Mercredi 18 décembre 2024 — *Durée : 2 heures*

Devoir Surveillé n°1 — *Corrigé*

L'usage de la calculatrice est INTERDIT.

REMARQUES

- + + + Ne coupez pas une fonction ou une requête SQL sur deux pages.
- Comme dans toutes les matières, les réponses doivent être JUSTIFIÉES. On ne peut se contenter d'une réponse balancée sans explication.
- Python :
  - attention à l'**indentation**,
  - attention aux limites dans le `range` pour les parcours de listes :
    RAPPEL pour une liste `L` de longueur `n`, les rangs des éléments sont compris entre 0 et `n` EXCLUS, doit entre 0 et `n-1` inclus.
    Pour parcourir la TOTALITÉ d'une liste, il faut donc indiquer `range(0,n)` ou `range(n)` qui va donc parcourir les entiers de 0 à `n` EXCLUS, soit de 0 à `n-1` INCLUS.
  - les fonctions doivent être **commentées** clairement : variables, rôles des variables, explication du fonctionnement des différentes étapes (sans enfoncer les portes ouvertes !),
- SQL :
  - attention à la syntaxe des **jointures** :

```
SELECT
JOIN table1
JOIN table2
JOIN table3
ON table1.id1=table2.id1
AND table2.id2=table3.id2
```

  - attention à l'ambiguïté possible lors d'une jointure entre deux tables ayant des attributs de même nom,
  - attention à la syntaxe `nom_table.nom_attribut` (si ambiguïté lors d'une jointure),
  - il est inutile d'indiquer le nom de la table devant le nom de l'attribut quand il n'y a pas d'ambiguïté sur l'attribut (présent dans une seule table),
  - attention à l'utilisation de `WHERE / HAVING`.

On suppose avoir importé le module math de la façon suivante : `from math import *`

Partie I Traitement d'images

Dans ce sujet, on considère qu'une image en niveau de gris est une matrice I d'entiers compris entre 0 (pour noir) et 255 (pour blanc), on note h le nombre de lignes (c'est-à-dire : la hauteur de l'image) et w le nombre de colonnes (c'est-à-dire : la largeur de l'image). Cette matrice sera présentée par la liste `img` de ses lignes (une ligne étant une liste d'entiers). Ainsi pour $i \in [\![0, h-1]\!]$ et $j \in [\![0, w-1]\!]$ le pixel en position (i, j) de l'image, que l'on notera $I_{i,j}$ correspondra en Python à la commande `I[i][j]`.

R1. Quelle est la taille nécessaire en octets pour stocker une image de hauteur 1024 et de largeur 2124 ?

Solution: $256 = 2^8$, chaque pixel est donc codé sur 1 octet. Par conséquent pour l'image citée, il faut 1024×2124 octets, soit 2174976 octets.

R2. Écrire une fonction `imini(L:list)->int` qui prend en argument une liste `L` et qui renvoie la position de la première occurrence du minimum.
Par exemple `imini([531, 160, 50, 237, 50])` renverra 2.

Solution:

```
def imini(L):
    rang_min=0
    for i in range(1,len(L)):
        if L[i]<L[rang_min]:
            rang_min=i
    return rang_min
```

R3. Écrire une fonction `dim(img:list[list])->(int,int)` qui prend en argument une image `img` et qui renvoie le couple d'entiers `(h,w)` où `h` est la hauteur et `w` la largeur de l'image.

Solution:

```
def dim(img):
    h = len(img) # compte le nombre de sous-liste dans la liste, soit le nombre de lignes
    w = len(img[0]) # compte le nombre d'éléments dans la sous-liste img[0], soit le nombre de colonnes (toutes les lignes ont le même nombre d'éléments)
    return h,w
# OU
def dim(img):
    return (len(img),len(img[0]))
```

R4. Écrire une fonction d'en-tête `nulle(h:int,w:int)->list[list]` qui renvoie une liste de listes qui correspond à la matrice nulle de `h` lignes et `w` colonnes. *L'utilisation des fonctions de* ***numpy*** *est interdite.*
Dit autrement, si on voit la matrice en terme comme une matrice, elle renvoie l'image complètement noire de hauteur h et de largeur w.

