

1 Introduction

La recherche d'algorithmes de compression est ancienne en informatique, avec deux buts visés :

- gagner de l'espace en réduisant l'espace de stockage;
- gagner du temps en réduisant le temps de transfert des données.

Les progrès technique ont permis d'améliorer notablement les capacités de stockage et les débits des liaisons mais avec le développement d'Internet, Les données numériques sont de plus en plus nombreuses, , on parle de Big Data. Ces données jouent un rôle de plus en plus important pour l'analyse et la prise de décision (data mining), le divertissement (flux video...) ou l'entraînement des algorithmes d'apprentissage (machine learning). Il donc est important de disposer de bons algorithmes de compression.

En informatique, toutes les données sont représentées au plus bas niveau sous forme de bits, huit bits formant un octet. Un **algorithme de compression** prend en entrée un flux d'octets de taille B et renvoie un flux d'octets compressés de taille C(B). Il est d'autant plus performant que le **ratio de compression** $\frac{C(B)}{B}$ est petit. Évidemment, un tel algorithme est utile seulement si on connaît un **algorithme de décompression** capable de restituer le flux d'octets initial à partir du flux compressé.

On distingue deux catégories d'algorithmes de compression :

- Les algorithmes de compression sans perte d'information dont nous étudierons un exemple dans ce mini-projet et qui exploitent des redondances dans les données sources pour réduire l'espace (images, textes...)
- Les **algorithmes de compression avec perte** pour les informations traitées par nos sens : une perte d'information est tolérée si elle ne réduit pas de façon significative la qualité de la perception (d'une image, d'un son ...)

Plan du projet:

- Partie A: développer des outils de conversion de textes, d'images, d'entiers en flux d'octets.
- Partie B : découverte de l'algorithme de compression RLE.

2 Cahier des charges

- 1. Les programmes de chaque partie doivent être rassemblés dans des fichiers différents nommés partieA.py, partieB.py et partieC.py.
- **2.** Chaque fonction de partieA.py ou partieB.py doit passer le test correspondant dans les programmes de tests test_partieA.py et test_partieB.py fournis sinon cela doit être mentionné dans le code source sous forme de commentaires.
- **3.** Les réponses aux questions qui ne nécessitent pas de code doivent être fournies dans le code source sous forme de commentaire en les préfixant par le numéro de la question.
- **4.** Chaque fonction doit être documentée avec une docstring.
- 5. Les parties les moins évidentes du code doivent être commentées de façon pertinente.
- **6.** *Un code client permettant de tester le programme ou les fonctions principales doit être fourni.*



3 Partie A: boîte à outils

L'objectif de cette partie est de nous outiller avec des fonctions de conversions (dans les deux sens) de textes (type str) ou d'images (avec le module PIL) en flux d'octets (de type bytes) qui seront compressés puis décompressés par l'algorithme présenté en partie B.

- 1. Extraire l'archive DM-Compression.zip dans un répertoire du même nom. Elle contient :
 - les deux squelettes de code partieA.py et partieB.py.
 - les deux fichiers de tests test_partieA.py et test_partieB.py.
 - des ressources (textes, images ...) utilisées pour tester le code
- 2. Renommer le répertoire en DM-Compression_Eleve1_Eleve2 avec les noms des membres du groupe séparés par des tirets bas (surtout pas d'espace). Par la suite on appellera ce répertoire le répertoire de base du projet.
- **3.** Exécuter le script partie A. py. Si le module PIL n'est pas installé, une erreur s'affiche, il faut alors l'installer en suivant la documentation en ligne sur :

```
https://pillow.readthedocs.io/en/stable/installation.html
```

```
1 from PIL import Image
```

4. Compléter la fonction index_premiere_occurence(element, tab) en respectant la spécification donnée dans la docstring.

Vérifier si la fonction passe le test unitaire index_premiere_occurence dans test-partieA.py.

```
def index_premiere_occurence(element, tab):
    """
    Parametres :
        element de type quelconque (le même que les éléments de tab)
        tab un tableau d'éléments de même type
    Valeur renvoyée:
        l'indice de la première occurence de element dans tab
    """
```

5. Compléter la fonction hex_to_decimal(hexadecimal) en respectant la spécification donnée dans la docstring.

Vérifier si la fonction passe le test unitaire test hex to decimal dans test partieA.py.



