BASES DE LA PROGRAMMATION (2)

TP 3 & 4 – Stéganographie

Adrien Guille - R2.02 - Université Lumière Lyon 2

Exercice 1 – Conversion de base (approche itérative)

Usuellement, un nombre s'écrit de la droite vers la gauche. Les chiffres qui le composent sont indicés à partir de la position 0. Durant les prochaines séances de TP nous auront besoin de basculer entre deux systèmes de numération, en base 10 (système décimal) et en base 2 (système binaire). Ci-après, l'écriture et la décomposition de l'entier 130 en base 10, et la même chose en base 2 :

$$(130)_{10} : 1 \times 10^2 + 3 \times 10^1 = 100 + 30 \leftrightarrow (10000010)_2 : 1 \times 2^7 + 1 \times 2^1 = 128 + 2$$

- Importer la bibliothèque `numpy` et la renommer `np`. Écrire la fonction `to_binary` qui reçoit un entier (entre 0 et 255) codé en base 10 et retourne son codage en base 2 sur 8 bits. Le codage en base 2 doit être un vecteur de type `np.array` de taille 8. Attention, ce vecteur doit se lire de gauche à droite, afin de respecter l'ordre naturel en Python.
  - **Principe** : l'algorithme à programmer repose sur une boucle qui calcule les puissances successives de 2, à commencer par la plus grande ; à vous de trouver la suite.
  - **Astuces** : la fonction `range` peut recevoir un pas négatif ; la fonction `np.flip` permet d'inverser l'ordre des valeurs dans un vecteur.
- Écrire la fonction `to_decimal` qui reçoit un vecteur tel que celui retourné par la fonction précédente et retourne l'entier correspondant codé en base 10.

Exercice 2 - Chargement et visualisation d'une image

- Importer la fonction `open` de la bibliothèque `PIL.Image`.
- Charger l'image `burger.png` avec la fonction `open`. Cette fonction retourne un tenseur à 3 dimensions, qu'il faut convertir en `np.array`. Afficher la taille de chaque dimension avec la fonction `np.shape`.
- Importer la fonction `imshow` de la bibliothèque `matplotlib.pyplot` et l'utiliser pour visualiser l'image d'après ce tenseur.
- Charger et visualiser une des images `burger_mystery_x.png`.

Exercice 3 - Manipulation des images

- Écrire la fonction `remove_lsd` qui reçoit un tenseur décrivant une image et un entier `n` (entre 1 et 8) et retourne un nouveau tenseur, où les couleurs ont été altérées via leur codage en base 2 : les `n` premiers bits (*least significant digits*, *lsd*) sont tous fixés à 0.
  - Visualiser les images retournées par la fonction pour différentes valeurs de `n`.
- Écrire la fonction `remove_msd` qui reçoit un tenseur décrivant une image et un entier `n` (entre 1 et 8) et retourne un nouveau tenseur, où les couleurs ont été altérées via leur codage en base 2 : les `n` derniers bits (*most significant digits*, *msd*) sont tous fixés à 0.
  - Visualiser les images retournées par la fonction pour différentes valeurs de `n`.
- Écrire la fonction `decode_image` qui reçoit un tenseur et en extrait un nouveau, tel que les 4 bits les plus forts du tenseur de sorties correspondent aux 4 bits les plus faibles du tenseur d'entrée. Les 4 bits les plus faibles du tenseur de sortie sont fixés à 0.

Exercice additionel

- Réécrire les fonctions `to_binary` et `to_decimal` de manière récursive.
- Écrire la fonction `hide_image` qui reçoit 2 tenseurs décrivant 2 images de même taille et qui retourne une nouvelle image avec la première cachée dans la seconde.