Solution:

```
def nulle(h,w):
    return [[0 for j in range(w)] for i in range(h)]
# OU
def nulle(h,w):
    return [[0]*w for i in range(h)]
# OU
def nulle (h,w):
    L=[]
    for i in range(h): # les lignes
        L_ligne=[] # liste vide d'initialisation d'une ligne
        for j in range(w): # on rempli la ligne i
            L_ligne.append(0)
    return L
```

R5. Écrire une fonction `copie(img:list[list])->list[list]` qui prend en argument une image `img` et qui renvoie une copie de l'image `img` totalement indépendante de `img`. Il est interdit d'utiliser le module `copy`

et les fonctions `deepcopy` ou `deepcopy`, par contre, vous êtes invités à utiliser les deux fonctions écrites précédemment.

Solution:

```
def copie(img):
    h,w=dim(img)
    img2=nulle(h,w) # initialisation de l'image à 0
    for i in range(h):
        for j in range(w):
            img2[i][j]=img[i][j]
    return img2
```

Partie II Redimensionnement naïf

Pour réduire la largeur de moitié, on peut prendre un pixel sur deux dans chaque ligne, par exemple :

4	3	1	25
12	1	15	22
28	20	20	23
132	128	32	250

$\longrightarrow$

4	1
12	15
28	20
132	32

On supposera dans toute cette partie Partie II que la largeur de l'image est paire.

R6. Écrire une fonction `reduc_moitie(img:list[list])->list[list]` qui prend en argument une image `img` et qui renvoie une image deux fois moins large avec la méthode naïve qui vient d'être présentée.
Quelle est la complexité de ce programme ?

Solution: question tout à fait accessible !

```
def reduc_moitie(img):
    h,w=dim(img)
    img_red=nulle(h,w//2) # tableau qui recevra l'image réduite
    for i in range(h): # parcours de img
        for j in range(w//2):
            img_red[i][j]=img[i][2*j]
    return img_red
```

Il y a deux boucles for imbriquées, la première est de h itérations, et la deuxième de $w/2$, chaque itération étant de coût constant : $O(hw)$

Pour éviter de perdre complètement les pixels qu'on supprime, on peut fusionner les pixels $I_{i,2k}$ et $I_{i,2k+1}$ en un seul et unique pixel contenant comme valeur $(I_{i,2k} + I_{i,2k+1})//2$, ce qui donne sur notre exemple.

4	3	1	25
12	1	15	22
28	20	20	23
132	128	32	250

$\longrightarrow$

3	13
6	18
24	21
130	141

R7. Écrire une fonction `reduc_moitie2(img:list[list])->list[list]` qui prend en argument une image `img` et qui renvoie une image deux fois moins large avec la méthode naïve qui vient d'être présentée.

Solution: question tout à fait accessible !

```
def reduc_moitie2(img):
    h,w=dim(img)
    img_red=nulle(h,w//2) # tableau qui recevra l'image réduite
    for i in range(h): # parcours de img
        for j in range(w//2):
            img_red[i][j]=(img[i][2*j]+img[i][2*j+1])//2
    return img_red
```

Partie III Redimensionnement « intelligent »

A) Énergie d'un point

Tous les pixels n'ont pas la même importance dans une image, en effet si on enlève un pixel entouré de pixels de même couleur on ne le remarquera pas trop. Pour cela, on introduit l'énergie du pixel (i,j) (qui n'est pas au bord de l'image) comme étant le nombre $e_{i,j}$ défini par :

$$e_{i,j} = \left|\frac{I_{i-1,j} - I_{i+1,j}}{2}\right| + \left|\frac{I_{i,j-1} - I_{i,j+1}}{2}\right|$$

- Si $i = 0$, on mettra $\frac{I_{i,j} - I_{i+1,j}}{2}$ à la place de $\frac{I_{i-1,j} - I_{i+1,j}}{2}$, dans la première valeur absolue.
- Si $i = h-1$, on mettra $\frac{I_{i-1,j} - I_{i,j}}{2}$ à la place de $\frac{I_{i-1,j} - I_{i+1,j}}{2}$, dans la première valeur absolue.
- De même, avec la deuxième valeur absolue si $j = 0$ ou $j = w-1$.

