Problème 1. Codage d'images en binaire

On s'intéresse dans cet exercice au codage sans perte des données. Dans tout l'exercice, on prendra l'exemple du codage d'images en *niveaux de gris* par des matrices d'entiers entre 0 et $k-1$, où 0 représente un pixel noir, et $k-1$ un pixel blanc. L'objectif est de comprendre comment coder une telle matrice par une suite de bits sur un ordinateur, et minimisant l'espace utilisé.

On note $\{0,1\}^\star$ l'ensemble des mots sur l'alphabet $\{0,1\}$, c'est-à-dire l'ensemble des suites finies de 0 et de 1. De même, pour tout ensemble X, on définit $X^\star$ comme l'ensemble des suites finies d'éléments de X.

Définition. Soit X un ensemble fini. Un *code (binaire)* pour X est une fonction $c : X \to \{0,1\}^\star$ qui à chaque élément $x \in X$ associe un mot sur l'alphabet $\{0,1\}$. Tout code pour X peut être étendu en une fonction $c^\star : X^\star \to \{0,1\}^\star$ définie par $c^\star([x_1, x_2, \dots, x_n]) = c(x_1) \cdot c(x_2) \cdots c(x_n)$ où le point désigne la concaténation. Les éléments $x \in X$ sont appelés des *symboles* et les mots $c(x)$ pour $x \in X$ sont des *mots de code*.

Dans le problème, les ensembles X considérés seront de la forme $\{0, \dots, k-1\}$ pour un entier $k > 0$ donné.

Représentations dans les algorithmes. Une matrice sera représentée par une liste de listes. Par exemple, la matrice $\left(\begin{smallmatrix}1 & 2\\ 3 & 0\end{smallmatrix}\right)$ sera représentée par la liste `[[1,2],[3,0]]`. Les mots sur l'alphabet $\{0,1\}$ seront représentés par des chaînes de caractères. Par exemple, le mot 01001 sera représenté par la chaîne `'01001'`. Les suites finies de symboles de $X = \{0, \dots, k-1\}$ seront représentés par des listes. Par exemple, la suite $[1,3,5,7]$ sera représentée par la liste `[1, 3, 5, 7]`. Un code binaire sera représenté par un dictionnaire qui associe à chaque symbole $x \in X$ son code sous forme de chaîne de caractères. Par exemple, le code donné en introduction de la partie 1.1 ci-dessous est représenté par le dictionnaire `{0:'00', 1:'01', 2:'10', 3:'11'}`.

1.1 Codage uniforme

Dans le codage uniforme, on associe à chaque symbole de X une suite de bits de même taille. Par exemple, un code uniforme possible pour l'ensemble $\{0,1,2,3\}$ est donné par

$$\begin{aligned} c(0) &= 00,\\ c(1) &= 01,\\ c(2) &= 10,\\ c(3) &= 11. \end{aligned}$$

On aura donc par exemple $c^\star([0,1,3]) = 000111$ et $c^\star([2,1]) = 1001$. On remarque que ce code revient à associer à chaque entier k l'écriture en base 2 de k, sur 2 bits.

Question 1.1.

(a) Écrire un algorithme `base_2` qui prend en entrée un entier positif k et une longueur (entière positive) l et retourne la chaîne de caractère donnant la représentation binaire de k sur l bits. Si l est trop petit pour pouvoir représenter k sur l bits, on tronquera la représentation de k aux l bits de poids faible. Par exemple `base_2(3,4) = '0011'`, `base_2(9, 4) = '1001'` et `base_2(17, 4) = '0001'` (car 17 s'écrit 10001 en binaire).

(b) Écrire un algorithme `code_uniforme` qui étant donné un entier k non nul, construit le code uniforme $c : \{0, \dots, k-1\} \to \{0,1\}^\star$ tel que $c(i)$ est la représentation binaire de i. *Attention : le code doit être uniforme, c'est-à-dire que tous les mots de code doivent avoir la même taille. On pourra écrire un algorithme qui calcule la longueur de la représentation binaire d'un entier.*

(c) Écrire un algorithme `inverse_code` qui prend en entrée un code c et renvoie le dictionnaire inversé, qui associe à chaque mot de code le symbole correspondant.

