DS 2 : lundi 16 décembre

Correction : algorithme de Seam carving

I] Préliminaires

A. SQL

Correction :

Q1.
```
SELECT titre
FROM photos
WHERE lieu="Dunkerque"
```

Q2.
```
SELECT titre,lieu
FROM photos
WHERE largeur>=1024 AND hauteur>=768
```

Q3.
```
SELECT lieu, COUNT()
FROM photos
GROUP BY lieu
```

Q4.
```
SELECT DISTINCT lieu
FROM photos
   JOIN estsur ON photos.id=estsur.idphoto
   JOIN animaux ON estsur.idanimal=animaux.id
WHERE animaux.nom="chien"
```

Q5.
```
SELECT photos.titre
FROM photos
   JOIN estsur ON photos.id=estsur.idphoto
   JOIN animaux ON estsur.idanimal=animaux.id
GROUP BY photos.id
HAVING COUNT()>=2
```

B. Fonction basique

Correction :

Q6.
```
def imin(lst):
    pos=0
    c=lst[0]
    for k in range(len(lst)):
        if lst[k]<c:
            pos=k
            c=lst[k]
    return pos
```

C. Traitement d'images

Correction :

Q7.
```
def dim(img):
    return (len(img[0]), len(img))
```

Q8.
```
def nulle(w,h):
    return [[0 for j in range(w)] for i in range(h)]
```

Q9. Cette fonction prend une image en argument et retourne l'image obtenue avec une symétrie verticale (ie d'axe (Oy)).

II] Redimensionnement naïf

Correction :

Q10.
```
def reduc_moitie(img):
    (w,h)=dim(img)
    res=nulle(w//2,h)
    for i in range(h):
        for k in range(w//2):
            res[i][k]=img[i][2*k]
    return res
```

Correction :

Q11.
```
def reduc_moitiev2(img):
    (w,h)=dim(img)
    res=nulle(w//2,h)
    for i in range(h):
        for k in range(w//2):
            res[i][k]=(img[i][2*k]+img[i][2*k+1])//2
    return res
```

III] Redimensionnement "intelligent"

A. Énergie d'un pixel

Correction :

Q12.
```
def energie(img):
    (w,h)=dim(img)
    res=nulle(w,h)
    for i in range(h):
        for j in range(w):
            if i==0:
                dx=img[i+1][j]-img[i][j]
            elif i==h-1:
                dx=img[i][j]-img[i-1][j]
            else:
                dx=(img[i+1][j]-img[i-1][j])/2
            if j==0:
                dy=img[i][j+1]-img[i][j]
            elif j==w-1:
                dy=img[i][j]-img[i][j-1]
            else:
                dy=(img[i][j+1]-img[i][j-1])/2
            res[i][j]=abs(dx)+abs(dy)
    return res
```

B. Présentation de l'algorithme

1/ Approche gloutonne

Correction :

Q13. On obtient le chemine `ae=[2,3,2,3]`

Q14.
```
def glouton(img):
    (w,h)=dim(img)
    ae=[0 for i in range(h)]

    ae[0]=imin(img[0])
    for i in range(1,h):
        j=ae[i-1]
        if j==0:
            aux=img[i][j:j+2]
            ae[i]=j+imin(aux)
        elif j==h-1:
            aux=img[i][j-1:j+1]
            ae[i]=j-1+imin(aux)
        else:
            aux=img[i][j-1:j+2]
            ae[i]=j-1+imin(aux)
    return ae
e = [[4, 3, 0, 1], [1, 2, 2, 1], [1, 1, 3, 3], [0, 4, 1, 0]]
ae=glouton(e)
```

Q15. Pour $M = \begin{pmatrix} 0 & 9 & 9 \\ 9 & 9 & 0 \\ 9 & 9 & 0 \end{pmatrix}$, la stratégie gloutonne donne le chemin [0,0,0] qui est d'énergie 18, alors que le chemin optimal est clairement [2,2,2] (qui est d'énergie 9).

