Aix*Marseille université
Initiative d'excellence
Année 2018-19

Diplôme Inter-Universitaire
Enseigner l'informatique au lycée

Semaine 2 : TP 2

Codage des fichiers et des images

1 Codage du texte

lettre	codage fixe
a	00000
b	00001
c	00010
d	00011
e	00100
f	00101
g	00110
h	00111
i	01000
j	01001
k	01010
l	01011
m	01100
n	01101
o	01110
p	01111
q	10000
r	10001
s	10010
t	10011
u	10100
v	10101
w	10110
x	10111
y	11000
z	11001
espace	11010

lettre	codage variable
a	1010
b	0010011
c	01001
d	01110
e	110
f	0111100
g	0111110
h	0010010
i	1000
j	011111110
k	011111111001
l	0001
m	00101
n	1001
o	0000
p	01000
q	0111101
r	0101
s	1011
t	0110
u	0011
v	001000
w	011111111000
x	01111110
y	0111111111
z	01111111101
espace	111

Jouons un peu avec deux codages (donnés dans les tables ci-dessus) en binaire des lettres d'un texte utilisant les 26 lettres de l'alphabet latin plus l'espace. Le codage fixe est un codage type ASCII ou unicode, où la longueur du code est fixe et identique pour chaque lettre à coder. Le codage variable, comme son nom l'indique, code les caractères avec des codes de longueur variable. Par exemple le mot "patate" se code 01000 1010 0110 1010 0110 110 avec le codage variable et 01111 00000 10011 00000 10011 00100 avec le codage fixe. Il est à noter que les espaces entre les codes des différentes lettres sont présents uniquement pour la lisibilité et ne font pas partie du code.

Décidons de stocker les tables de codage dans des dictionnaires, associant à chaque caractère à coder, son code binaire représenté comme une chaîne de caractères '0' et '1'. Pour vous aider, nous avons déjà créé le dictionnaire `variable_table` pour vous : vous pouvez aller récupérer le fichier Python qui le contient sur Ametice. Vous allez cependant devoir créer vous-même le dictionnaire `fixed_table` qui contiendra la table pour le codage fixe. On remarque que les codes suivent une progression arithmétique de raison 1, si on considère

les chaînes de '0' et de '1' comme des codages binaires d'entiers positifs. Plutôt que d'écrire le dictionnaire à la main, on peut donc le générer automatiquement. Le fichier Python que vous avez téléchargé contient également des idées pour vous permettre de faire cela.

Question 1. Générer le dictionnaire `fixed_table` en complétant d'abord la fonction `increment` qui prend une chaîne de '0' et de '1' représentant un entier positif n, et qui renvoie la chaîne codant pour l'entier $n + 1$.

Utilisons désormais les tables de codage pour coder et décoder des textes.

Question 2. Écrire une fonction `encode` qui prend en arguments un message et une table de codage, et qui renvoie le message encodé. Lorsqu'on rencontre un caractère dont on n'a pas de code, on l'encode comme si c'était un espace. N'oubliez pas de tester vos fonctions, ici et dans toute la suite ! Vérifier par exemple que le codage variable du mot "decode" est égal au codage fixe de la chaîne "ozedw".

Question 3. Pour décoder un message, il est a priori plus simple de commencer par construire une table de décodage, qui consiste simplement à inverser les clés et leurs valeurs dans la table d'encodage. Écrire une fonction `build_decoding_table` qui prend une table de codage et renvoie la table de décodage associée.

Question 4. Il est très simple de décoder un message encodé avec le codage fixe, puisqu'il suffit de récupérer les cinq caractères '0' et '1', d'utiliser la table de décodage pour le décoder, et de continuer ainsi jusqu'à épuisement du message. En déduire une fonction `fixed_decode` qui prend un message codé en entrée et renvoie le message décodé en sortie.

Question 5. Le décodage du code variable est plus délicat. Expliquer d'abord pourquoi il existe *au plus* une façon de décoder toute chaîne de caractères '0' et '1'. En déduire une fonction `variable_decode` qui prend un message codé en entrée et renvoie le message décodé à l'aide du code variable en sortie.

