

Dernière mise à jour	Informatique	Denis DEFAUCHY
09/02/2022	7 - Matrices de pixels et images	TD 7-6 - Images cachées

Informatique

7

Matrices de pixels et images

TD 7-6

Images cachées

Bits de poids faibles et forts

Dernière mise à jour	Informatique	Denis DEFAUCHY
09/02/2022	7 - Matrices de pixels et images	TD 7-6 - Images cachées

Affichage des images

Afin d'assurer un fonctionnement rapide sur tous les ordinateurs, je vous mets à disposition un dossier à télécharger COMPLETEMENT, soit le dossier contenant les 3 fichiers, et non les 3 fichiers séparément.



Se connecter

Enregistrer dans Dropbox

Télécharger

Dossier_Partagé

Trié par nom



7-5 - TD - Images cachées

Sans ouvrir le dossier, faite juste « Télécharger – Téléchargement direct » puis mettez ce dossier dans votre répertoire personnel.

[LIEN](#)

Si le téléchargement est sous forme de Rar, Zip... Pensez à dézipper l'archive afin d'avoir le dossier voulu !

Question 1: Télécharger et exécuter le code fourni qui affichera les trois images « Image_A_Decoder_1 », « Image_A_Decoder_2 » et « Image_Test » sous Python

Dernière mise à jour	Informatique	Denis DEFAUCHY
09/02/2022	7 - Matrices de pixels et images	TD 7-6 - Images cachées

Travail sur les bits des nombres binaires

A un entier N entre 0 et 255 (codé donc sur 8 bits) correspond un nombre binaire $0bX_7X_6X_5X_4X_3X_2X_1X_0$ où chaque X_i vaut soit 0, soit 1, de sorte que :

$$N = X_7 \cdot 2^7 + X_6 \cdot 2^6 + X_5 \cdot 2^5 + X_4 \cdot 2^4 + X_3 \cdot 2^3 + X_2 \cdot 2^2 + X_1 \cdot 2^1 + X_0 \cdot 2^0$$

L'opérateur « & » appliqué à deux nombres entiers sous forme décimale donne en résultat un nombre décimale. Faites l'essai :

$$78 \& 102 = 70$$

Pour traduire un nombre entier en binaire, il suffit d'utiliser la commande `bin(N)`. Attention, le résultat est une chaîne de caractères, nous ne l'utiliserons que pour voir l'effet de cet opérateur « & ».

<code>bin(78)</code>	<code>'0b1001110'</code>
<code>bin(102)</code>	<code>'0b1100110'</code>
<code>bin(70)</code>	<code>'0b1000110'</code>

A	78	0	1	0	0	1	1	1	0
B	102	0	1	1	0	0	1	1	0
A&B	70	0	1	0	0	0	1	1	0

Regardez bien... L'opérateur & effectue le test logique entre les bits de chacun des nombres 78 et 102 en binaire. Ainsi, $1 \& 1 = 1$, $1 \& 0 = 0$ et $0 \& 0 = 0$

Génial non ? Si on veut récupérer les 4 bits de poids fort d'un nombre N supposé codé sur 8 bits, on peut utiliser l'opérateur & entre le nombre N et l'entier K dont la représentation binaire est « 11110000 ».

De même si on veut récupérer les 3 bits de poids faible d'un nombre N sur 8 bits, on fera $N \& K$ avec K l'entier associé à la représentation binaire « 0000111 ».

Soient deux entiers $K_{Faibles}$ et K_{Forts} permettant de créer un nouveau nombre composé par les Nb bits soit de poids faibles, soit de poids forts d'un entier N à l'aide des opérations $N \& K_{Faibles}$ et $N \& K_{Forts}$

Nous noterons, quel que soit l'entier N :

$$N_{Faibles} = N \& K_{Faibles} \quad ; \quad N_{Forts} = N \& K_{Forts}$$

Question 2: Mettre en place deux fonctions $f_Forts(Nb)$ et $f_Faibles(Nb)$ renvoyant ces deux nombres entiers (en décimal) K_{Forts} et $K_{Faibles}$ en fonction du nombre de bits Nb souhaités sachant que nous travaillerons exclusivement sur des nombres codés sur 8 bits.