2/ Approche dynamique

Correction :

Q16. Pour $i = 0$, on a clairement que le chemin d'énergie minimale qui relie un pixel de la première ligne à un pixel de la première ligne est juste ce pixel (l'unique manière de relier), donc $f(i,j) = e_{i,j}$.
Pour $i \neq 0$ et j quelconque, si on a un chemin $c = [k_0, \dots, k_j]$ d'énergie minimale qui relie un pixel de la première ligne à ce pixel (i,j) (on a $k_j = j$) alors si on met de côté le dernier élément du chemin on a un chemin $c' = [k_0, \dots, k_{j-1}]$ qui va jusqu'au pixel $(i-1, j')$ où $j' = k_{j-1} \in \{j-1, j, j+1\}$ (enlever $j-1$ si $j = 0$ et $j+1$ si $j = w-1$) et forcément ce chemin c' est d'énergie minimale (sinon on pourrait construire un chemin d'énergie plus petite que notre chemin, ce qui contredirait la minimalité du chemin initial), ce qui donne la formule.

Q17. **Q18.**, **Q19.** et **Q20.** Sans mémoïsation, le même calcul est effectué plusieurs fois, ce qui fait que la complexité est exponentielle, ainsi une telle fonction n'est pas utilisable en pratique.

```
fdict=dict()
chemdict=dict()
def f(i,j):
    if (i,j) in fdict:
        return fdict[(i,j)]
    else:
        if i==0:
            res=e[i][j]
            chemdict[(i,j)]=[j]
        else:
            if j==0:
                aux=[f(i-1,j),f(i-1,j+1)]
                jp=j+imin(aux)
                res=e[i][j]+f(i-1,jp)
```

```
                    chemdict[(i,j)]=chemdict[(i-1,jp)]+[j]
                elif j==len(e[0])-1:
                    aux=[f(i-1,j-1),f(i-1,j)]
                    jp=j-1+imin(aux)
                    res=e[i][j]+f(i-1,jp)
                    chemdict[(i,j)]=chemdict[(i-1,jp)]+[j]
                else:
                    aux=[f(i-1,j-1),f(i-1,j),f(i-1,j+1)]
                    jp=j-1+imin(aux)
                    res=e[i][j]+f(i-1,jp)
                    chemdict[(i,j)]=chemdict[(i-1,jp)]+[j]
            fdict[(i,j)]=res
            return res
```

Q21. Il faut déterminer j tel que $f(h,j)$ soit minimal et récupérer le chemin minimal associé.

Q22.
```
e = [[4, 3, 0, 1], [1, 2, 2, 1], [1, 1, 3, 3], [0, 4, 1, 0]]
(w,h)=dim(e)
der=[f(h-1,j) for j in range(w)]
j=imin(der)
ae=chemdict[(h-1,j)]
```

Correction :

Q23. Ici j'ai fait le choix de garder en mémoire tous les chemins partiels, il est possible de ne pas le faire et de retrouver `ae` à la fin, en effet le dernier élément de `ae` est l'indice j_{h-1} qui réalise le minimum sur la dernière ligne de `res`, l'indice j_{h-2} ne peut prendre que trois valeurs, il suffit de prendre le j (parmi ces trois possible) où la valeur `res[h-2][j]` est la plus petite.

```
def seamcarving(e):
    (w,h)=dim(e)
    res=nulle(w,h)
    chem=[[[] for j in range(w)] for i in range(h)]
    for j in range(w):
        res[0][j]=e[0][j]
        chem[0][j]=[j]

    for i in range(1,h):
        for j in range(0,w):
            if j==0:
                aux=[res[i-1][j],res[i-1][j+1]]
                jp=j+imin(aux)
                res[i][j]=e[i][j]+res[i-1][jp]
                chem[i][j]=chem[i-1][jp]+[j]
            elif j==len(e[0])-1:
                aux=[res[i-1][j-1],res[i-1][j]]
                jp=j-1+imin(aux)
                res[i][j]=e[i][j]+res[i-1][jp]
                chem[i][j]=chem[i-1][jp]+[j]
            else:
                aux=[res[i-1][j-1],res[i-1][j],res[i-1][j+1]]
                jp=j-1+imin(aux)
                res[i][j]=e[i][j]+res[i-1][jp]
                chem[i][j]=chem[i-1][jp]+[j]
    jd=0
    jmin=res[h-1][jd]
    for j in range(1,w):
        if res[h-1][j]<jmin:
            jd=j
            jmin=res[h-1][jd]
    return chem[h-1][jd]
```