Photomosaïque CCS 2020

I] Pixels et images

A) Pixels

1) Une composante RGB est codée sur 8 bits, elle peut donc prendre $2^8 = 256$ valeurs. On peut donc réaliser $2^8 \times 2^8 \times 2^8 =$ 16,8 millions de couleurs.

2) Un pixel blanc est designé par le triplet (255, 255, 255) soit : pixel_blanc = np.array([255, 255, 255], np.uint8).

3) 280 > 255 ; Il y a un dépassement de capacité donc
a = 280 − 256 = 24.
On a par contre b = 240

De même
- a+b = 8
- a−b = 40
- a//b = 0

} le résultat reste un uint8.

- a/b = 0.1 → le résultat est un floltant.

4)
```
def gris(p):
    niveauGris = round(np.sum(p)/3, 0)
    return np.uint8(niveauGris)
```

B) Images.

5) source.shape renvoie la taille du tableau numpy codant l'image, Ici, 3000 pixels de hauteur,

4000 pixels pour la largeur et chaque pixel contient les 3 composantes RGB.

source[0,0] renvoie la composante RGB du pixel du coin supérieur gauche.

6) On pourrait proposer une fonction "vectorisée" mais ici, l'énoncé suggère d'utiliser la fonction gris(). On peut donc proposer:

```
def conversion(a):
    hauteur, largeur, osef = a.shape
    imageGris = np.zeros((hauteur, largeur), np.uint8)
    for i in range(hauteur):
        for j in range(largeur):
            imageGris[i,j] = gris(a[i,j,:])
    return imageGris
```

II] Redimensionnement d'image

A] Le contexte.

7) La photomosaïque mesure 2m de large et contient 40 image par ligne. Chaque vignette mesure donc 5 cm de large. Leur résolution est de 10 pixels/mm. Soit 500 pixels de large.
La hauteur est donc de $5 \times \frac{3}{4}$ cm, soit 375 pixels.

La photomosaïque contient donc: $(500 \times 40) \times (375 \times 40)$
$= 20\,000 \times 15\,000$ pixels
$= 3.10^8$ pixels.

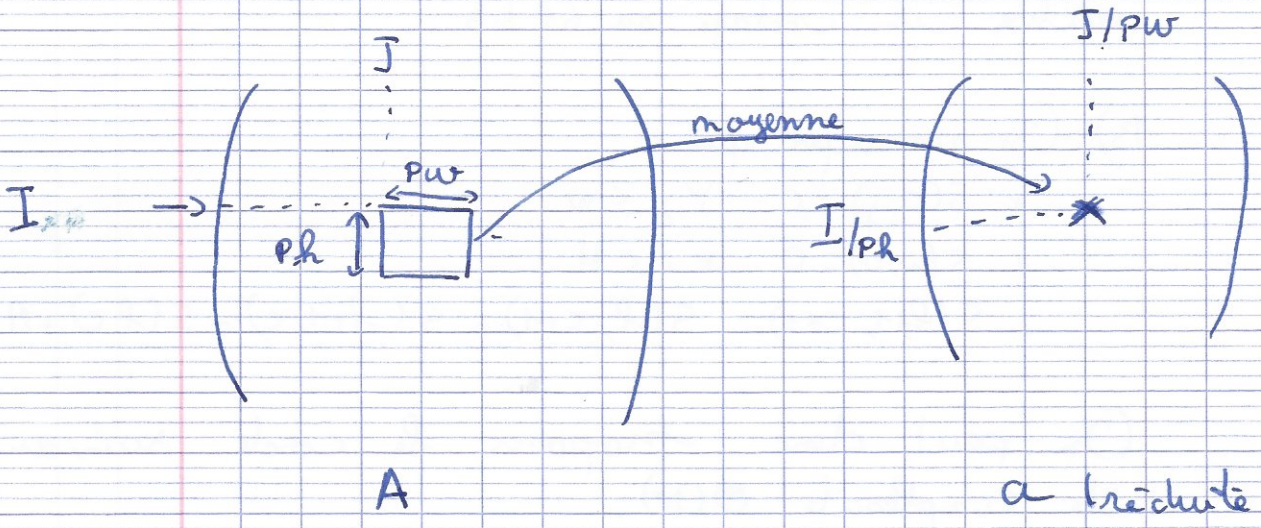
B] Algorithme d'interpolation au plus proche voisin.

