TP n°2 : Codage Huffman

Auteur : Rémi Cozot
Version : 2019.0 – Date : Août 2019

Introduction

Le codage de Huffman (1952) est une méthode de compression de données qui permet de réduire la taille du codage d'un alphabet. Il substitue à un code de longueur fixe (le code ASCII, par exemple, où chaque caractère est codé sur 8 bits) un code de longueur variable. Ce principe de compression est utilisé dans de nombreuses applications. Le principe général du codage de Huffman est d'associer à chaque symbole du texte à encoder un code binaire qui est d'autant plus court que le caractère correspondant a un nombre d'occurrences élevé dans le texte à encoder.

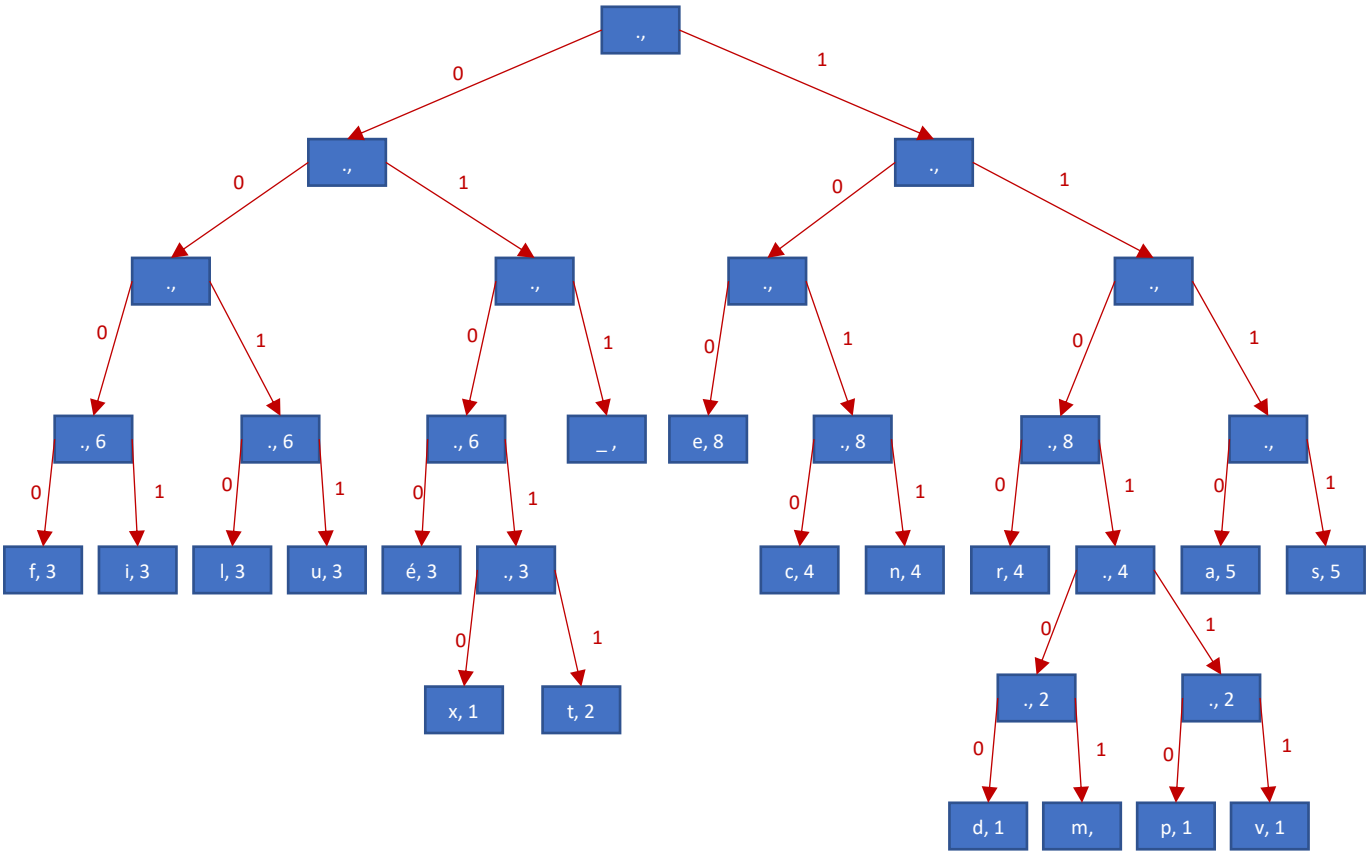

Figure 1 : arbre d'Huffman de codage et de décodage

Par exemple, dans le texte "***les crépuscules dans cet enfer africain se révélaient fameux***" (extrait du « Voyage au bout de la nuit» de Céline), le caractère ' ' (espace) qui apparaît 8 fois pourra être codé avec 3 bits "011", le caractère 'e' qui apparaît 8 fois pourra être codé avec 3 bits "100", le caractère 's' qui apparaît 5 fois pourra être codé avec 4 bits '1111' ; le caractère 'p' qui apparaît 1 fois pourra être codé avec 6 bits "110110", etc. Voici le même texte codé par le codage de Huffman :

```
00101001111011101011000100110110001111111010001100101001111011110100111010
11111011101010001011011100101100001001100011111000001100000110101110000110
10111111100011110001001101110100001011100001100101101011011000011101101011
00001101010
```

La longueur du texte ainsi codé est de 235 bits, qui peuvent se représenter avec 30 octets alors que le texte non codé occupe 60 octets. Les codes étant de longueur variable, il est nécessaire, pour pouvoir décoder un texte, que le code de chaque caractère ne soit préfixe du code d'aucun autre caractère. Pour obtenir cette propriété, on représente un code de Huffman par un arbre binaire dont les feuilles sont étiquetées par des couples (caractère, fréquence). La ***Figure 1*** donne l'arbre de Huffman qui permet de coder puis de décoder les caractères du texte ci-dessus.

