Introduction à la stéganographie

Ce mini-projet a pour objectif de retrouver un contenu (texte puis image) caché dans une image. Au départ, les caractéristiques du contenu caché (taille, nombre de bits utilisés pour dans les pixels pour le stocker) seront codées en dur, puis progressivement rendues paramétrables.

Fonctionnalité 1 : lecture d'un texte caché, terminé par un délimiteur

Spécifications

Dans cette question, on supposera que le texte caché dans l'image respecte les spécifications suivantes :

- le texte contient uniquement des caractères présents dans la table ASCII
- un caractère particulier, appelé délimiteur, indique la fin du texte (il ne fait pas partie du texte)
- chaque caractère du texte est stocké sous la forme de code ASCII
- les codes ASCII sont stockés les uns à la suite des autres
- chaque code ASCII sont représenté en binaire naturel sur 7 bits
- les bits correspondant des codes ASCII successifs sont stockés en altérant le dernier bit de chaque composante de chaque pixel de l'image
- les pixels sont altérés en partant du coin supérieur gauche et en les parcourant dans le sens naturel de lecture : de gauche à droite et de haut en bas. Exemple avec une image 3 x 4 :

pixel1	pixel2	pixel3
pixel4	pixel5	pixel6
pixel7	pixel8	pixel9
pixel10	pixel11	pixel12

- les composantes de chaque pixel sont altérées dans l'ordre naturel R, G, B

Exemple

Considérons que l'on cacher du texte "LE" avec le délimiteur de fin de texte "*" dans l'image 3 x 4 ci-dessus :

- code ASCII de L : 76 en décimal => 1001100 en binaire naturel sur 7 bits
- code ASCII de E : 69 => 1000101
- code ASCII de * : 42 => 0101010
- contenu à cacher : 10011001000101010101010
- stockage du contenu dans l'image (les bits notés X ne sont pas altérés) :

(XXXXXXX1,XXXXXXX0,XXXXXXX0)	(XXXXXXX1,XXXXXXX1,XXXXXXX0)	(XXXXXXX0,XXXXXXX1,XXXXXXX0)
(XXXXXXX0,XXXXXXX0,XXXXXXX1)	(XXXXXXX0,XXXXXXX1,XXXXXXX0)	(XXXXXXX1,XXXXXXX0,XXXXXXX1)
(XXXXXXX0,XXXXXXX1,XXXXXXX0)	(XXXXXXXX,XXXXXXXX,XXXXXXXX)	(XXXXXXXX,XXXXXXXX,XXXXXXXX)
(XXXXXXXX,XXXXXXXX,XXXXXXXX)	(XXXXXXXX,XXXXXXXX,XXXXXXXX)	(XXXXXXXX,XXXXXXXX,XXXXXXXX)

Travail demandé

Implémentez une fonction :

```
getHiddenTextWithDelimiter(image, delimiter)
```

qui renvoie le texte caché dans l'image `image` (instance de PIL.Image). Le texte est terminé par le délimiteur `delimiter` (paramètre facultatif, valeur par défaut : "\0"). Le délimiteur ne doit pas faire partie du texte retourné.

Astuce : étant donnée une image `i`, l'appel `list(i.getdata())` renvoie tous les pixels de `i` sous la forme d'une liste de tuples.

Tests

```
image = Image.open('hiddenText1.png')
print( getHiddenTextWithDelimiter(image) )
```

Doit aboutir à l'affichage du texte : "Hello, World!" (hiddenText1.png étant l'image du même nom fournie sur l'ENT).

Identifiez le texte caché dans l'image hiddenText2.png fournie. Le caractère de terminaison utilisé est "\0".

Fonctionnalité 2 : lecture d'un texte caché, dont la taille est donnée en entête

Spécifications

On suppose cette fois que les premiers pixels de l'image sont utilisés pour stocker la taille du texte comme suit :

- la taille est représentée en binaire naturel sur n bits
- le dernier bit des n premières composantes (en parcourant les pixels puis les composantes comme dans la fonctionnalité 1) est utilisé pour stocker les n bits codant la taille.