6. Compléter la fonction decimal_to_hex(n) en respectant la spécification donnée dans la docstring. Vérifier si la fonction passe le test unitaire test decimal to hex dans test-partieA.py.

```
def decimal_to_hex(n):
    """
    Parametres :
        n de type int
    Valeur renvoyée:
        représentation de n en base 16 sous forme de chaîne de caractè
            res
            on rajoute un 0 à gauche si un seul chiffre en base 16
    """
    rep = ''
    #à compléter
```

7. partieA. py contient ensuite trois fonctions de conversion entre les types str des chaînes de caractères et bytes des flux/séquences d'octets qui seront manipulés par nos algorithmes de compression.

```
def str to bytes(chaine, encodage = 'utf8'):
   Paramètres :
       chaine de type str, une chaine de caractères
       encodage un paramètre de type str fixant l'encodage, par défaut
           utf8
   Valeur renvoyée :
       un flux d'octets de type bytes obtenu par encodage de chaine
   return chaine.encode(encoding = encodage)
def bytes to str(flux, encodage = 'utf8'):
   Paramètres :
       flux d'octets de type bytes
       encodage un paramètre de type str fixant l'encodage, par défaut
           utf8
   Valeur renvoyée :
       une chaine de caractères de type str obtenue par décodage de
           flux
   return flux.decode(encoding = encodage)
def hex_to_bytes(hexadecimal):
   0.00
   Paramètres :
       hexadecimal de type str représentant un nombre à 2 chiffres en
            base 16
   Valeur renvoyée :
       un octet de type bytes représentant le même nombre qu'
           hexadecimal
   H/H/H
```



```
return bytes.fromhex(hexadecimal)
```

Pour comprendre leur fonctionnement, on donne ci-dessous quelques exemples, disponibles également dans l'interpréteur en ligne Basthon. On peut remarquer que les caractères imprimables de la table ASCII (ordinal Unicode inférieur à 128) sont représentés tels quels dans le type bytes.

Pourquoi n'est-ce pas le cas pour le caractère accentué é dans l'exemple ci-dessous?

Pour simplifier, nous travaillerons dans ce DM uniquement avec des chaînes de caractères de la table ASCII codés sur un octet.

```
>>> [str_to_bytes(c) for c in ['1','a', 'A', 'é']]
[b'1', b'a', b'A', b'\xc3\xa9']
>>> [bytes_to_str(b) for b in [b'1', b'a', b'A', b'\xc3\xa9']]
['1', 'a', 'A', 'é']
>>> [hex_to_bytes(h) for h in ['00', '06', '0B', 'A1', 'FF']]
[b'\x00', b'\x06', b'\x0b', b'\xa1', b'\xff']
```

_____Complément _____

Les dernières fonctions outils fournies dans partiel. py permettent de convertir un fichier représentant une image en niveaux de gris (pixel sur un octet) en un flux d'octets de type bytes et réciproquement de convertir un flux d'octets en image (en précisant sa largeur et sa hauteur).

On utilisera ces fonctions comme des boîtes noires. Pour comprendre leur fonctionnement on donne ci-dessous un exemple d'exécution sur une image de 2×2 pixels représentant un damier : on extrait le flux d'octets de l'image source, on le transforme en remplaçant les octets par leur complémentaire à 255 puis on convertit ce flux de sortie en image.