La matrice $(e_{i,j})$ obtenue est appelée matrice d'énergie de l'image I.

R8. Écrire une fonction `energie(img:list[list])->list[list]` qui prend en argument une image `img` et qui renvoie une matrice correspondant à son énergie (les valeurs des "pixels" ne seront plus nécessairement des entiers). La case (i,j) de la matrice contient la valeur $e_{i,j}$ de l'énergie associée au pixel (i,j).
Afin de simplifier l'écriture de la fonction, on ne considérera que les indices (i,j) avec $0 < i < h-1$ et $0 < j < w-1$. Mais pour la suite du sujet, on supposera que la fonction `energie` renvoie l'énergie d'un pixel (y compris ceux sur les bords).

Solution:

```
# La fonction demandée :
def energie(img):
    h,w=dim(img)
    E=nulle(h,w)
    for i in range(1,h-1): # attention pour i dans ]0,h-1[
        for j in range(1,w-1): # attention pour j dans ]0,w-1[
            val1=abs((img[i-1][j]-img[i+1][j])//2)
            val2=abs((img[i][j-1]-img[i][j+1])//2)
            E[i][j]=val1+val2
    return E
# La vraie fonction qui fonctionne
def energie(img):
    h,w=dim(img)
    E=nulle(h,w)
    for i in range(h):
        for j in range(w):
```

```
            if i==0:
                val1=abs((img[i][j]-img[i+1][j])//2)
            elif i==h-1:
                val1=abs((img[i-1][j]-img[i][j])//2)
            elif j==0:
                val2=abs((img[i][j+1]-img[i][j])//2)
            elif j==w-1:
                val2=abs((img[i][j-1]-img[i][j])//2)
            else:
                val1=abs((img[i-1][j]-img[i+1][j])//2)
                val2=abs((img[i][j-1]-img[i][j+1])//2)
            E[i][j]=val1+val2
    return E
```

B) Présentation de l'algorithme

Pour avoir une image d'une colonne de moins, il suffit de supprimer un pixel par ligne, si on enlève le pixel d'énergie minimale de chaque ligne il risque d'y avoir un effet de décalage pas très gracieux. On pourrait aussi calculer la somme des énergies de chaque colonne et enlever la colonne de plus basse énergie. Soyons plus fin. On définit un chemin de pixels comme étant une suite de pixels se "touchant" (verticalement ou en diagonale) dont le premier est sur la première ligne de l'image et le dernier est sur la dernière ligne de l'image et qui possède exactement un pixel par ligne de l'image.

On définit l'**énergie d'un chemin** comme étant la **somme des énergies des pixels de ce chemin**.

Par exemple, voici un chemin représenté dans une matrice E (ce chemin est d'énergie 10) : $E =$

4	3	0	1
1	2	2	1
1	1	3	3
0	4	1	0

On représentera un chemin par une liste `chemin` de `h` nombres où `chemin[i]` est l'abscisse du chemin à la ligne `i` de la matrice, ainsi le chemin donné en exemple correspond à la liste `chemin=[1,2,1,1]`.

L'objectif est donc de déterminer un chemin d'énergie minimale pour ensuite supprimer les pixels correspondants de l'image.

C) Approche gloutonne

On peut appliquer une approche gloutonne, pour cela on détermine le pixel d'énergie minimale de la première ligne, c'est le premier pixel de notre chemin, ensuite on choisit parmi les trois (ou deux) pixels en dessous celui d'énergie minimale, c'est notre second pixel du chemin, etc... jusqu'à arriver en bas de l'image.