Question 1.2. Pour coder une *matrice* M, on lit ses entrées ligne par ligne, puis on code la liste obtenue. Par exemple, la matrice $\left(\begin{smallmatrix}1 & 2\\ 3 & 0\end{smallmatrix}\right)$ sera représentée par la liste $(1,2,3,0)$, codée avec le codage donné en début de partie par le mot 01101100. L'objectif de cette question est d'écrire des fonctions d'encodage et de décodage pour passer d'une matrice à son code binaire, et inversement.

(a) Écrire un algorithme `racine_entiere` qui étant donné un entier n calcule la racine carrée de n si n est un carré parfait, et renvoie 0 sinon.

(b) Écrire un algorithme `matrice_vers_liste` qui prend en entrée une matrice carrée (représentée par une liste de listes) et retourne sa représentation sous forme de liste, et un algorithme `liste_vers_matrice` qui fait la transformation inverse. On pourra supposer pour l'algorithme `matrice_vers_liste` que toutes les listes font la même taille, et pour l'algorithme `liste_vers_matrice` que la longueur de la liste donnée en entrée est un carré parfait.

(c) Écrire un algorithme `encode_matrice` qui prend en entrée une matrice carrée d'entiers positifs (représentée par une liste de listes) et un code uniforme c et renvoie la chaîne de caractères qui encode la matrice avec le code c.

(d) Écrire un algorithme `decode_vers_matrice` qui prend en entrée une chaîne de caractères w et un code uniforme c représenté par un dictionnaire, et renvoie la matrice carrée M dont l'encodage avec le code c est w.

(e) Soit M une matrice de dimensions $n \times n$ qui contient des entiers entre 0 et $k-1$. Quelle est la taille du mot obtenu en encodant la matrice M par le code uniforme décrit précédemment ? Peut-on trouver une représentation plus compacte en utilisant un autre code uniforme ?

1.2 Codage non uniforme

Pour améliorer les performances du code et obtenir un codage plus compact des images, on peut utiliser un code non uniforme, c'est-à-dire un code dans lequel les mots de code n'ont pas tous la même longueur. Par exemple, on peut coder les symboles de l'ensemble $\{0,1,2,3\}$ par le code c suivant : $c(0) = 0$, $c(1) = 10$, $c(2) = 110$ et $c(3) = 101$. On aura alors $c^\star([2,0,1]) = 110010$ par exemple.

On dit qu'un code est *uniquement décodable* si pour tout mot $w \in \{0,1\}^\star$, il existe une unique suite $S \in X^\star$ telle que $c^\star(S) = w$, autrement dit si la fonction $c^\star : X^\star \to \{0,1\}^\star$ est injective.

Question 1.3. Montrer que le code donné ci-dessus n'est pas uniquement décodable.

Un mot u est préfixe d'un mot v, noté $u \preceq v$, si u est égal au début de v : si u est de longueur k, alors $u_i = v_i$ pour tout $i < k$ (où u_i est la i-ème lettre de u, et v_i la i-ème lettre de v). Un code est dit préfixe si pour tout couple de symboles distincts $(x,y) \in X^2$, $c(x) \not\preceq c(y)$ et $c(y) \not\preceq c(x)$. Par exemple, le code donné ci-dessus n'est pas préfixe car $c(1) \preceq c(3)$.

De même un mot u est suffixe d'un mot v si le mot u est égal à la fin du mot v, et un code c est dit *suffixe* si les mots de codes ne sont jamais suffixes l'un de l'autre.

Question 1.4.

(a) Montrer qu'un code préfixe est uniquement décodable. Montrer que c'est également le cas d'un code suffixe.

(b) Écrire un algorithme de décodage qui prend en entrée un mot w et un code préfixe c et retourne une liste L de symboles de X telle que $c(L) = w$. Il n'est pas nécessaire de traiter une éventuelle erreur de décodage, c'est-à-dire le cas où il n'existe pas de suite L telle que $c(L) = w$.

(c) Analyser la complexité de votre algorithme en fonction du nombre d'éléments dans le dictionnaire et de la taille du mot w.

Question 1.5. Dans cette question, on étudie les liens qui existent entre le fait qu'un code soit préfixe (ou suffixe), et le fait qu'il soit uniquement décodable.