Le code d'un caractère est donné par le chemin suivi depuis la racine jusqu'à la feuille où il est situé, en associant un "0" au fils gauche et un "1" au fils droit ; on voit donc que le code du caractère 'f' est "0000", celui du caractère 'x' est "01010" et celui du 'v' est "110111".

Étapes du codage et du décodage

Cette partie indique les grandes étapes du codage et du décodage ; la réalisation concrète est décrite plus loin.

Codage

1. **Lire** le texte contenu dans le fichier à coder ;
2. **Calculer** la fréquence d'apparition de chaque caractère et construire une table de fréquences ;
3. **Enregistrer** cette table dans le fichier de sortie ;
4. **Construire** l'arbre de Huffman puis l'afficher pour vérification ;
5. **Construire** la table de codage associée puis l'afficher pour vérification ;
6. **Coder** le texte avec la table de codage ;
7. ***Enregistrer*** le texte codé dans le fichier de sortie.

Décodage

1. **Lire** la table de fréquences dans le fichier à décoder ;
2. **Construire** l'arbre de Huffman puis l'afficher pour vérification;
3. **Lire** le texte codé ;
4. **Décoder** avec l'arbre de Huffman ;
5. ***Enregistrer*** le texte décodé dans le fichier de sortie.

Codage : construction de l'arbre de Huffman

1. À partir d'un texte codé comme une chaîne de caractère, écrivez une fonction qui calcule la table de fréquences (c'est-à-dire le nombre d'occurrence de chaque caractère). **Note** : la fonction python **ord()** renvoie le code ascii entier d'un caractère, par exemple :
   ord('c') -> 99
2. A partir du tableau de fréquences et en ne retenant que les caractères réellement présents (nombre d'occurrence >0) dans le texte, construire **une liste dont les éléments sont des arbres binaires de couples (caractère, fréquence)**. **Cette liste doit être triée par fréquence croissante**.
3. Construire l'arbre de Huffman, de la manière suivante :
   a. **prendre les deux éléments** (arbres) minimaux de la liste ci-dessus,

b. en faire un nouvel arbre dont la racine est un couple dont le **caractère est quelconque** (sans signification, dans l'exemple le caractère retenu est '.' car il n'est pas présent dans le texte) et **la fréquence est la somme des fréquences** des deux éléments retirés de la liste (Cf. ***Figure 2***).