R9. Déterminer le chemin obtenu si on applique la stratégie gloutonne à la matrice E juste au dessus.

Solution: Selon glouton :

- ligne 0, on prend celui d'énergie minimale : 0
- ligne 1, on prend celui d'énergie minimale parmi 2, 2, 1 (accessibles depuis 0) : 1
- ligne 2, on prend celui d'énergie minimale parmi 3, 3 (accessibles depuis 1) : 3
- ligne 3, on prend celui d'énergie minimale 4,1,0 (ou 1,0) selon le choix du pixel de valeur 3 effectué avant : 0

On obtient donc le chemin `[2,3,2,3]` d'énergie totale 4.

R10. La solution obtenue par méthode gloutonne est-elle optimale ? On justifiera sa réponse.

Solution: On peut obtenir un chemin de longueur 3 : 0 -> 2 -> 1 -> 0
La stratégie gloutonne ne donne pas ici le résultat optimal.

R11. Écrire une fonction `glouton(e:list[list])->list` qui prend en argument une matrice d'énergie `e` (de la forme de E) et qui renvoie une liste `C` décrivant le chemin d'énergie minimale (`C` est de la forme de `chemin`) selon l'algorithme glouton. *Il faudra bien penser au fait qu'on ne peut accéder qu'aux cases touchant le pixel précédent, et distinguer différents cas (en bout de ligne : 2 cas ; au sein d'une ligne).*
Quel est la complexité de ce programme ?

Solution: `C` doit contenir les rangs, et non les valeurs des minimums.

```
def glouton(e):
    C=[] # chemin
    h,w=dim(e)
    j=imini(e[0])
    C.append(j) # 1er choix : plus petite valeur de E
    for i in range(1,h): # parcours des lignes suivantes de e, à la recherche de la plus petite valeur accessible depuis j
        if j==0:
            if e[i][j]>e[i][j+1]:
                j=j+1 # sinon la plus petite valeur est au rang j
        elif j==w-1:
            if e[i][j-1]<e[i][j]:
                j=j-1 # sinon la plus petite valeur est au rang j
        else:
            if e[i][j-1]<e[i][j] and e[i][j-1]<e[i][j+1]:
                j=j-1
            elif e[i][j+1]<e[i][j] and e[i][j+1]<e[i][j-1]:
                j=j+1
        C.append(j)
    return C
```

Cet algorithme parcourt une fois les lignes, et à chaque fois elle parcourt 3 éléments de la ligne pour rechercher le rang du minimum.
Ainsi la complexité est en $O(h)$.

R12. Écrire un programme `chemin_glouton(img:list[list])->list` prenant en argument une image `img` et qui renvoie le chemin minimal obtenu.

Solution:

```
def chemin_glouton(img):
    En=energie(img) # matrice des énergies de img
    return glouton(En) # calcul le chemin associé à ces énergies
```

D) Une approche dynamique

Pour le pixel de coordonnée (i, j) on considère l'ensemble des chemins qui vont de la première ligne à ce pixel de coordonnée (i, j) (ce pixel est inclus dans le chemin), parmi ces chemins l'un d'entre eux possède une énergie plus basse que tous les autres, on note $f(i, j)$ cette énergie.

Pour la matrice E précédente, le tableau contenant les valeurs de $f(i, j)$ (la case (i, j) contient $f(i, j)$) est :

4	3	0	1
4	2	2	1
3	3	4	4
3	7	4	4

R13. Notons e la matrice d'énergie de l'image. Donner la valeur $f(i, j)$ dans les cas suivants :

(a) Si $i = 0$.

Solution: Si $i = 0$, dans la première ligne, le chemin d'énergie minimale pour arriver en (i, j) est le contenu de la case (i, j) : $\boxed{f(i = 0, j) = e(i = 0, j)}$

(b) Si $i > 0$ et $j = 0$.