(a) Montrer que le code suivant $c : \{0,\dots,4\} \to \{0,1\}^\star$ est uniquement décodable, mais qu'il n'est ni suffixe ni préfixe : $c(0) = 00$; $c(1) = 10$; $c(2) = 11$; $c(3) = 110$; $c(4) = 100$.

(b) Un code c est *surjectif* si pour tout mot $w \in \{0,1\}^\star$, il existe (au moins) une suite L telle que $c^\star(L) = w$. Montrer qu'un code surjectif uniquement décodable est à la fois suffixe et préfixe.

1.3 Codage de Huffman

Le codage de Huffman est un code préfixe non uniforme qui permet de minimiser la taille de la représentation de l'image. Pour le construire, on utilise une représentation des codes préfixes sous forme d'arbre binaire étiqueté.

Soit A un arbre binaire. Tout nœud dans l'arbre peut être uniquement identifié par un mot de $\{0,1\}^\star$: le mot décrit le chemin (unique) qui part de la racine et aboutit au nœud en question, avec la convention que 0 signifie « descendre dans le sous-arbre gauche » et 1 signifie « descendre dans le sous-arbre droit ».

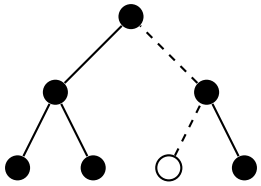

FIGURE 1 – Le nœud *vide* est identifié par le mot 10 puisqu'on l'atteint depuis la racine en descendant d'abord à droite puis à gauche.

Pour représenter un code préfixe c par un arbre, on construit le plus petit arbre tel que pour tout mot de code, il existe un nœud auquel est associé ce mot. Si $c(i) = w$, alors on ajoute l'étiquette i au nœud associé à w. Il y a donc (au plus) k nœuds étiquetés dans l'arbre.

Question 1.6.

(a) Dessiner l'arbre correspondant au codage préfixe suivant : $c(0) = 1$, $c(1) = 001$, $c(2) = 01$, $c(3) = 000$.

(b) Montrer que l'ensemble des mots associés aux feuilles d'un arbre est un ensemble préfixe, c'est-à-dire que pour deux mots u et v dans l'ensemble, $u \not\preceq v$ et $v \not\preceq u$.

(c) Montrer que dans l'arbre qui représente un code préfixe, toutes les feuilles sont étiquetées.

(d) Montrer que dans l'arbre qui représente un code préfixe, seules les feuilles sont étiquetées.

Pour manipuler des arbres, on dispose d'une classe `Arbre` qui s'utilise de la manière suivante.

— si i est un entier, `Arbre(i)` crée un arbre constitué d'un unique sommet, étiqueté par l'entier i ;

— si `G` et `D` sont deux `Arbres`, `Arbre(G, D)` crée l'arbre dont le fils gauche est `G` et le fils droit est `D` ;

— si `A` est un arbre,

  — `A.sag` est le sous-arbre gauche de la racine de `A` (`None` en l'absence de fils gauche),

  — `A.sad` est son sous-arbre droit (`None` en l'absence de fils droit),

  — `A.etiquette` est l'étiquette de la racine de `A`.

Une implantation possible et un exemple d'utilisation est donné à la figure 2.

Question 1.7. Compléter l'algorithme `Arbre_vers_code` qui prend en entrée un objet de type `Arbre` et renvoie le code préfixe correspondant, sous la forme d'un dictionnaire.

```
def Arbre_vers_code(A):
    if A.sag is None and A.sad is None:
        return { A.etiquette: ... }
    dico_g, dico_d = {}, {}      # Dictionnaires vides
    if A.sag is not None:
        ...
    if A.sad is not None:
        ...
    return ...
```

```python
class Arbre:
    def __init__(self, arg1 = 0, arg2 = None):
        if isinstance(arg1, int) and arg2 is None:
            self.sag = self.sad = None
            self.etiquette = arg1
        elif isinstance(arg1, Arbre) and isinstance(arg2, Arbre):
            self.sag = arg1
            self.sad = arg2
            self.etiquette = None
        else:
            raise ValueError("Les paramètres du constructeur n'ont pas le bon type.")

# Arbre du code de la question 1.6
A = Arbre(
      Arbre(
        Arbre(
            Arbre(3),
            Arbre(1)),
        Arbre(2)),
      Arbre(0)
    )
```

FIGURE 2 – Une implantation possible de la classe `Arbre`.

