STRUCTURE DE DONNÉES

Codage Huffman

Introduction

Le codage d'un texte se fait aujourd'hui majoritairement à travers la **norme UTF8**, qui inclut en particulier le **code ASCII** pour lequel les caractères les plus courants sont codés sur **8 bits**. Le **codage Huffman** (publié en 1952 par David Albert Huffman) est une méthode de compression de données qui s'appuie sur les fréquences d'utilisation de chaque caractère : plus un caractère est utilisé, et plus le code le représentant est court.

1 Principe du codage

1.1 Premier essai

Pour illuster le principe du codage Huffman, on cherchera à coder un message simple : « abracadabra ».

1. Compléter le tableau suivant en indiquant le nombre d'occurences de chaque caractère dans ce message.

Caractère					
Nombre d'occurences					

On propose une série de codes les plus simples possibles (0 à 4 en binaire).

2. Associer chaque caractère dans le tableau suivant à un code en respectant le principe du codage de Huffman : plus les caractères sont utilisés et plus leur code doit être court.

Code	0	1	10	11	100
Caractère					

Une fois la **table de codage**[1] construite :

- on réalise le codage du message en mettant bout à bout les codes binaires des différents caractères constituant le message d'origine, sans espace entre les bits ;
- on réalise le décodage en découpant le message codé en une suite de mots binaires les plus petits possibles présents dans la table et en les remplacant par leurs valeurs.

3. Coder le message en utilisant la table de codage définie à la question précédente.
4. Décoder le message à l'aide de la table de codage. Le message décodé correspond-il au message d'origine ? Pourquoi ?

1. Tableau reliant un code à sa valeur.

1.2 Algorithme de Huffman

Il est nécessaire de créer un **code-préfixe**, c'est-à-dire un codage dans lequel aucun code n'est le début d'un autre code.

L'algorithme de Huffman permet de générer un code-préfixe optimal[2] de taille variable[3] en construisant un **arbre binaire**, une **structure de données** hiérarchisée autour de **nœuds pondérés**[4]. A la base de l'arbre, on trouve la **racine**, reliée à un maximum de deux nœuds **fils** (appelés **sous-arbre gauche** et **sous-arbre droit**), eux-même **pères** d'un maximum de deux autres nœuds. A l'extrémité des branches de l'arbre, on trouve les **feuilles**. La **hauteur** de l'arbre correspond à distance entre la racine et la feuille la plus éloignée.

Algorithme d'Huffman :

```
pour (chaque caractère dans le message) :
    creer une feuille dont le poids = nombre d'occurences
tant que (tous les noeuds ne sont pas sous une meme racine) :
    classer les feuilles et les racines par poids croissants
    creer une racine à partir des 2 feuilles/racines de poids faibles
```

1.3 Exemple

On souhaite coder pour cet exemple uniquement le message suivant : « copie-colle ».

Etape 1. On détermine le nombre d'occurences de chaque caractère et on crée des feuilles dont le poids correspond au nombre d'occurences.

FIG. 1 – *Etape 1*

Etape 2. On classe les feuilles par ordre croissant de leur poids.

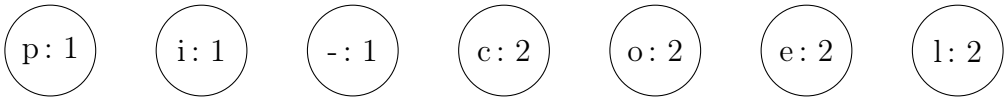

FIG. 2 – *Etape 2*

Etape 3. A partir des deux feuilles de poids le plus faible, on crée une racine dont le poids est la somme des poids des feuilles.

2. Permettant d'obtenir une taille en bits la plus faible.
3. Dans lequel les différents caractères sont codés sur un nombre de bits différents.
4. Qui possèdent des poids différents.

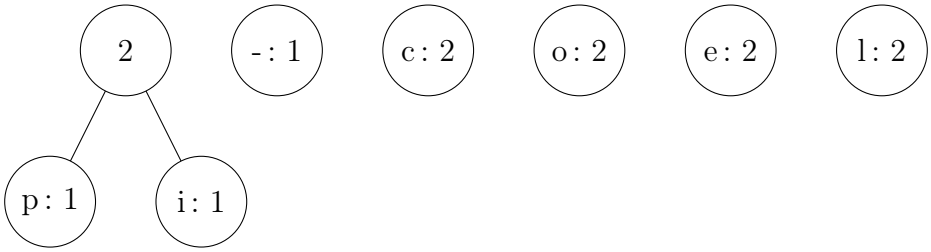

FIG. 3 – *Etape 3*

Etape 4. On classe les feuilles restantes et le(s) racine(s) par ordre croissant.