Solution: Si $i > 0$ et $j = 0$, on arrive à la ligne i, en venant de la ligne $i - 1$, juste au-dessus (colonne j) ou juste à droite (colonne $j + 1$), il faut choisir le plus court des deux à qui on ajoute la valeur de $e(i, j)$.

$\boxed{f(i, j) = \min(f(i - 1, j), f(i - 1, j + 1)) + e(i, j)}$

(c) Si $i > 0$ et $j = w - 1$.

Solution: De la même façon : $\boxed{f(i, j) = \min(f(i - 1, j - 1), f(i - 1, j)) + e(i, j)}$

(d) Si $i > 0$ et $0 < j < w - 1$.

Solution: Il y a trois possibilités pour arriver en (i, j) : au-dessus à gauche, au-dessus, et au-dessus à droite : $\boxed{f(i, j) = \min(f(i - 1, j - 1), f(i - 1, j), f(i - 1, j + 1)) + e(i, j)}$

R14. Écrire une fonction `tableau_energie(e:list[list])->list[list])` qui prend en argument la matrice des énergies `e` et qui renvoie un tableau de même taille que `e` contenant les valeurs de $f(i, j)$. Quelle est la complexité de ce programme ?

Les fonctions de python ***min****,* ***max*** *pourront être utilisées dans cette question.*

Solution:

```
def tableau_energie(e):
    h,w=dim(e)
    tab=nulle(h,w)
    for j in range(0,w): # première ligne (cas de base)
        tab[0][j]=e[0][j] # f(i,j)=e(i,j)
    for i in range(1,h): # parcours des lignes
        tab[i][0]=min(tab[i-1][0],tab[i-1][1])+e[i][0]
        tab[i][w-1]=min(tab[i-1][w-1],tab[i-1][w-2])+e[i][w-1]
        for j in range(1,w-1) : # parcours des colonnes
            tab[i][j]=min(tab[i-1][j-1],tab[i-1][j],tab[i-1][j+1])+e[i
   ][j]
    return tab
```

La complexité est en $O(hw)$.

R15. Connaissant tous les $f(i,j)$ (c'est-à-dire le tableau généré précédemment), expliquer comment déterminer le chemin minimal recherché.

Solution:

— On part de la dernière ligne du tableau ($i = h-1$), et on cherche la case (i,j) la plus faible possible. C'est l'arrivée du chemin, son contenu est l'énergie de ce chemin.
Ici l'arrivée est $(3,0)$ (d'énergie minimale $= 3$)

— Puis, on remonte d'une ligne, et on cherche la plus petite des deux/trois valeurs du dessus. Et ainsi de suite pour arriver sur la première ligne.

R16. Écrire une fonction `determine_chemin(tab:list[list])->list` qui prend en argument un tableau `tab` contenant les valeurs $f(i,j)$ et qui renvoie le chemin minimal recherché.

Solution:

```
def determine_chemin(tab):
    h,w=dim(tab)
    C=[0]*h # autant d'étape que de lignes
    i=h-1
    jmin=imini(tab[i])
    C[i]=jmin
    while i>0:
        i=i-1 # on remonte dans les lignes
        if 0<jmin<w-1:
            jmin=imini(tab[i][jmin-1:jmin+2])+jmin-1 # il ne faut pas oublier le décalage de jmin-1, car imini renverra 0 ou 1 ou 2
        elif jmin==0:
            jmin=imini(tab[i][0:2])
        else: # on cherche le rang de la case de la ligne i qui contient la plus petite valeur parmi ceux qui sont accessibles depuis le rang précédent
            jmin=imini(tab[i][w-2:w])+w-2
        C[i]=jmin
    return C
```

R17. On cherche à déterminer les valeurs de $f(i,j)$ par mémoïsation, c'est-à-dire de haut en bas.
Recopier et compléter la fonction ci-dessous qui doit renvoyer le dictionnaire `Df` dont les clés sont les couples (i,j) et les valeurs l'énergie correspondante.

