Proposition de corrigé du TP6 : Transformations d'images

1 Traitement d'images avec Python

1.1 Symétrie axiale

1. Créer une fonction `matrice_blanche(p,q)` qui renvoie la matrice associée à une image entièrement blanche (elle contient uniquement le triplet `[255,255,255]` répété pq fois). Comment obtient-on une image noire ? Une image bleue ?

```
def image_blanche(p,q):
    ligne = [[255,255,255] for i in range(p)]
    matrice_complete = [ligne for j in range(q)]
    img_retour = np.array(matrice_complete)
    return(img_retour)
```

```
def image_bleue(p,q):
    ligne = [[0,0,255] for i in range(p)]
    matrice_complete = [ligne for j in range(q)]
    img_retour = np.array(matrice_complete)
    return(img_retour)
```

2. Étant donné un pixel a de coordonnées (x, y) dans l'image initiale, exprimer ses coordonnées (x', y') dans l'image résultat d'une symétrie d'axe vertical passant par le centre de l'image.
On a :
$$(x', y') = (x, q - 1 - y)$$

3. En déduire une fonction `symetrie` qui prend en argument une matrice-image `img` et qui renvoie une nouvelle matrice-image résultat de la symétrie d'axe vertical. Les dimensions $p \times q$ de l'image initiale peuvent être calculées par l'instruction `np.shape`.

```
def symetrie(img):
    p, q = img.shape[0], img.shape[1]
    img2 = image_blanche(p,q)
    for x in range(p):
        for y in range(q):
            img2[x, q-1-y] = img[x, y]
    return img2
```

1.2 Rotation d'un quart de tour

4. Étant donné un pixel a de coordonnées (x, y) dans l'image initiale, exprimer ses coordonnées (x', y') dans l'image résultat d'une rotation d'un quart de tour vers la droite.
On a :
$$(x', y') = (y, p - 1 - x)$$

5. En déduire une fonction `rotation(img)` qui prend en argument une matrice-image `img` et qui renvoie une nouvelle matrice-image résultat de la rotation d'un quart de tour.

```
def rotation(img):
    p, q = img.shape[0], img.shape[1]
    img2 = image_blanche(q,p)
    for x in range(p):
        for y in range(q):
            img2[y, p-1-x] = img[x, y]
    return img2
```


2 Transformation du photomaton

6. À partir des coordonnées (x, y) d'un pixel de l'image initiale, exprimer ses coordonnées (x', y') dans l'image résultat. Il sera nécessaire de distinguer quatre cas suivant la parité de x et de y.
On a :
$$(x', y') = \begin{cases} (x/2, y/2) & \text{si } x \text{ et } y \text{ sont pairs} \\ (x/2, \lfloor y/2 \rfloor + q/2) & \text{si } x \text{ est pair et } y \text{ impair} \\ (\lfloor x/2 \rfloor + p/2, y/2) & \text{si } x \text{ est impair et } y \text{ pair} \\ (\lfloor x/2 \rfloor + p/2, \lfloor y/2 \rfloor + q/2) & \text{si } x \text{ et } y \text{ sont impairs} \end{cases}$$

7. En déduire une fonction `photomaton(img)` qui prend en argument une matrice-image et renvoie une nouvelle matrice-image résultant de la transformée du photomaton de l'image initiale.
On peut commencer par simplifier la fonction principale :

```
def photomat(k, d):
    if k % 2 == 0:
        return k // 2
    else:
        return k // 2 + d // 2
```

ce qui conduit à la définition suivante :

```
def photomaton(img):
    p, q = img.shape[0], img.shape[1]
    img2 = image_blanche(p,q)
    for x in range(p):
        for y in range(q):
            img2[photomat(x, p), photomat(y,q)] = img[x, y]
    return img2
```

8. Rédiger un script Python pour visualiser les transformations successives de l'image *mario.png* jusqu'à revenir à celle-ci. Quelle est la période de la transformation du photomaton pour cette image ? On considère que les pixels sont revenus à leur place lorsque les pixels de même teinte sont aux mêmes endroits.
Le plus simple est de passer par une boucle while et de vérifier quand est ce que l'image calculée est égale à celle du départ. Ici on rajoute un compteur pour obtenir la période de la transformation.

```
## Version utilisant la fonction python all() qui renvoie True si tous les éléments

img = mario
compteur=0
while True:
    img = photomaton(img)
    plt.figure()
    plt.imshow(img)
    plt.show()
    compteur+=1
    if (img == mario).all():
        print(compteur)
        break

## Version en codant une fonction réalisant la comparaison d'image

img = mario
compteur=0
while True:
    img = photomaton(img)
    plt.figure()
    plt.imshow(img)
    plt.show()
    compteur+=1
```

```
    print(compteur)
    if test_image(img,mario):
        print(compteur)
        break
```

9. Rédiger une fonction `periode_photomaton(img)` qui prend en argument une matrice-image et retourne la période de la transformation du photomaton pour celle-ci.

```
def periode_photomaton(img):
    p, q = img.shape[0], img.shape[1]
    n, t = 1, 2
    while (t-1) % (p-1) != 0 or (t-1)% (q-1) != 0:
        n += 1
        t *= 2
    return n
```

