Info

TP image - Photomaton

Dans ce projet, nous allons nous intéresser à certaines transformations bijectives d'une image. Ce type de transformation déplace les points d'une image d'un endroit à un autre sans en ajouter ni en enlever aucun. Ceci revient à appliquer une permutation sur l'ensemble des points d'une image.

Une propriété remarquable de ces transformations bijectives est qu'elles reviennent toujours au point de départ après un nombre d'applications plus ou moins important (ordre de transformation). Par exemple, la transformée par symétrie de l'image par rapport à un axe vertical passant par son milieu est d'ordre 2, tandis que la transformée par rotation d'un quart de tour est d'ordre 4.

1. Traitement d'image en Python

Pour ce TP, nous allons utiliser les bibliothèques `numpy` et `matplotlib.pyplot`.

On utilisera en particulier les deux fonctions suivantes de la bibliothèque `matplotlib.pyplot` :

- la fonction `plt.imread` prend en argument une chaîne de caractères décrivant le chemin d'accès à un fichier image au format `.png` et retourne un tableau `Numpy`. Ce tableau a même dimension que l'image, et ses cases contiennent un quadruplet donnant les 3 composantes RGB du pixel correspondant ainsi qu'une dernière composante correspondant à l'indice de transparence (toujours égal à 1 ici et que nous n'utiliserons pas).
- la fonction `plt.imshow` prend en argument un tableau `Numpy` et affiche l'image associée à cette matrice. On fera suivre cette instruction par la commande `plt.show()`.
- la fonction `plt.savefig(Nom_Fichier)` qui prend en argument une chaîne de caractères (par exemple `'img.png'` et qui permet de sauvegarder notre image dans le répertoire courant avec le nom 'img.png' et l'extension `.png` (on pourra utiliser d'autres extensions...).

1.1. Travail préliminaire

1. Copiez les fichiers `.png` présents dans le dossier `Documents en consultation` de la classe dans votre espace de travail.

2. En vous reportant éventuellement à votre cours d'introduction à `Numpy` (Info 12), rappelez les instructions permettant d'avoir accès aux dimensions d'un tableau `Numpy` (nombre de lignes, nombre de colonnes et nombre d'éléments par case ou pixel). Rappeler aussi la fonction permettant de créer un tableau de rempli de $0.$ et ayant la même structure que le tableau matrice-image.

Rq

On rappelle que pour créer une copie de tableau , on peut utiliser la fonction `np.copy`. On peut aussi pour créer un tableau vierge (rempli de 0) en utilisant la fonction `zeros_like(img)` qui renvoie un tableau de mêmes dimensions (structure) que le tableau `img`.

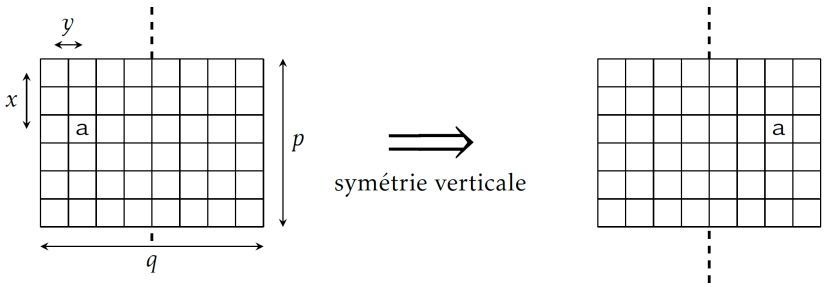
3. Créez un script Python permettant d'affecter à une variable `picasso` le tableau `numpy` associé à l'image `picasso.png`. Ecrire les instructions permettant d'afficher les dimensions de l'image. Vérifiez si la structure du tableau `picasso` correspondant à celle décrite ci-dessus. Affichez l'image picasso (par une commande `Python` !).

1.2. Transformation par symétrie

Nous allons commencer par créer une fonction permettant de transformer une image par symétrie verticale. Il faudra recopier chacun des pixels de la matrice-image initiale à son nouvel emplacement dans une matrice-image vierge.

Dans le cas de la symétrie d'axe vertical, l'image transformée a les mêmes dimensions que l'image initiale. On commencera donc par créer une matrice-image vierge de même dimension.

