarqué sur quelques copies des confusions entre le langage Python et le langage mathématique : les symboles comme la barre de fraction ou le symbole mathématique de la partie entière ne peuvent pas être utilisés en Python.

La dernière question du sujet (**Q32**) était une question de programmation ouverte, laissant les candidats exprimer leur créativité. Cette dernière question a permis de valoriser les meilleures copies.

SQL

Le sujet comporte un certain nombre de questions demandant d'écrire des requêtes SQL (**Q19**, **Q20**, **Q21**, **Q22**, **Q23**, **Q25**). Les premières questions de SQL ont été réussies par une large majorité de candidats. Ces questions, très progressives, ont permis de bien différencier les candidats selon leur maitrise du SQL.

Le jury attire l'attention des candidats sur l'importance de bien comprendre la nature des objets manipulés (valeurs, attributs, tables, ...).

Questions de compréhension

Le sujet comporte des questions de compréhension de code (**Q10** et **Q14**). Il ne s'agit ici évidemment pas de paraphraser le code, mais bien d'en extirper les idées principales pour en comprendre le fonctionnement.

Prise de recul

Certaines questions, demandent de prendre du recul par rapport au problème (**Q5**, **Q10**, **Q14**, **Q16**, **Q18**, **Q24**, **Q31**). Il est préférable de rédiger de manière claire et synthétique, sans diluer le propos et souvent un schéma est le bienvenu.

Complexité

Plusieurs questions portent sur la complexité (**Q9**, **Q11**, **Q15**, **Q17**, **Q30**). Le jury s'attend à une justification laconique de la complexité. Cette justification permet au jury d'évaluer la compréhension du problème de la complexité par le candidat, mais aide aussi grandement les candidats à clarifier leur pensée. En effet, la majorité des complexités qui ont été données dans les copies sans justification sont fausses.

Certaines copies ont oublié la complexité cachée due à des slicing ou à la fonction `mean`.

Remarques particulières sur certaines questions

Q1. C'est une question de dénombrement, 66 % des candidats ont donné la bonne formule. Lorsqu'une question demande un nombre, le jury s'attend à ce que l'application numérique soit effectuée, 52 % des candidats ont la bonne formule et ont pensé à faire l'application numérique.

Q2. Pour l'essentiel, les erreurs à cette question sont dues à une mauvaise lecture de l'énoncé, notamment à une confusion entre le pixel blanc demandé et le pixel noir donné en exemple. Le jury est satisfait que l'essentiel des candidats sache faire la différence entre une instruction et une fonction.

Q3. Que 280 ne soit pas représentable sur 8 bits a échappé à un certain nombre de candidats.

Q4. Il est demandé aux candidats d'écrire une fonction avec de nombreuses spécifications (meilleure approximation, `uint8`, etc.). 88 % des candidats ont écrit une fonction répondant à au moins une spécification, mais seuls 10 % ont répondu à toutes les spécifications.

Q13. La principale difficulté à cette question a été l'exigence d'une complexité en `O(N)`. La majorité des candidats a donné une solution satisfaisant au moins une partie des exigences, seule une minorité a obtenu la complexité en `O(N)`.

Q19. Cette première question de SQL permet de tester les compétences élémentaires des candidats dans ce langage. Elle a été traitée par 96 % des candidats et a été parfaitement réussie par 82 % des candidats. Moins de 1 % des candidats ayant traité cette question ont donné une réponse totalement fausse.

Q20 Cette question teste la capacité à écrire des jointures et à écrire une condition correcte (avec un « `OR` »). 30 % des candidats ont parfaitement réussi cette question, 83 % l'ont partiellement réussie. Les erreurs viennent en majorité d'une mauvaise écriture du `JOIN` ou d'une confusion entre `AND` et `OR` dans la condition du `WHERE`.

Q25 Les étudiants devaient utiliser la structure qu'ils ont eux-mêmes créée à la **Q24**, ce qui a permis de valoriser ceux qui ont proposé, à la question précédente, une structure simple et efficace par rapport à ceux qui ont choisi une structure complexe et difficile à utiliser.

Q27 Une des difficultés de l'énoncé était l'exigence : « en prenant garde aux dépassements de capacité ». Certains candidats ont purement et simplement ignoré cette contrainte.

Q31 et **Q32**. Le sujet termine par deux questions ouvertes, permettant d'évaluer la capacité des étudiants à réfléchir sur un problème qui n'a pas été déjà cadré. Ces questions ont été peu abordées, mais le jury est positivement surpris par la pertinence des réponses de certains candidats.

Conclusion

Le sujet aborde de nombreux points du programme. Les résultats sont globalement satisfaisants.

Le recul vis-à-vis du problème posé, la compréhension des objets manipulés, la clarté et la concision de l'expression sont autant d'atouts qui serviront les candidats en informatique et, plus généralement, dans leur carrière d'ingénieur.