Astuce : on pensera à réaliser une somme de puissances de 2
Vérifiez :

```
>>> f_Forts(0)  >>> f_Faibles(0)
0               0
>>> f_Forts(1)  >>> f_Faibles(1)
128             1
>>> f_Forts(4)  >>> f_Faibles(4)
240             15
>>> f_Forts(7)  >>> f_Faibles(7)
254             127
>>> f_Forts(8)  >>> f_Faibles(8)
255             255
```

Dernière mise à jour	Informatique	Denis DEFAUCHY
09/02/2022	7 - Matrices de pixels et images	TD 7-6 - Images cachées

Récupération des n bits de poids forts ou faibles des entiers R , G et B d'un pixel d'une image

Dans la suite, on appelle Px un pixel de l'image, c'est-à-dire un triplet de 3 entiers codés sur 8 bits contenu dans une image, elle-même sous forme d'array de triplets d'entiers sur 8 bits.

Question 3: Créer la fonction $f_RGB_Faibles(Px,n)$ retournant le triplet $[R_{Faibles}, G_{Faibles}, B_{Faibles}]$ correspondant aux entiers associés aux n bits de poids faibles des entiers R , G et B de Px

```
>>> f_RGB_Faibles([100,121,157],0)
[0, 0, 0]

>>> f_RGB_Faibles([100,121,157],1)
[0, 1, 1]

>>> f_RGB_Faibles([100,121,157],4)
[4, 9, 13]

>>> f_RGB_Faibles([100,121,157],7)
[100, 121, 29]

>>> f_RGB_Faibles([100,121,157],8)
[100, 121, 157]
```

Vérifiez :

Question 4: Créer la fonction $f_RGB_Forts(Px,n)$ retournant le triplet $[R_{Forts}, G_{Forts}, B_{Forts}]$ correspondant aux entiers associés aux n bits de poids forts des entiers R , G et B de Px

```
>>> f_RGB_Forts([100,121,157],0)
[0, 0, 0]

>>> f_RGB_Forts([100,121,157],1)
[0, 0, 128]

>>> f_RGB_Forts([100,121,157],4)
[96, 112, 144]

>>> f_RGB_Forts([100,121,157],7)
[100, 120, 156]

>>> f_RGB_Forts([100,121,157],8)
[100, 121, 157]
```

Affichage des seuls bits de poids faibles ou forts d'une image

Dans cette partie, chaque fonction créée devra retourner une image.

L'idée sera la suivante :

- Créer une copie de l'image traitée (pas d'égalesisation)
- Changer les triplets RGB de ses pixels
- Retourner la nouvelle image

On souhaite pouvoir afficher une image en ne prenant que, soit ses n bits de poids forts, soit ses n bits de poids faibles, les autres bits étant mis à 0. n sera donc un entier compris entre 1 et 7 (à 0, image noire, et à 8, image initiale).

Question 5: Créer la fonction $f_ImBitsFortsForts_n(im,n)$ récupérant sur l'image im les n bits de poids forts de chaque couleur de chaque pixel, et les affectant à une nouvelle image qui sera retournée par la fonction à son appel

Question 6: Créer la fonction $f_ImBitsFaiblesFaibles_n(im,n)$ récupérant sur l'image im les n bits de poids faibles de chaque couleur de chaque pixel, et de les affecter à une nouvelle image qui sera retournée par la fonction à son appel

Question 7: Testez vos fonctions sur l'image « Image_Test.bmp », en affichant ses 4 bits de poids fort, 2 bits de poids forts et 4 bits de poids faibles

Dernière mise à jour	Informatique	Denis DEFAUCHY
09/02/2022	7 - Matrices de pixels et images	TD 7-6 - Images cachées

Création d'images par utilisation des 4 bits de poids faibles ou forts d'une autre image

Les opérateurs « << » et « >> » ont un rôle de décalage des bits qui composent un entier. Essayez les exemples suivant en affichant en même temps les entiers binaires associés aux deux nombres et au résultat :

15 << 1 ; 15 << 2 ; 15 << 4 ; 15 >> 1 ; 15 >> 2 ; 15 >> 4

Attention pour la suite :

- L'opérateur >> décale les bits vers la droite, fait donc disparaître les bits les uns après les autres jusqu'à ce que le nombre soit égal à 0, par exemple 15>>5 donne 0
- L'opérateur << agrandit infiniment le nombre. Toutefois, nous travaillons avec des images, chaque pixel est un triplet d'entiers codés sur 8 bits. Lorsque l'on applique l'opérateur << à un pixel (pas à une liste d'entier), il y a un overflow avec << qui permet d'obtenir (par chance) le bon résultat car l'overflow tronque les bits en trop à gauche 😊

```
>>> bin(15)
'0b1111'
>>> bin(15<<5)
'0b111100000'
>>> bin(np.uint8(0b111100000))
'0b11100000'
```

Pour simplifier la suite, et même si vous n'avez pas compris cette remarque, vous créerez dans les deux prochaines fonctions un array avec la commande :

```
Px_New = np.array([0,0,0],dtype="uint8")
```

Cela permettra de reproduire avec ces fonctions, ce qu'il se passera sur les pixels de l'image.