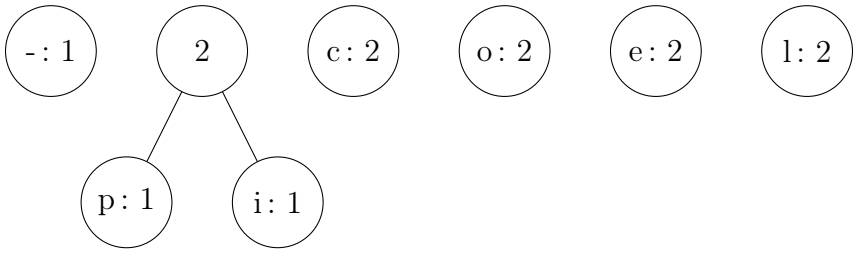

FIG. 4 – *Etape 4*

Etape 5. On crée une nouvelle racine à partir des deux feuilles (ou racines) de poids le plus faible.

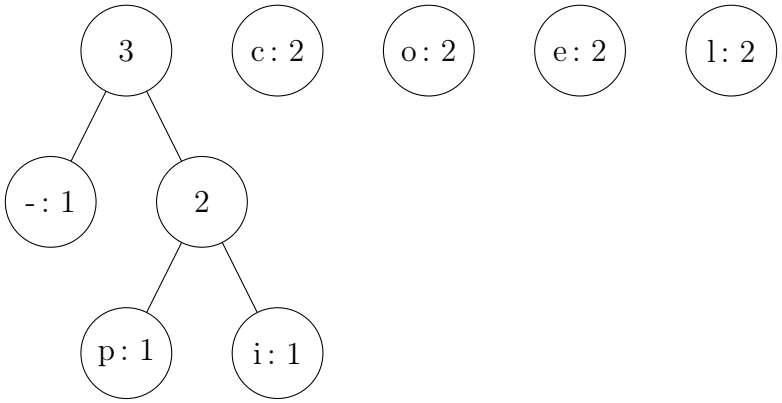

FIG. 5 – *Etape 5*

Etape 6. On réitère les étapes 4 et 5 jusqu'à ce que l'ensemble soit sous une unique racine.

Etape 7. Pour obtenir la table de codage, on part de la racine et pour chaque nœud: le chemin de gauche vaut 0 et le chemin de droite vaut 1.

Caractère	c	o	-	p	i	e	l
Code	00	01	100	1010	1011	110	111

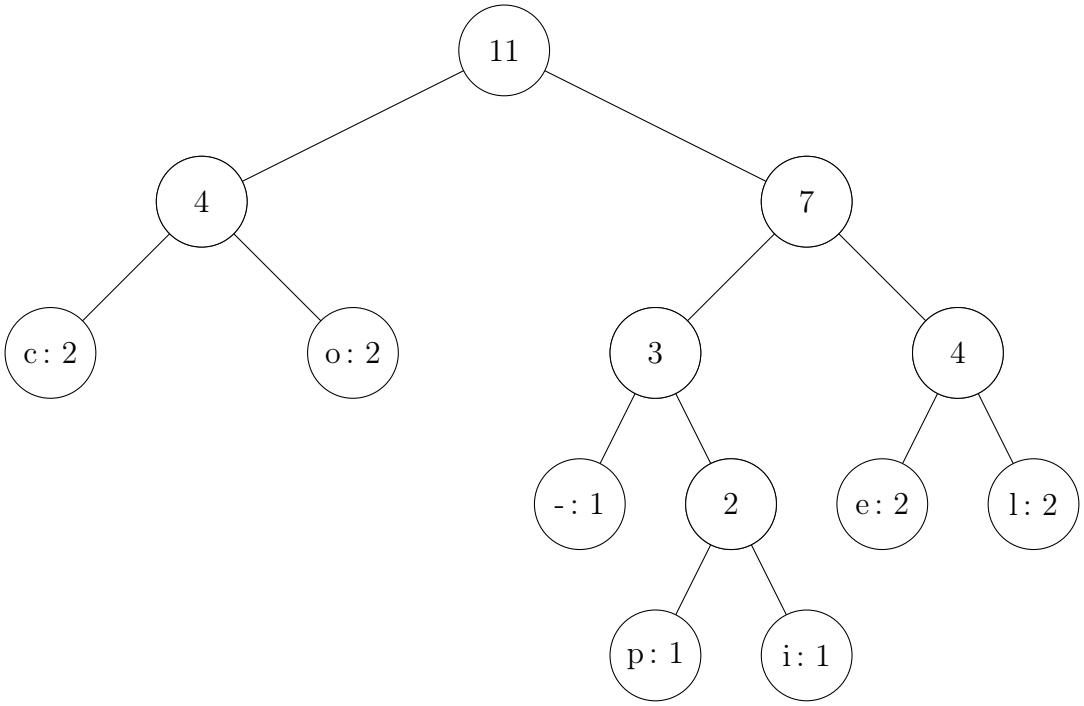

FIG. 6 – *Etape 6*

Etape 8. On remplace chaque caractère par son code dans la table de codage.