Question 6. Encoder le mot 'information' à l'aide du code variable. Décoder les chaînes obtenues à partir de l'encodage obtenu, en inversant le 2ème caractère uniquement, puis le 3ème caractère uniquement, puis le 12ème caractère uniquement. Que remarquez-vous ? Donner une liste d'avantages et d'inconvénients des codes variables et fixes.

2 Fichiers

Le texte ou toute autre information est souvent stocké dans des fichiers. Python permet, comme nous l'avons déjà fait avec des fichiers csv par exemple, de lire le contenu de fichier texte (ou binaire) et d'écrire dans des fichiers. Profitons-en !

Question 7. Récupérer sur Ametice le fichier 'chiffre_variable.txt' qui contient le code variable d'un long texte. Le décoder en écrivant le message obtenu dans un nouveau fichier : on prendra soin de conserver intacts les retours à la ligne. Vous devriez reconnaître le texte obtenu...

Question 8. Encoder le texte obtenu à l'aide du codage fixe. Comparer la taille des trois fichiers obtenus.

3 Codage des images

On sait désormais coder un roman, mais pas une bande dessinée : il nous manque la possibilité de coder des images. Un format simple (mais gourmand en espace) consiste à représenter une image comme un tableau bidimensionnel (une matrice) : chaque élément du tableau est alors appelé un pixel. C'est le format *bitmap*. Le codage d'un pixel consiste à représenter la couleur de la zone correspondante de l'image. Pour ce faire, on utilise généralement trois entiers qui représentent les quantités (entre 0 et 255) de rouge, de vert et de bleu dans la couleur. Par exemple, une zone de l'œil de la Joconde est agrandie dans la figure ci-dessous : cette zone comporte 9 colonnes et 10 lignes de pixels. Deux de ces pixels sont détaillés à droite. Celui du haut a une couleur marron pâle qui est codée par le triplet (204, 164, 93) : la couleur est donc constituée de rouge à hauteur de $204/(204 + 164 + 93) = 44,25\%$. Ainsi, la couleur verte à 100% sera représentée par le triplet (0, 255, 0).

Question 9. Combien d'octets sont nécessaires pour stocker en format bitmap une image en couleurs de 1920 par 1200 pixels.

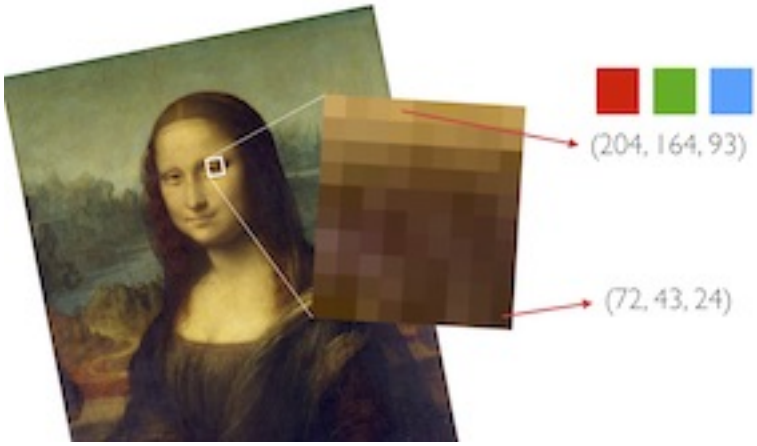

FIGURE 1 – Codage d'une image en format bitmap

C'est beaucoup pour une simple image... En pratique, on ne stocke donc que rarement les images en format bitmap : on utilise plutôt des formats *compressés* qui essaient d'épargner de la mémoire en profitant de redondances dans l'image ou en supprimant des détails invisibles à l'œil nu. Le format *jpeg* est un exemple de tel format d'image compressé. La compression jpeg est basée principalement sur l'utilisation du codage RLE et d'un codage de Huffman, similaire au codage variable que nous avons utilisé précédemment pour le texte. Le codage RLE permet de condenser des séquences de caractères identiques en les remplaçant par le caractère suivi du nombre d'occurrences successives : ainsi la séquence de pixels noir (B) et blanc (W)

$WWWWWWWWWWWWBWWWWWWWWWWWWBBB$
$WWWWWWWWWWWWWWWWWWWWWWWWBWWWWWWWWWWW$

sera codée à l'aide de $12W1B14W3B23W1B11W$.