Question 8: Créer la fonction *f_PxFaiblesForts(Px)* créant un pixel *Px_New*, remplaçant ses 3 valeurs par les valeurs du pixel *Px* après décalage de leurs 4 bits de poids faibles sur leurs 4 bits de poids fort, et renvoyant ce nouveau pixel

Question 9: Créer la fonction *f_PxFortsFaibles(Px)* créant un pixel *Px_New*, remplaçant ses 3 valeurs par les valeurs du pixel *Px* après décalage de leurs 4 bits de poids forts sur leurs 4 bits de poids faibles, et renvoyant ce nouveau pixel

Vérifiez :

```
>>> f_PxFaiblesForts([255,0,127])
array([240,  0, 240], dtype=uint8)
>>> f_PxFortsFaibles([255,0,127])
array([15,  0,  7], dtype=uint8)
```

Pour les deux questions suivantes, vous vous inspirerez de vos fonctions *f_ImBitsFortsForts_n* et *f_ImBitsFaiblesFaibles_n*.

Question 10: Créer la fonction *f_ImBitsFaiblesForts(im)* récupérant sur l'image *im* les 4 bits de poids faibles de chaque couleur de chaque pixel, les affectant à une nouvelle image sur ses bits de poids forts (les autres étant laissés à 0), et renvoyant l'image obtenue

Question 11: Créer la fonction *f_ImBitsFaiblesFaibles(im)* récupérant sur l'image *im* les 4 bits de poids faibles de chaque couleur de chaque pixel, les affectant à une nouvelle image sur ses bits de poids faibles (les autres étant laissés à 0), et renvoyant l'image obtenue

Dernière mise à jour	Informatique	Denis DEFAUCHY
09/02/2022	7 - Matrices de pixels et images	TD 7-6 - Images cachées

Décodage des images fournies

Vous avez à votre disposition deux images nommées « Image_A_Decoder_1.bmp » et « Image_A_Decoder_2.bmp ».

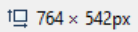
La première image contient, sur ses 4 bits de poids faibles, les 4 bits de poids forts d'une image cachée. La seconde image contient, sur ses 4 bits de poids faibles, les 4 bits de poids faibles d'une image cachée.

Question 12: A l'aide des fonctions créées précédemment, créer et afficher une image « Im_Decodee_Forts » ne contenant que les 4 bits de poids forts de l'image cachée

Question 13: A l'aide des fonctions créées précédemment, créer et afficher une image « Im_Decodee_Faibles » ne contenant que les 4 bits de poids faibles de l'image cachée

Question 14: En remarquant la propriété de la somme de deux entiers dont les codes binaires associés sont *0b00001111* et *0b11110000*, recréer très simplement l'image cachée entière « Im_Decodee » et l'afficher

Insertion par vous-même d'une image secrète dans une autre image

Ouvrez les images Eleve_Masque_1, Eleve_Masque_2 et Eleve_A_Cacher dans Paint. Collez n'importe quelle impression d'écran dans ces images et redimensionnez-les afin qu'elles tiennent dans le cadre de l'image initialement ouverte. Autrement dit, ATTENTION ! Ne pas changer les dimensions de l'image (nombres de pixels en ligne et colonne). En bas de la fenêtre de Paint, vous devriez voir  apparaître ces informations :

Si les dimensions ont quelque peu évolué après collage, déplacez les bordures de l'image (cadre) afin d'avoir les mêmes dimensions 764 X 542.

Enregistrez les modifications.

Les masques seront les images servant de support (poids forts visibles) sur lesquels vous mettrez sur les poids faibles des moitiés de l'image à cacher.

Question 15: Créer les fonctions `f_ImBitsFortsFaibles` et `f_ImBitsFortsForts` sur le même principe que les fonctions `f_ImBitsFaiblesForts` et `f_ImBitsFortsFaibles`

Question 16: A l'aide des fonctions créées précédemment, créer deux images Eleve_A_Decoder_1 et Eleve_A_Decoder_2 en respectant le principe que j'ai respecté pour créer les images de ce sujet, et donnez-les à un autre élève pour décodage

Rappel : pour enregistrer une image *im* sous le nom « im.bmp », utiliser la syntaxe :

```
plt.imsave( "im.bmp", im)
```