8)
```
def procheVoisin(A, w, h):
    a = np.zeros((h, w), np.uint8)
    H, W = A.shape
    for i in range(h):
        for j in range(w):
            a[i, j] = A[math.floor(i*H/h), math.floor(j*W/w)]
    return a
```

9) On a un algorithme avec deux boucles for imbriquées.
La complexité est en $O(h, w)$

C) Algorithme de réduction par moyenne locale.

10) La fonction moyenne locale réduit l'image en une vignette de taille h, w. Chaque pixel de la vignette a pour valeur la valeur moyenne des H//h × W//w pixels de l'image original.



11) On a affaire à deux boucles for imbriquées, dans lesquelles on effectue une moyenne.
La fonction mean exécutée ph×pw valeurs de A, sa complexité est donc $\Theta(ph \times pw)$.
La complexité totale de l'algorithme est donc en $\Theta\left(\frac{H}{ph} \times \frac{W}{pw} \times ph \times pw\right) = \Theta(H \times W)$

Elle est plus élevée que la complexité de l'algorithme précédent.

D) Optimisation de la réduction par moyenne locale.

12) La plus grande valeur codable sur 32 bits est:
$2^{32} - 1 = 4,29 \cdot 10^9$.
Comparons cette valeur à la plus grande valeur que l'on puisse trouver dans S soit $255 \times 50 \cdot 10^6 = 1,28 \cdot 10^{10}$.

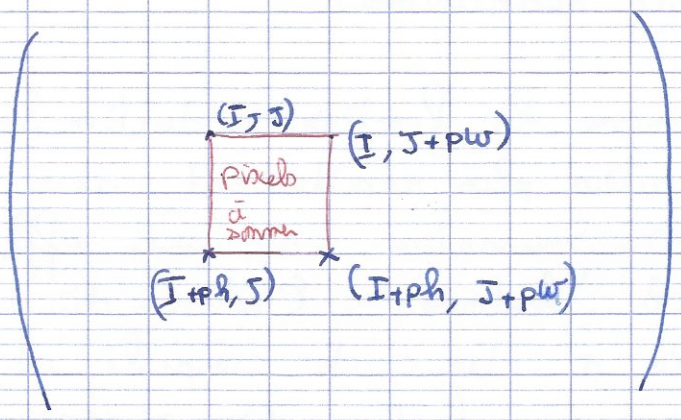
Le codage sur 32 bits n'est pas suffisant. On peut utiliser un codage sur 64 bits.

13) On remarque que :
* $S(0,j) = 0 \quad \forall j \leq w$
* $S(i,0) = 0 \quad \forall i \leq H$
* $S(l+1, c+1) = S(l, c+1) + S(l+1, c) - S(l,c) + A(l,c)$

On peut donc proposer le code suivant de complexité $\Theta(N)$:

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

14) X correspond à la somme des valeurs des pixels compris entre I et I+ph et entre J et J+pw :



$$A$$

On divise ensuite X par le nombre de pixels sommés, ce qui donne bien la valeur moyenne.

15) La complexité temporelle est en $\Theta(H/ph \times W/pw) = \Theta(hw)$.

16) Sred contient les éléments de S mais échantillonnés tous les ph et pw.
dc contient la différence Sred(i+1, j+1) - Sred(i, j)

ce qui correspond au terme S[I+ph, J+pw] - S[I+ph, J] de X.

De même dl correspond à X

Finalement d contient bien les mêmes éléments que a.

17) La fonction reductionSommation2 possède la même complexité temporelle que la fonction reductionSommation et une complexité spatiale un peu plus grande (création des tableaux sred, sd, dl) mais ce n'est pas rédhibitoire).

Néanmoins la fonction réduction Sommation 2 s'exécutera plus rapidement que la fonction réductionSommation car elle effectue des calculs "vectorisés". C'est-à-dire qu'elle fait appelle à du code compilé et non pas interprété. On gagne un facteur 100 à 1000 sur la rapidité d'exécution.