4. Etant donné un pixel a de coordonnées (x, y) dans l'image initiale, exprimer ses coordonnées (x_s, y_s) dans l'image résultat d'une symétrie d'axe vertical passant par le centre de l'image :



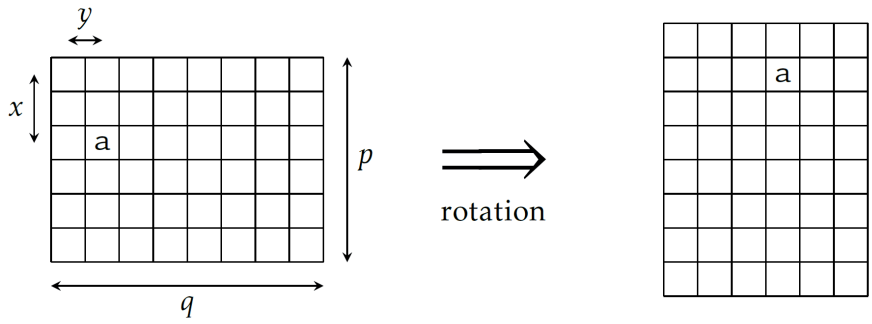
5. En déduire une fonction `symetrie(img)` qui prend en argument une matrice-image `img` et qui renvoie une nouvelle matrice-image résultat de la symétrie d'axe vertical.

6. On testera cette fonction en l'appliquant à la matrice `picasso` et en affichant l'image transformée.

7. Vérifiez par un test l'ordre de cette transformation.

1.3. Rotation d'un quart de tour

Pour une rotation d'un quart de tour, les dimensions de l'image résultat diffèrent de celles de l'image initiale lorsque cette dernière est rectangulaire. On ne pourra donc pas utiliser la commande `zeros_like(img)` pour initialiser une matrice-image vierge.



8. Etant donné un pixel a de coordonnées (x, y) dans l'image initiale, exprimez ses coordonnée (x_R, y_R) dans l'image résultat d'une rotation d'un quart de tour vers la droite.

.....

.....

9. En déduire une fonction `rotation(img)` qui prend en argument une matrice-image et qui renvoie une nouvelle matrice-image résultat de la rotation d'un quart de tour.
On testera cette fonction en faisant apparaître le résultat de `rotation(picasso)` puis de ses itérées.

2. Ordre d'une transformation

Une propriété remarquable des transformations bijectives est qu'elles permettent toujours de revenir au point de départ après un nombre d'applications plus ou moins important appelé ordre de transformation.

Rq

Pour comparer deux tableaux Numpy, on peut utiliser les commandes suivantes :

- `(img1==img2).any()` renvoie `True` si au moins un élément de `img1` est égal à un élément de `img2`. On peut modifier la nature du test avec par exemple : `(img1!=img2).any()` qui renvoie `True` si `img1` et `img2` ont au moins un élément différent.
- `(img1==img2).all()` renvoie `True` si tous les éléments de `img2` sont identiques aux éléments de `img1` et `False` sinon.

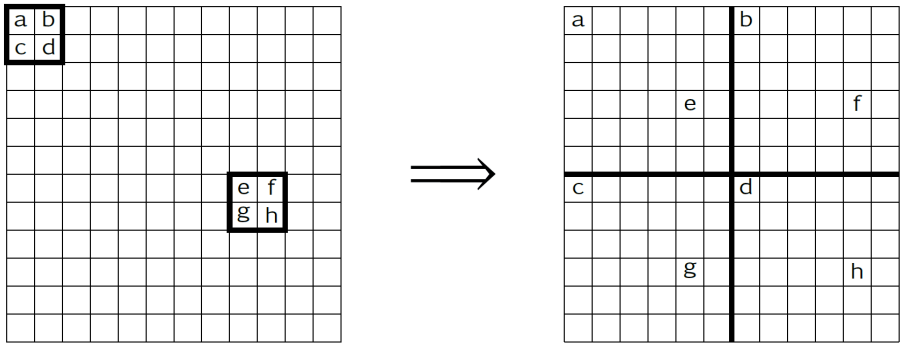
10. Proposez une fonction `ordre(fct,img)` qui prend en argument une fonction bijective `fct` et une matrice-image `img` et qui renvoie un entier correspondant à l'ordre de transformation de la fonction `fct`.

11. Testez votre fonction `ordre` sur les fonctions `symetrie` et `rotation`.

3. Transformation du photomaton

Cette transformation "réduit" la taille de l'image de moitié pour obtenir quatre morceaux analogues que l'on place en carré pour obtenir une image de même taille que l'image origine.

Pour que cette transformation soit bijective, on a découpé l'image initiale en paquets carrés de quatre pixels (2×2) puis pour chaque paquet carré de quatre pixels, on utilise celui en haut à gauche pour l'image réduite en haut à gauche, celui en haut à droite pour l'image réduite en haut à droite, et ainsi de suite.
On notera que pour que cette transformation soit définie, il est nécessaire que les dimensions de l'image soient paires.



12. A partir des coordonnées d'un pixel initial (x, y), exprimez ses coordonnées (x_p, y_p) dans l'image résultat. Il pourra être nécessaire de distinguer quatre cas suivant la parité de x et y.