Le test unitaire test_image fourni dans test_partieA.py permet de reproduire cet exemple.

```
In [7]: img = fichier_to_image('damier_2x2.png')
In [8]: img
Out[8]: <PIL.Image.Image image mode=L size=2x2 at 0x7FA724517340>
In [9]: img_bytes = image_to_bytes(img)
In [10]: img_bytes
Out[10]: b'\xff\x00\x00\xff'
In [11]: img_bytes_sortie = b''.join([(255 - img_bytes[k]).to_bytes(1, byteorder='little') for k in range(len(img_bytes))])
In [12]: img_bytes_sortie
Out[12]: b'\x00\xff\xff\x00'
In [13]: img_sortie = bytes_to_image(img_bytes_sortie, img.width, img.height)
In [14]: img_sortie
Out[14]: <PIL.Image.Image image mode=L size=2x2 at 0x7FA7240C4C40>
```





4	Partie B	: com	pression	RLE
---	----------	-------	----------	------------

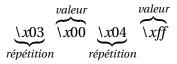
Comandána ant	
Complément	

^L Méthode

Run Length Encoding est un algorithme efficace pour compresser des données contenant de longues séquences de valeurs répétées.

La chaîne de caractères "ABBBAABBBB" peut ainsi être compressée en "1A3B2A4B".

Le principe est simple : on représente chaque séquence de caractères identiques par sa longueur suivie du caractère répété.



On fixe également une longueur maximale pour les répétitions : dans notre mini-projet, on choisit de les coder sur 1 octet soit au plus 255 valeurs consécutives.

 Fin de complément	

1. Compresser avec l'algorithme Run Length Encoding la séquence d'octets ci-dessous en notation hexadécimale. Donner la séquence compressé en notation décimale également.

2. Décompresser la séquence d'octets ci-dessous compressée avec l'algorithme Run Length Encoding :

 $\x02\xb1\x0a\x00\x01\xfc\x0f\x0a$

3. Ouvrir le script partieB.py. Quel est le rôle de la première instruction ci-dessous?

```
from partieA import *
```



Complément

Avant de traiter les questions suivantes, il faut se familiariser avec les objets de type bytes. Consulter le tutoriel Exemples_Operations_Bytes.pdf et ce notebook Basthon.

Fin de complément	

4. Compléter la fonction decompresse_rle(flux) en respectant la spécification donnée dans la docstring. Vérifier si la fonction passe le test unitaire test_decompresse_rle dans test_partieB.py.

5. Compléter la fonction compresse_rle(flux) en respectant la spécification donnée dans la docstring. Vérifier si la fonction passe le test unitaire test_compresse_rle dans test_partieB.py.

6. Compléter la fonction ratio_rle(flux, flux_compresse) en respectant la spécification donnée dans la docstring.

```
def ratio_rle(flux_compresse, flux):
    """
    Paramètre :
        flux de type bytes est un flux d'octets
        flux_compresse de type bytes est un flux d'octets
        Valeur renvoyée :
```



```
un nombre de type float représentant le ratio entre flux et
flux_compresse
"""
"à compléter"
```

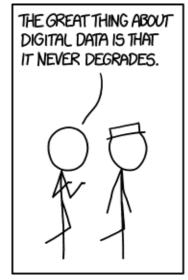
7. Compléter le script partieB.py avec un code client permettant de tester l'exemple fourni dans la page Wikipedia sur la compression Run Length Encoding.

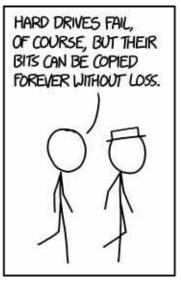
Compression RLE d'un texte

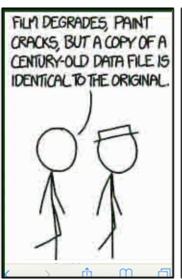
Flux d'octets compressé : b'\x0cW\x01B\x0eW\x03B\x17W\x01B\x0bW'

Ratio de compression : 0.215385

8. Compléter le script partieB.py avec un code client permettant de compresser l'image en niveau de gris enregistrée dans le fichier lena.png: extraction du flux d'octets représentant l'image, compression de ce flux, affichage du ratio, décompression et conversion du flux de sortie en image. On utilisera les fonctions fichier_to_image, image_to_bytes et bytes_to_image définies dans partieA.py.









https://www.explainxkcd.com/wiki/index.php/1683:_Digital_Data