Il existe encore d'autres façons de stocker une image, par exemple, les méthodes vectorielles qui consistent à coder les éléments géométriques constituant l'image, plutôt que de découper celle-ci en pixels colorés.

Étudions les encodages bitmap et vectoriels en plus grand détail en simplifiant un peu la situation. Supposons ainsi qu'on souhaite effectuer des dessins sur une grille carrée comprenant $64 \times 64 = 4096$ cases (ou pixels).

On suppose que les pixels de la grille sont numérotés selon un ordre conventionnel (par exemple de gauche à droite et de haut en bas). Dans une représentation bitmap, un dessin (en noir et blanc) est représenté par une suite de bits $b_1 \cdots b_n$ dont le i-ième bit b_i est égal à 0 si le i-ième pixel du dessin est blanc et 1 s'il est noir.

Question 10. Combien de bits sont nécessaires pour coder une image telle que la maison ci-dessous sous ce format ?

Étudions ensuite un encodage vectoriel, en supposant que les dessins sont composés de trois types d'éléments uniquement : des segments et des rectangles dont les côtés sont parallèles aux axes et qui pourront être pleins ou vides. Ainsi, le dessin de maison ci-dessus est composé d'un rectangle vide $ADJI$, d'un rectangle plein $BCFE$ et de deux segments GK et KH.

Chaque point peut être repéré par un couple d'entiers (x, y) où x et y sont compris entre 0 et 63. On supposera que x et y sont écrits en binaire.

Question 11. Combien faut-il de bits pour stocker une coordonnée comprise entre 0 et 63 ?

On choisit alors de coder les trois types d'éléments de la façon suivante :

— Un segment $[AB]$ dont les extrémités ont pour coordonnées (x_A, y_A) et (x_B, y_B) sera représenté par 4 octets $01x_A$ $01y_A$ $01x_B$ $01y_B$.

— Un rectangle vide $ABCD$ dont deux sommets opposés sont A et C sera représenté par 4 octets $10x_A$ $10y_A$ $10x_C$ $10y_C$.

— Un rectangle plein $ABCD$ dont deux sommets opposés sont A et C sera représenté par les 4 octets $11x_A$ $11y_A$ $11x_C$ $11y_C$.

Un dessin est représenté par la suite des représentations des éléments qui le compose.

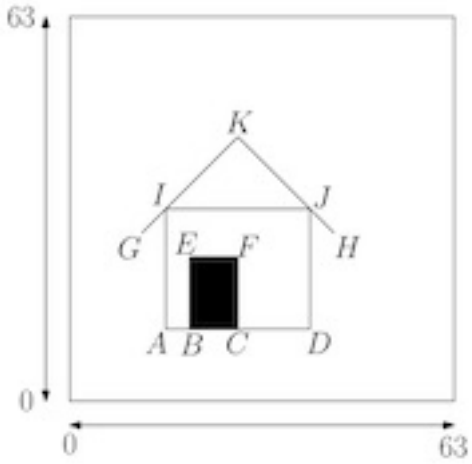

FIGURE 2 – Dessin d'une maison

Question 12. Combien faut-il de bits pour représenter le dessin de maison ? Combien cela fait-il en octets ?

Question 13. Indiquer quel dessin est représenté par le codage suivant 10001111 10001111 10101111 10101111 01001111 01001111 01101111 01101111 01001111 01101111 01101111 01001111

On souhaite enrichir les codages possibles en représentant directement des lignes brisées $A_1A_2 \dots A_k$ (c'est-à-dire des réunions de segments $[A_1A_2], [A_2A_3], \dots, [A_{k-1}A_k]$).

Question 14. Suggérer un moyen d'étendre le codage ci-dessus pour représenter de telles lignes brisées ? Combien faudrait-il alors de bits pour représenter le dessin de maison ?