E) Synthèse

18) Pour réduire une image, la fonction procheVoisin s'exécutera rapidement mais fournira une image de moins bonne qualité que les deux autres fonctions. La fonction reductionSommation sera à privilégier car de complexité moindre que moyenneLocale.

III] Selection des images de la banque

A] Quelques requêtes

19) SELECT PH_id FROM Photo
WHERE PH_haut / PH_larg = 0.75 ;

calcul de hauteur/largeur plutôt que l'inverse pour eviter les problèmes d'arrondis.

20) SELECT COUNT(*) FROM Photo JOIN Personne
ON PE_prenom = PH_auteur
WHERE PE_prenom = 'Alice' OR PE_prenom = 'Bernard';

21) SELECT PH_id, PH_date
FROM Photo JOIN Decrit
ON PH_id = MC_id
JOIN Motcle
ON Decrit.Mc_id = Motcle_Mc_id
WHERE MC_texte = 'surf' and
EXTRACT(year FROM PH_date) < 2006 ;

22) SELECT PE_personne, PH_id FROM Photo JOIN Personne
ON PH_auteur = PE_id
JOIN Present ON PH_id = Personne_PE_id
WHERE Present.PE_id = PH_auteur AND Photo.PH_id = Present.PH_id;

23) On cherche les photos où apparaissent Alice et Bernard puis on enlève les photos où il y a une personne autre :

```
SELECT PH_id FROM
        SELECT PH_id FROM Present JOIN Personne ON
        Present_PE_id = Personne.PE_id WHERE PE_Prenom = 'Alice'
        INTERSECT
        SELECT PH_id FROM Present JOIN Personne ON
         PE_id   WHERE PE_prenom = "Bernard"
EXCEPT
        SELECT PH_id FROM Present JOIN Personne
        ON PE_id WHERE PE_prenom <> "Alice"
        AND PE_prenom <> "Bernard" ;
```

B) Internationalisation des mots-clés :

24) On peut modifier la table Motclé pour qu'elle serve de dictionnaire. Elle aurait comme attribut :

- MC_id
- lang : type varchar(30) qui donne la langue utilisée
- MC_Texte : type varchar(30) qui donne la traduction du mot clé dans la langue.

Clé primaire : (MC_id, lang). Pour toute valeur de MC_id, il doit y avoir une entrée par langue.

25) SELECT PH_id FROM Decrit JOIN Motcle ON Decrit.Mc_id = Motcle.MC_id
WHERE lang = "english" AND Mc_texte = "mountain";

IV] Placement des vignettes.

A) Préparatifs

26)
```
def initMosaique(source, w, h, p):
    S = conversion(source)
    return procheVoisin(S, p*w, p*h)
```

27)
```
def L1(a, b):
    h, w = a.shape        L = 0
    for i in range(h):
        for j in range(w):
            L = L + abs(a[i,j] - b[i,j])
    return L
```

28)
```
def choixVignette(pave, vignettes):
    i_min = 0
    min = L1(pave, vignettes[imin])
    for i in range(1, len(vignettes)):
        if L1(pave, vignettes[i]) < min:
            i_min = i
            min = L1(pave, vignettes[i])
    return indice
```

B) Méthode sans restriction du choix des vignettes.

29) def construireMosaique(source, vignettes, p):
    h, w = vignettes[0].shape
    Mosaique = initMosaique(source, w, h, p)
    for i in range(0, h*p, h):
        for j in range(0, w*p, w):
            Mosaique[i:i+h, j:j+h] =
            vignettes[choixVignette(Mosaique[j:i+h, j:j+w]
    return Mosaique

30) init_mosaique est de complexité $O(h \times w \times p^2) = O(n \times x)$
L1 est de complexité $O(h \times w) = O(n)$
choixVignette est dans de complexité $O(h \times w \times q) = O(nq)$

La fonction construire mosaique a donc une complexité en $O((n+q))$.

31) 32) : Questions ouvertes de fin de sujet. Les correcteurs sont généreux sur ce genre de questions n'hésitez pas à proposer une solution même si elle n'est pas idéale.