Les composantes suivantes sont ensuite utilisées comme dans la fonctionnalité n°1 pour stocker le texte lui-même.

Exemple

Considérons que l'on cache le texte "LE" dans l'image 3 x 4 décrite dans la fonctionnalité 1. On suppose que l'on utilise 4 bits pour stocker la taille du texte (taille maximale du texte : 15) :

- taille du texte : 2 en décimal => 0010 en binaire naturel sur 4 bits
- code ASCII de L : 76 en décimal => 1001100 en binaire naturel sur 7 bits
- code ASCII de E : 69 => 1000101

- contenu à cacher : 00101001100 1000101
- stockage du contenu dans l'image (les bits notés X ne sont pas altérés) :

(XXXXXXX0,XXXXXXX0,XXXXXXX1)	(XXXXXXX0,XXXXXXX1,XXXXXXX0)	(XXXXXXX0,XXXXXXX1,XXXXXXX1)
(XXXXXXX0,XXXXXXX0,XXXXXXX1)	(XXXXXXX0,XXXXXXX0,XXXXXXX0)	(XXXXXXX1,XXXXXXX0,XXXXXXX1)
(XXXXXXXX,XXXXXXXX,XXXXXXXX)	(XXXXXXXX,XXXXXXXX,XXXXXXXX)	(XXXXXXXX,XXXXXXXX,XXXXXXXX)

(XXXXXXXX,XXXXXXXX,XXXXXXXX)	(XXXXXXXX,XXXXXXXX,XXXXXXXX)	(XXXXXXXX,XXXXXXXX,XXXXXXXX)

Travail demandé

Implémentez une fonction :

```
getHiddenTextOfLength(image, nbBitsForLength)
```

qui renvoie le texte caché dans l'image `image` (instance de PIL.Image), dont la taille est stockée dans un entier codé sur `nbBitsForLength` bits, caché dans les premiers pixels de `image`.

Conseil : une fois une première version de `getHiddenTextOfLength()` fonctionnelle, essayez d'identifier les parties communes à `getHiddenTextOfLength()` et `getHiddenTextWithDelimiter()`, et de répartir ce code dans des sous-fonctions.

Tests

```
image = Image.open('hiddenText3.png')
print( getHiddenTextOfLength(image, 4) )
```

Doit aboutir à l'affichage du texte : "Hello, World!".

Identifiez le texte caché dans l'image hiddenText4.png fournie. La taille du texte est codée sur 16 bits.

Fonctionnalité 3 : lecture d'une image cachée, dont les dimensions sont données en entête

Spécifications

De manière similaire à la fonctionnalité précédente, les premiers pixels de l'image sont utilisés pour stocker les dimensions de l'image cachée. Chaque dimension est stockée sur n bits.

- le dernier bit des n premières composantes est utilisé pour stocker la largeur de l'image cachée
- puis, le dernier bit des n composantes suivantes est utilisé pour stocker la hauteur de l'image cachée
- enfin, le dernier bit des composantes suivantes est utilisé pour stocker les pixels de l'image cachée : de gauche à droite, de haut en bas, chaque composante étant stockée dans l'ordre R, G, B.
  8 bits sont utilisés pour stocker chaque composante de l'image cachée.

Exemple

Supposons une image A ayant les caractéristiques suivantes :
- ses dimensions sont 100x200
- ses deux premiers pixels sont (1,2,3) et (4,5,6)

Si on cache l'image A dans l'image B en utilisant 8 bits pour stocker la taille de A, alors les bits cachés dans les composantes des premiers pixels de B seront :

01100100110010000000000100000010000000110000100...