10. Récupérer l'image *mario.png*. Quelle est la période de sa transformation du photomaton ?

```
print(periode_photomaton(mario))
```

11. Rédiger une fonction `photomaton2(img,n)` qui prend en arguments une image et un entier n et qui retourne la n^{e} itération de l'image par la transformation du photomaton, en calculant celle-ci directement.

```
def photomaton2(img, n):
    p, q = img.shape[0], img.shape[1]
    img2 = image_blanche(p,q)
    ligne = []
    for x in range(p):
        u = x
        for _ in range(n):
            u = photomat(u, p)
        ligne.append(u)
    colonne = []
    for y in range(q):
        v = y
        for _ in range(n):
            v = photomat(v, q)
        colonne.append(v)
    for x in range(p):
        for y in range(q):
            img2[ligne[x], colonne[y]] = img[x, y]
    return img2
```

3 Transformation du boulanger

12. Rédiger une fonction `boulanger(img)` qui prend en argument une image et retourne la transformée du boulanger de celle-ci.

```
def boulange(x, y, p, q):
    x1, y1 = x // 2, 2 * y + x % 2
    if y1 < q:
        return x1, y1
    else:
        return p - 1 - x1, 2 * q - 1 - y1

def boulanger(img):
    p, q = img.shape[0], img.shape[1]
    img2 = image_blanche(p,q)
    for x in range(p):
        for y in range(q):
            img2[boulange(x, y, p, q)] = img[x, y]
    return img2
```



```
boulang_mario=boulanger(mario)
plt.figure()
plt.imshow(boulang_mario)
plt.show()
```

13. Rédiger un script *Python* pour visualiser les transformations successives de l'image `mario.png` jusqu'à revenir à celle-ci. Quelle est la période de la transformation du boulanger pour cette image ?

```
img = mario
compteur=0

while True:
    img = boulanger(img)
    plt.figure()
    plt.imshow(img)
    plt.show()
    compteur+=1
    if test_image(img,mario):
        break
print(compteur)
```

4 Et pour les plus rapides

14. Écrire une fonction `periode_pixel(x,y,p,q)` qui prend en arguments les coordonnées (x, y) d'un pixel ainsi que la taille (p, q) de l'image auquel il appartient et qui retourne la période de retour de ce pixel à sa place initiale.

```
def periode_pixel(x, y, p, q):
    n = 1
    x1, y1 = boulange(x, y, p, q)
    while (x1, y1) != (x, y):
        n += 1
        x1, y1 = boulange(x1, y1, p, q)
    return n
```

15. En déduire une fonction `tableau_des_periodes(img)` qui calcule le tableau des périodes de tous les pixels d'une image. On optimisera le temps de calcul en observant que tous les pixels appartenant à une même orbite [1] ont une période identique.

```
def tableau_des_periodes(img):
    p, q = img.shape[0], img.shape[1]
    r = np.zeros((p, q), dtype=int)
    for x in range(p):
        for y in range(q):
            if r[x, y] == 0:
                s = periode_pixel(x, y, p, q)
                x1, y1 = x, y
                for u in range(s):
                    r[x1, y1] = s
                    x1, y1 = boulange(x1, y1, p, q)
    return r
```

16. Rédiger enfin une fonction `periode_boulanger(img)` qui calcule la période de la transformation du boulanger d'une image. On pourra utiliser la fonction `gcd` du module `math` pour calculer le *PGCD* de deux entiers naturels. Quelle est la période de l'image *mario.png* ?

```
def pgcd(a,b):
    while b != 0:
        a,b=b,a%b
```

1. On appelle orbite d'un pixel l'ensemble des positions qu'il occupe durant les applications successives de la transformation du boulanger.

```
    return a

def ppcm(a,b):
    if (a==0) or (b==0):
        return 0
    else:
        return (a*b)//pgcd(a,b)


def periode_boulanger(img):
    p, q = img.shape[0], img.shape[1]
    r = tableau_des_periodes(img)
    lst = []
    for x in range(p):
        for y in range(q):
            if r[x, y] not in lst:
                lst.append(int(r[x, y]))
    m = 1
    for n in lst:
        m = ppcm(m, n)
    return m
```

17. Rédiger une fonction `boulanger2(img,n)` qui calcule la n^{e} transformation du boulanger d'une image.

```
def boulanger2(img, n):
    p, q = img.shape[0], img.shape[1]
    r = tableau_des_periodes(img)
    img2 = np.empty_like(img)
    for x in range(p):
        for y in range(q):
            if r[x, y] != 0:
                u, v = x, y
                for _ in range(n % r[x, y]):
                    u, v = boulange(u, v, p, q)
                x1, y1 = x, y
                for _ in range(r[x, y]):
                    img2[u, v] = img[x1, y1]
                    x1, y1 = boulange(x1, y1, p, q)
                    u, v = boulange(u, v, p, q)
                    r[x1, y1] = 0
    return img2
```