Solution:

```
def tab_mem(e):
    Df={}
    h,w=dim(e)
    def tab_f(e,i,j):
        """  renvoie la valeur de f(i,j) """
        if (i,j) in Df :
            return Df[(i,j)]
        else:
            if i==0:
                Df[(i,j)]=e[i][j]
            elif j==0:
                Df[(i,j)]=min(tab_f(e,i-1,j),tab_f(e,i-1,j+1))+e[i][j]
```

```
            elif j==w-1:
                Df[(i,j)]=min(tab_f(e,i-1,j),tab_f(e(i-1,j-1))+e[i][j]
            else:
                Df[(i,j)]=min(tab_f(e,i-1,j-1),tab_f(e,i-1,j),tab_f(e,i
    -1,j+1))+e[i][j]
        return Df[(i,j)]
    for j in range(w):
        tab_f(e,h,j)
    return Df
```

Partie IV Base de données

On s'intéresse à une base de données de photographies qui contient trois tables, dont on donne quelques lignes :

`Photos`

id	titre	largeur	hauteur	lieu
1	Les goélands sur la plage	640	480	Dunkerque
2	La chèvre sur son rocher	1024	768	Barbières
3	Le chien suspect	800	600	Valence
⋮	⋮	⋮	⋮	⋮

`Animaux`

id	nom	type
1	goéland	oiseau
2	chèvre	mammifère
⋮	⋮	⋮

`EstPresent`

id	idphoto	idanimal	nombre
1	1	1	2
2	3	2	1
3	3	2	1
⋮	⋮	⋮	⋮

La première table `Photos` a cinq attributs dont une clé primaire `id` pour chaque photo.

La seconde table `Animaux` qui a trois attributs dont une clé primaire `id` pour chaque animal.

La troisième `EstPresent` a 4 attributs et fait la jonction entre les deux autres tables. Elle a une clé primaire `id`, elle permet de savoir que sur la photo dont l'identifiant est `idphoto` (c'est une clé étrangère en relation avec la clé primaire de la table `Photos`) il y a un animal dont l'identifiant est `idanimal` (c'est une clé étrangère en relation avec la clé primaire de la table `Animaux`) et en précise le nombre (par exemple il y a deux goélands sur la première photo et un chien sur la troisième, mais aussi une chèvre sur la troisième photo).

R18. Écrire une requête SQL renvoyant les titres des photos prises à "Valence".

Solution:

```
SELECT titre
FROM Photos
WHERE lieu="Valence"
```

R19. Écrire une requête SQL renvoyant les titres et les lieux des photos qui font plus que 1024 de large et plus de 768 de haut.

Solution:

```
SELECT titres, lieux
FROM Photos
WHERE largeur>1024 AND hauteur>768
```

R20. Écrire une requête SQL renvoyant le nombre de photos de chaque ville.

Solution:

```
SELECT lieu , COUNT(id)
FROM Photos
GROUP BY lieu
```

R21. Écrire une requête SQL renvoyant les lieux où au moins 20 photos ont été prises

Solution:

```sql
SELECT lieu , COUNT(id)
FROM Photos
GROUP BY lieu
HAVING COUNT(id)>=20
```

R22. Écrire une requête SQL renvoyant les lieux où il y a une photo contenant une chèvre.

Solution: Vous ne pouvez pas utiliser l'exemple donné par l'énoncé.

```sql
SELECT lieu
FROM Photos AS P
JOIN Animaux AS A
JOIN EstPresent AS EP
ON P.id=idphoto AND A.id=idanimal
WHERE nom="chèvre"
```

R23. Écrire une requête SQL renvoyant le nombre d'animaux par photos (le titre de la photo devra apparaître).

Solution:

```sql
SELECT titre,SUM(nombre)
FROM Photos AS P
JOIN Animaux AS A
JOIN EstPresent AS EP
ON P.id=idphoto AND A.id=idanimal
GROUP BY idphoto
```