4. **Insérer l'arbre ainsi obtenu à sa place dans la liste triée ;**

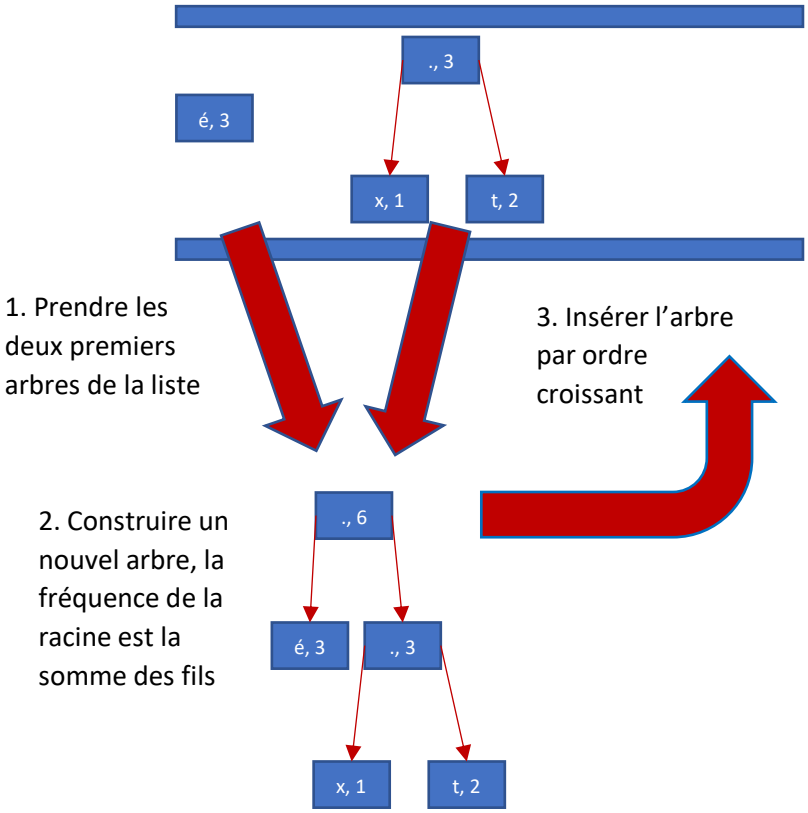

Figure 2 : principe de construction de l'arbre d'Huffman

5. Répéter jusqu'à ce que la liste ne contienne plus qu'un seul élément **: l'arbre obtenu est l'arbre de (dé)codage associé au texte**.
6. Écrire une fonction **print_Huffman** qui affiche au terminal la valeur de chaque feuille de l'arbre de Huffman (caractère et fréquence), ainsi que le code correspondant. Exemple de résultat :

(f,3) : 0000
(i,3) : 0001
(l,3) : 0010
(u,3) : 0011
(é,3) : 0100
(x,1) : 01010
(t,2) : 01011
(,8) : 011
(e,8) : 100
(c,4) : 1010
(n,4) : 1011
(r,4) : 1100

(d,1) : 110100
(m,1) : 110101
(p,1) : 110110
(v,1) : 110111
(a,5) : 1110
(s,5) : 1111

A l'issue du développement de la partie codage vous devriez avoir les fonctions suivantes :

- **build_Frequency_table** (chaîne_de_caractères) → table_des_frequences
- **build_Huffman_Tree_List**(table_des_frequences) → liste d'arbres feuilles d'Huffman
- **insert_Huffman_tree_in_list**(liste_arbres, arbre) → liste avec l'arbre insérer par ordre croissant de fréquences
- **build_Huffman_Tree**(table_des_frequences) → arbre_huffman
- **get_Dict_Huffman**(arbre_huffman, dict_caractere_code) → table de hachage permettant d'avoir en un appel le code correspondant à un caractère.
- **code_text**(texte_brut, dict_caractere_code) → texte codé
- **print_Huffman**(arbre_huffman) → affiche l'arbre

Décodage

1. **Écrire une fonction** qui, étant donné un texte codé et l'arbre de Huffman associé (reconstruit à partir de la table des fréquences) renvoie le texte décodé dans une chaîne de caractères.

Aides

Créer une liste contenant n fois la valeur v

```
list = [v]*n
tableFrequence = [0]*255
```

Caractère vers entier et inverse

```
ord('c') -> 99
chr(99)  -> 'c'
```

Insertion dans une liste python

```
myList.insert(i,data)
```

Lecture/écriture dans un fichier avec json

```
import json

x = [freqTable, texteEncode]

# saving file
f = open('texteEncode.txt', 'w')
json.dump(x, f)
f.close()

# read file
f = open('texteEncode.txt', 'r')
x = json.load(f)
f.close()

freqTable = x[0]
texteEncode =x[1]
```