$$copie - colle = \underbrace{00}_{c}\underbrace{01}_{o}\underbrace{1010}_{p}\underbrace{1011}_{i}\underbrace{110}_{e}\underbrace{100}_{-}\underbrace{00}_{c}\underbrace{01}_{o}\underbrace{111}_{l}\underbrace{111}_{l}\underbrace{110}_{e} \quad (1)$$

$$= 00011010101111010000011111111110 \quad (2)$$

1.4 Application

1. Etablir l'arbre binaire du message « abracadabra »en utilisant l'algorithme de Huffman.
2. En déduire la table de codage correspondante.

Code					
Lettre					

3. S'agit-il d'un code-préfixe ?
4. Coder le message en utilisant la table.
5. Décoder le code. S'agit-il du message d'origine ?

1.5 Compression

On peut définir le taux de compression comme le rapport entre le poids gagné grâce à la compression et le poids initial du fichier :

$$t = \frac{poids_{initial} - poids_{compresse}}{poids_{initial}}$$

Le taux de compression peut être exprimé en pourcentage : plus il est grand, et plus la compression a été efficace.

1. Calculer le poids du message codé en ASCII.
2. Calculer le poids du message codé à l'aide de l'algorithme de Huffman.
3. En déduire le taux de compression.

2 Généralisation à un cas plus complexe

On dispose d'un long texte (variable *texte* dans le fichier *table_de_codage.py*) qu'il serait très fastidieux de coder « à la main ». On préfère donc utiliser l'outil informatique (Python) afin d'automatiser la tâche : l'ensemble des fonctions à coder devra être enregistré dans un fichier nommé *huffman.py*.

1. Créer une fonction *frequenceCaracteres(texte)* prenant comme argument une chaîne de caractères, et renvoyant un **dictionnaire** dont les clés seront les différents caractères du texte et les valeurs la fréquence d'apparition (en pourcentage) de ces caractères. On définit la **fréquence** de la façon suivante :

$$f_{caractere} = \frac{\text{nombre d'occurences d'un caractère dans le texte}}{\text{nombre total de caractères dans le texte}}$$

2. Importer le module *table_de_codage* et utiliser la fonction *frequenceCaracteres* avec le texte contenu dans ce module.

On fournit la table de codage obtenue par l'algorithme de Huffman des caractères (minuscules, sans ponctuation) les plus utilisés dans la langue française [5].

Caractère	Code	Caractère	Code
'a'	1110	'o'	0010
'b'	000010	'p'	00110
'c'	10010	'q'	1001100
'd'	11110	'r'	0101
'e'	110	's'	0111
'f'	1001111	't'	0100
'g'	000011	'u'	11111
'h'	000000	'v'	000001
'i'	1000	'w'	1001110111
'j'	10011010	'x'	10011011
'k'	100111010	'y'	10011100
'l'	0001	'z'	1001110110
'm'	00111	' '	101
'n'	0110		

FIG. 7 – *Table de codage des caractères les plus courants dans la langue française.*

5. La table est incluse dans le module *table de codage*.

3. La table de codage fournie est-elle cohérente par rapport aux fréquences obtenue à la question précédente ?
4. Créer une fonction *codage(texte)* prennant comme argument une chaîne de caractères, et renvoyant une chaîne de caractères contenant le message codé. Vérifier que les premiers caractères sont codés correctement.
5. Créer une fonction *decodage(texte)* prennant comme argument une chaîne de caractères, et renvoyant une chaîne de caractères contenant le message décodé. Vérifier que le décodage du message codé renvoie le message d'origine.

Dans le codage Huffman, plusieurs variantes sont possibles :

- Les caractères sont codés à partir d'une table de codage indépendante du message[6] et enregistrée dans le logiciel de codage/décodage (à l'image de la table fournie dans *table_de_codage.py*) : c'est le codage **statique**.
- Les caractères sont codés à partir d'une table issue d'un arbre binaire construit à l'aide de l'algorithme d'Huffman : c'est le codage **semi-adaptatif**. Dans ce cas, l'arbre doit être envoyé avec le message codé pour pouvoir être décodé.

6. Calculer la taille du message « abracadabra »en utilisant la table de codage fournie dans *table_de_codage.py*. En déduire le taux de compression $t_{statique}$. Comparer avec le taux de compression $t_{semi-adaptatif}$ calculé à la question 1.5.3. Conclure.

6. Mais liée à la langue.