13. En déduire une fonction `photomaton(img)` qui prend en argument une matrice-image `img` et renvoie une nouvelle matrice-image résultant de la transformée du photomaton de l'image initiale. On pourra commencer par écrire une fonction `coord_photo(x,N)` qui prend en argument une coordonnée d'un pixel (abscisse ou ordonnée) ainsi que le nombre de lignes ou de colonnes et qui renvoie la nouvelle position de ce pixel après la transformation du photomaton.

14. Rédigez un script Python permettant de visualiser les transformations successives de l'image `picasso.png` jusqu'à revenir à celle-ci. Quelle est la période de la transformation du photomaton pour cette image ?

15. Plus généralement, si on considère une image de taille $p \times q$ il est possible de prouver que la période de la transformation du photomaton est le plus petit entier n pour lequel $p - 1$ et $q - 1$ divisent $2^n - 1$.

Proposez une fonction `periode_photomaton` qui prend en argument une matrice image et qui renvoie la période de la transformation du photomaton. Vérifiez la cohérence de votre résultat en appliquant cette fonction ainsi que la fonction `ordre` à la matrice-image. Conclusion. Testez cette fonction sur la seconde image disponible.

On souhaite visualiser la 180^e itération de l'image `matisse.png`, mais il n'est pas question de générer les 179 images (ou tableaux) intermédiaires car le calcul serait trop long.

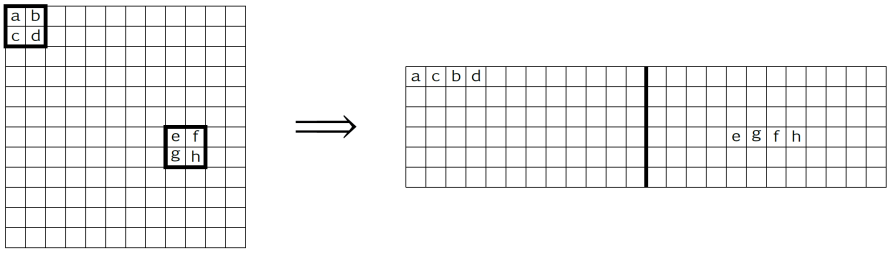
16. Rédigez une fonction `photomaton2(img,n)` qui prend en arguments une image `img` et un entier `n` et qui retourne la n^e itérée de l'image par la transformation du photomaton en calculant cell-ci directement.

17. Utilisez cette fonction pour visualiser la 180^e itération de la seconde image disponible. Pouvez-vous expliquer le résultat obtenu ?

4. Transformation du boulanger

Cette transformation s'appelle ainsi car elle s'apparente au travail du boulanger qui réalise une pâte feuilletée. L'image est tout d'abord aplatie, puis coupée en deux, puis la partie droite est placée sous la partie gauche en la faisant tourner de 180°.

Pour que cette transformation soit bien définie, il est nécessaire là encore que les dimensions de l'image soient paires. L'image initiale est découpée en paquets carrés de quatre pixels puis ceux-ci sont entrelacés pour obtenir l'image intermédiaire aplatie.



18. Proposez une fonction `boulanger(img)` qui prend un argument une image, et retourne la transformée du boulanger de celle-ci. On pourra dans un premier temps définir une fonction `coord_boulanger(x,y)` qui prend en argument les coordonnées d'un pixel (x, y) et qui renvoie les nouvelles coordonnées de ce pixel après la transformation du boulanger. Quelle est la période de la transformation du boulanger pour cette image ?

5. Pour les plus rapides

Le calcul de la période de la transformation du boulanger est plus délicat à réaliser et peut conduire à des valeurs très importantes. On l'obtient en déterminant d'abord la période de retour $r(x,r)$ de chaque pixel de coordonnées (x,y) puis en prenant le ppcm de toutes ces valeurs $r(x,y)$.

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

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

20. En déduire une fonction `tableau_periodes(img)` qui calcule le tableau des périodes de tous les pixels d'une image. On optimisera le temps de calculs en observant que tous les pixels appartenant à une même orbite ont une période identique.

21. Ecrivez une fonction `periode_boulanger(img)` qui calcule la période de la transformation du boulanger dans une image. On pourra utiliser la fonction `gcd` du module `fractions` pour calculer le pgcd de deux entiers naturels. Quelle est la période de l'image `matisse.png`).

22. Ecrivez une fonction `boulanger2(img,n)` qui calcule la n^e transformation du boulanger d'une image. Pour pouvoir utiliser de grandes valeurs de n sans que temps d'attente soit rédhibitoire, il faudra calculer la position finale de chaque pixel (x,y) à l'aide du reste de la division euclidienne de n par (x,y) et traiter chaque pixel d'une même orbite dans le même temps.

Afficher l'itération 6791 de l'image `matisse.png`. Quel est le pourcentage de pixels situés à leur place initiale ? Même questions avec n=149677500038576200.