UNIVERSITÉ DE REIMS CHAMPAGNE-ARDENNE

Licence Informatique
Info0603 – 2021/2022

Travaux dirigés et TP n° 3

Compression

Exercice 1 (La classe CodeurCA)
C'est une manière de définir une classe abstraite. Un Codeur crée un Binaire603 codé à partir d'un Binaire603 et peut bien sûr renvoyer le Binaire603 d'origine à partir du Binaire603 codé.

on créera une nouvelle classe héritant de CodeurCA à chaque nouvelle méthode de compression ou de chiffrement.

Les méthodes à surcharger :

* __init__(self)
* __str__(self) : (Un simple texte)
* __repr__(self) : (un appel au constructeur)
* binCode(self,monBinD :Binaire603)->Binaire603 : (renvoie le Binaire603 codé)
* binDecode(self,monBinC :Binaire603)->Binaire603 : (Renvoie le Binaire d'origine à partir du codé)

Chaque nouvelle classe aura bien sûr sa méthode démo plus ou moins adapté de celle de CodeurCA. Afin de bien comprendre la classe CodeurCA, construire la classe ChiffrementParDecalage :

```
def binCode(self,monBinD:Binaire603)->Binaire603:
    """
    >> ChiffreurParDecalage(2).binCode(Binaire603([1,2,3,4,255]))
    Binaire603([ 0x03, 0x04, 0x05, 0x06, 0x01])
def binDecode(self,monBinC:Binaire603)->Binaire603:
    """
    >> ChiffreurParDecalage(2).binDecode(Binaire603([1,2,3,4,255]))
    Binaire603([ 0xff, 0x00, 0x01, 0x02, 0xfd])
```

Exercice 2 (La classe Texte603)
Elle permet de faire le lien textuel avec Binaire603 notamment pour tester le chiffrement ou la compression de textes.

* Texte603 (constructeur à partir d'une chaine, d'un Texte603 ou d'un Binaire603)
* afficheCompressionParDico (une démo à voir en CM)
* exTexte603 (renvoie des exemples d'instances)
* toBinaire603

On vous demande de tester votre classe ChiffrementParDecalage sur un texte de votre choix.

Exercice 3 (Compression par répétition)
On vous demande de créer une classe CompressionSimpleParRepetition et de la tester sur un texte de votre choix. On pensera notamment à comparer les entropies des textes compressés et non compressés.

Exercice 4 (La classe Image603)
Elle permet de travailler sur des images et de tester ainsi des méthodes de compression et de chiffrement.

Les attributs : "Une Image603 contient un tableau de triplet d'octet représentant la couleur de chaque pixel"

— self.lg self.ht sont les dimensions de l'image

— self.coul un tableau couleurs de dimensions lg*ht. Ce tableau à deux dimensions est une liste et chaque couleur est un tuple de trois entiers de [0..255]

Les méthodes déjà écrites : Image603(lg,ht) Le constructeur avec les dimensions en paramètres.

— eq, str, repr

— affiche

— 'exImage603'Renvoie des exemples d'Image603

— 'iterXY' : Un itérateur des coordonnées de tous les pixels de l'image
— 'binXYCo' : renvoie un binaire codant complètement l'image
— 'dPalette' : renvoie un dictionnaire indexé par couleurs, contenant, en tupple, le classement et l'effectif de la couleur. (Par exemple im1.dPalette().[(255,0,0)]==(5,18) signifie que le rouge est à la sicième couleur la plus utilisée et qu'elle l'est 18 fois dans l'image603 im1.
— 'imgDepuisBinXYCo',
— 'imgDepuisBmp',
— 'imgDepuisSV1',
— 'imgDepuisSV2',
— 'imgDepuisSVRH',

Exercice 5 (Compression RLE, format PCX)
En reprennant l'exemple du cours et le format PCX https ://fr.wikipedia.org/wiki/PCX, créer une classe CompressionPCX et la tester sur une image à 256 couleurs.

Exercice 6 (Compression d'Huffman)
Créer la classe CompressionHuffman. On pourra éventuellement passer par les méthodes suivantes :

```
def dicoHuffmanDepuisArbre(arbre):
    """Renvoie les dictionnaires associant les étiquettes à leur codage d'Huffman (format txt)
    >> a=CompresseurHuffman.arbreDepuisListePonderee([("B",0.3),("L",0.2),("E",0.2),("I",0.1),("A",0.1
    >> CompresseurHuffman.dicoHuffmanDepuisArbre(a) ({'000': 'E', '0010': 'S', '0011': 'N',
'0100': 'A', '0101': 'T', '011': 'I', '10': 'B', '11': 'L'}, {'E': '000', 'S': '0010', 'N':
'0011', 'A': '0100', 'T': '0101', 'I': '011', 'B': '10', 'L': '11'})
    """
    >> CompresseurHuffman.construireDicoH(arbre1,d)
    >> print(d)
    '0': 'A', '10': 'B', '11': 'C'
    """
def arbreDepuisListePonderee(lp):
    """Transforme la liste de couple (Etiquette,Entropie) en un tuple modélisant un arbre.
    Un arbre pondéré est un tuple de la forme (Etiquette,pondération) ou (Arbre,pondération)
    >> CompresseurHuffman.arbreDepuisListePonderee([("A",0.2),("B",0.3),("C",0.4)])
    (((('B', 0.3), ('A', 0.2)), 0.5), ('C', 0.4)), 0.9)
    """
def codageHuffman(monBin,verbose=False):
    """Renvoie les dictionnaire associant les clés d'Huffman aux valeurs d'octets"
    >> CompresseurHuffman.codageHuffman(Binaire603([5,5,5,5,5,5,5,5,6,6,6,7,7,9]))
    ('00': 6, '01': 5, '10': 7, '11': 9, 6: '00', 5: '01', 7: '10', 9: '11')
    """
def binCode(self,monBin,verbose=False):
    "renvoie une chaine Binaire codée par Huffman"
def binDecode(self,binC,verbose=False):
    """renvoie une chaine Binaire decodée par Huffman
    >> monCodeur=CompresseurHuffman()
    >> monBin=Binaire603([6,6,6,6,6,5,5,5,5,6,6,6,7,8,9,8,8])
    >> monBinC=monCodeur.binCode(monBin)
    >> monBin==monCodeur.binDecode(monBinC)
    True
    """
```

Exercice 7
(Traveil personnel : Compression avec pertes) Lire et tester les feuilles Jupyter et les programmes crelatifs aux transformées de Fourrier et d'ondelettes.