En effet :
- 01100100 correspond à la dimension 100 (largeur)
- 11001000 correspond à la dimension 200 (hauteur)
- 00000001 correspond à la composante R du pixel 1 de l'image cachée : 1
- 00000010 correspond à la composante G du pixel 1 de l'image cachée : 2
- 00000011 correspond à la composante B du pixel 1 de l'image cachée : 3

- 00000100 correspond à la composante R du pixel 2 de l'image cachée : 4
- … et ainsi de suite

Les premiers pixels de l'image B seraient :

- (XXXXXXX0,XXXXXXX1,XXXXXXX1)
- (XXXXXXX0,XXXXXXX0,XXXXXXX1)
- (XXXXXXX0,XXXXXXX0,XXXXXXX1)
- (XXXXXXX1,XXXXXXX0,XXXXXXX0)
- (XXXXXXX1,XXXXXXX0,XXXXXXX0)
- (XXXXXXX0,XXXXXXX0,XXXXXXX0)
- … et ainsi de suite

Travail demandé

Implémentez une fonction :

`getHiddenImage(image, nbBitsForSize)`

qui renvoie l'image cachée dans l'image `image` (instance de PIL.Image), dont la taille est stockée dans deux entiers codés sur `nbBitsForSize` bits, cachés dans les premiers pixels de `image`.

Tests

L'exécution du code suivant :

```
image = Image.open('hiddenImage1.png')
hidden = getHiddenImage(image, 8)
hidden.show()
```

Doit afficher l'image :

Identifiez l'image cachée dans l'image hiddenImage2.png fournie. La dimension de l'image cachée est codée sur 8 bits.

Fonctionnalité 4: lecture d'une image cachée, avec paramétrage du nombre de bits de poids faible utilisés

Spécifications

Le format est quasiment identique à celui utilisé pour la fonctionnalité 3. Seule modification : les k derniers bits des composantes sont utilisés pour stocker l'information cachée, au lieu d'un seul comme précédemment.

Si on reprend l'exemple des images A et B donné pour illustrer la fonctionnalité 3, les bits cachés dans les composantes des premiers pixels de B sont :

01100100110010000000001000000100000011000001 00...

Avec k = 2 bits, les premiers pixels de l'image B seraient de la forme :

- (XXXXXX01,XXXXXX10,XXXXXX01)
- (XXXXXX00,XXXXXX11,XXXXXX00)
- (XXXXXX10,XXXXXX00,XXXXXX00)
- (XXXXXX00,XXXXXX00,XXXXXX01)
- (XXXXXX00,XXXXXX00,XXXXXX00)
- (XXXXXX10,XXXXXX00,XXXXXX00)
- … et ainsi de suite

Travail demandé

```
getHiddenImageInLastBits(image, nbBitsForSize, nbLastBits)
```

qui renvoie l'image cachée dans l'image `image` (instance de PIL.Image), dont la taille est stockée dans deux entiers codés sur `nbBitsForSize` bits, cachés dans les premiers pixels de `image`.

Les informations cachées (taille et pixels) se trouvent dans les `nbLastBits` derniers bits des composantes des pixels de image.

Conseil : comme pour la fonctionnalité n°2, une fois `getHiddenImageInLastBits()` fonctionne, vous pourrez noter que `getHiddenImage()` n'est qu'un cas particulier d' `getHiddenImageInLastBits()` avec `nbLastBits = 1`. On peut donc remplacer tout le code de `getHiddenImage()` par un appel à getHiddenImageInLastBits() avec les arguments appropriés.

Tests

L'exécution du code suivant :

```
image = Image.open('hiddenImage3.png')
hidden = getHiddenImageInLastBits(image, 8, 2)
hidden.show()
```

Doit afficher l'image :

Identifiez l'image cachée dans l'image hiddenImage4.png fournie. Chaque dimension de l'image cachée est codée sur 16 bits. Les 2 derniers bits de chaque composante de l'image support ont été utilisés pour stocker l'information cachée.