Question 1.8. Soit M une matrice $(n \times m)$ représentant une image avec k niveaux de gris et $c : \{0, \dots, k-1\} \to \{0,1\}^\star$ un code quelconque. Pour tout $i \in \{0, \dots, k-1\}$, on note ℓ_i la longueur du mot $c(i)$ et p_i la fréquence d'apparition de la couleur i dans l'image, c'est-à-dire le nombre de pixels ayant la couleur i dans l'image divisé par le nombre total de pixel.

(a) Calculer les fréquences de la matrice $\begin{pmatrix} 1 & 3 & 2 \\ 2 & 0 & 0 \\ 2 & 1 & 2 \end{pmatrix}$.

(b) Quelle est la valeur de $\sum_{i=0}^{k-1} p_i$ pour une matrice M ?

(c) Exprimer la taille de la représentation d'une matrice M par le code c en fonction de p_i et ℓ_i, $0 \leq i < k$.

(d) Écrire un algorithme qui prend en entrée une matrice M sous forme de liste de listes, et renvoie la liste des fréquences d'apparition de chaque couleur. La liste `L` renvoyée par l'algorithme contiendra la fréquence de la couleur i en case `L[i]`, pour $0 \leq i < k$.

L'algorithme de Huffman permet de construire, à partir de la liste des fréquences, un code optimal pour représenter M. Pour cela, on construit un arbre de la manière itérative suivante :

— on crée pour chaque valeur $i \in \{0, \dots, k-1\}$ un arbre contenant comme unique sommet la feuille d'étiquette i, et on associe la fréquence p_i à l'arbre ;

— tant qu'il reste plusieurs arbres, on considère les deux arbres A_1 et A_2 ayant les plus faibles fréquences associées et on les fusionne en créant un nouvel arbre dont les sous-arbres sont A_1 et A_2. La fréquence associée au nouvel arbre est la somme des fréquences associées à A_1 et A_2.

Le code de Huffman est alors le code préfixe correspondant à l'arbre ainsi construit.

Question 1.9.

(a) Pour la matrice $\begin{pmatrix} 1 & 3 & 2 \\ 2 & 0 & 0 \\ 2 & 1 & 2 \end{pmatrix}$ de la question précédente, dessiner l'arbre de Huffman associé à ses fréquences, puis l'encodage de M à l'aide du code de Huffman associé.

(b) Écrire un algorithme `Arbre_Huffman` qui prend en entrée une liste de fréquences $[p_0, \dots, p_{k-1}]$ et renvoie l'arbre de Huffman associé aux fréquences.

(c) Écrire un algorithme `code_Huffmann` qui prend eni entrée une liste de fréquences et renvoie le code de Huffman correspondant.

(d) Écrire un algorithme `encode_Huffman` qui prend en entrée une matrice et renvoie la matrice codée avec le code de Huffman associé aux fréquences des couleurs de la matrice, ainsi que le code lui-même (pour pouvoir décoder !).

(e) Quelle est la complexité de l'algorithme de Huffman de construction de l'arbre ?

1.4 Codage des différences

Dans une image *typique* (c'est-à-dire non aléatoire), de nombreuses régularités apparaissent. L'exploitation de ces régularités est à la base des techniques de compression d'images. Nous allons voir une façon très basique d'effectuer une telle compression, sans perte de qualité. L'idée est d'exploiter le fait qu'il y a souvent dans les images des grandes zones relativement uniformes. Plutôt que de coder la couleur de chaque pixel, on peut alors ne coder que la différence entre la couleur du pixel courant et la couleur du pixel précédent. Si on garde la couleur du premier pixel, on pourra inverser la transformation.

Plus précisément, étant donnée une matrice M représentant une image en niveaux de gris, on transforme M en liste L de taille N. Si on note $v_0, v_1, \ldots, v_{N-1}$ les couleurs des pixels de M, on définit la suite des différences par $d_i = v_i - v_{i-1}$ pour $i = 1$ à $N - 1$.

Question 1.10.

(a) Quelles valeurs peuvent prendre les entiers d_i si M possède k niveaux de gris ?

(b) Comment retrouver les valeurs des v_i à partir de v_0 et des d_i ?

(c) Écrire un algorithme qui étant donné M, renvoie v_0 et la suite des différences.

(d) Écrire un algorithme qui étant donné la suite des différences et v_0, renvoie la matrice M de départ.

Question 1.11. Écrire un algorithme qui étant donné M, calcule le codage de Huffman associé à la suite des différences de M. *Attention : le codage devra comporter la valeur de v_0 pour pouvoir reconstruire la matrice, et il faudra prendre en considération le fait que les différences ne sont pas toutes positives.*

1.5 Codage optimal et entropie

Soit $p_0, \ldots, p_{k-1}$ les fréquences d'apparition des couleurs dans une matrice M. On définit l'*entropie* H_M de la matrice M par la formule

$$H_M = -\sum_{i=0}^{k-1} p_i \log p_i$$

où le logarithme est le logarithme en base 2, en utilisant la convention que $p_i \log p_i = 0$ si $p_i = 0$.

Question 1.12.

(a) Montrer que pour toute matrice M, $H_M \geq 0$. Dans quel(s) cas a-t-on égalité $H_M = 0$?

(b) Montrer que pour toute matrice M, $H_M \leq \log k$. Dans quel(s) cas a-t-on égalité $H_M = \log k$? *Indications : on pourra remarquer que* $\log k = \sum_i p_i \log k$ *et utiliser (sans la démontrer) l'inégalité* $\log z \leq z - 1$, *valable pour tout* $z > 0$.

On définit le *nombre moyen de bits par symbole* d'une représentation d'une matrice M comme étant la taille de la représentation de M divisée par le nombre d'éléments dans la matrice M (qui vaut mn pour une matrice de dimensions $m \times n$). On cherche à montrer que l'entropie d'une matrice M donne une borne inférieure sur le nombre moyen de bits par symbole nécessaire pour la représenter.

Question 1.13. On considère à nouveau la matrice $\begin{pmatrix} 1 & 3 & 2 \\ 2 & 0 & 0 \\ 2 & 1 & 2 \end{pmatrix}$.

(a) Quelle est son entropie ?

(b) Quel est le nombre moyen de bits par symbole dans la représentation de cette matrice par le code de Huffman ?

Question 1.14. Considérons un code c uniquement décodable, et notons $\ell_0, \ldots, \ell_{k-1}$ les longueurs respectives de $c(0), \ldots, c(k-1)$. On suppose sans perte de généralité que $\ell_0 \leq \ell_1 \leq \cdots \leq \ell_{k-1}$. On pose $K = \sum_{i=0}^{k-1} 2^{-\ell_i}$. On va montrer que $K \leq 1$.

Pour tout couple d'entiers (u, ℓ), on note $C_{u,\ell}$ le nombre de suites $[x_1, \ldots, x_u] \in X^\star$ de taille u telles que la longueur du mot $c^\star([x_1, \ldots, x_u])$ est égale à ℓ.

(a) Montrer que $C_{u,\ell} = 0$ pour $\ell < u\ell_0$ et $\ell > u\ell_{k-1}$.

(b) Montrer que pour $u\ell_0 \leq \ell \leq u\ell_{k-1}$, $C_{u,\ell} \leq 2^\ell$.

(c) Montrer que pour tout u, ℓ,

$$C_{u+1,\ell} = \sum_{m=\ell_0}^{\ell_{k-1}} C_{u,\ell-m} C_{1,m}.$$

(d) En déduire que pour tout u,

$$K^u = \sum_{\ell=u\ell_0}^{u\ell_{k-1}} C_{u,\ell} 2^{-\ell}.$$

(e) En déduire que pour tout u, $K \leq (u(\ell_{k-1} - \ell_0) + 1)^{1/u}$.

(f) En considérant la limite de l'expression précédente pour $u \to +\infty$, montrer que $K \leq 1$.

Question 1.15.

(a) En remarquant que $\sum_i p_i \ell_i = \sum_i p_i \log(2_i^\ell)$, montrer que

$$\sum_{i=0}^{k-1} p_i \log(1/p_i) - \sum_{i=0}^{k-1} p_i \ell_i \leq 0.$$

On pourra utiliser l'inégalité $\log z \leq z - 1$ *pour tout* z.

(b) En déduire que pour représenter M par un code uniquement décodable, le nombre moyen de bits par symbole doit être supérieur ou égal à H_M.