TP 3 de Compression des Données

Réalisé par : MULAPI TITA Ketsia

Introduction et notion de base sur la méthode de Huffman

Lors du TP précédent, nous avons vu la méthode classique de Shannon et, la méthode BPE. Dans ce deuxième TP, nous allons à la découverte d'une autre méthode de compression, il s'agit de celle de Huffman.

L'objectif étant de compresser nos texte « OliverTwist », nous allons d'abord essayer de comprendre ce que nous voulons obtenir avec un court exemple.

Pour comprendre, de manière effective ce que nous allons faire dans ce TP, reprenons l'exemple vu lors de la séance précédente de TD, soit les fréquences d'apparitions suivantes, on va supposer que nous avons les caractères ci-dessous qui représente un texte quelconque et on associe à chaque caractère une fréquence d'apparition.

- Caractères : « ABCDEF »
- Fréquences :

Caractères	Fréquences
A	0.07
B	0.12
D	0.17
C	0.23
E	0.31
F	0.10

Pour déterminer le code d'une paire de caractères et, la taille du message voici comment procéder à l'aide de la méthode de Huffman :

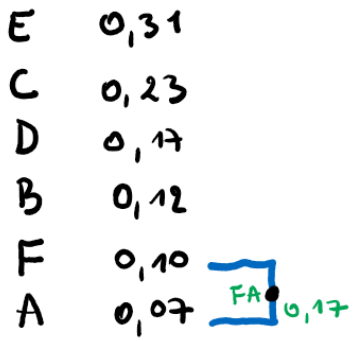
1- Trier dans l'ordre décroissant des fréquences, chaque caractère.

Caractères	Fréquences
E	0.31
C	0.23
D	0.17
B	0.12
F	0.10
A	0.07

(voir statistiques textuelles textuelle).

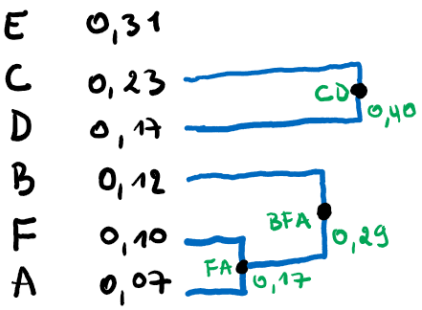
2- Regrouper les 2 moins fréquents et les coder par un même symbole, répéter jusqu'à n'avoir qu'un symbole (la racine) : c'est l'étape de construction de l'arbre de Huffman

Itenation 1

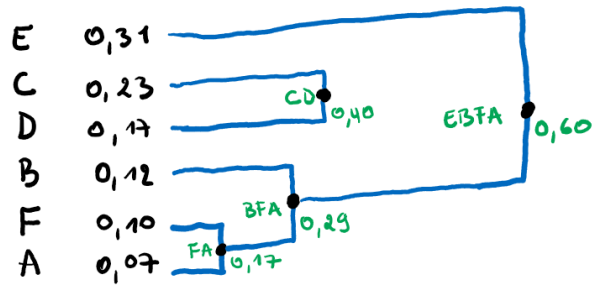


Itenation 2

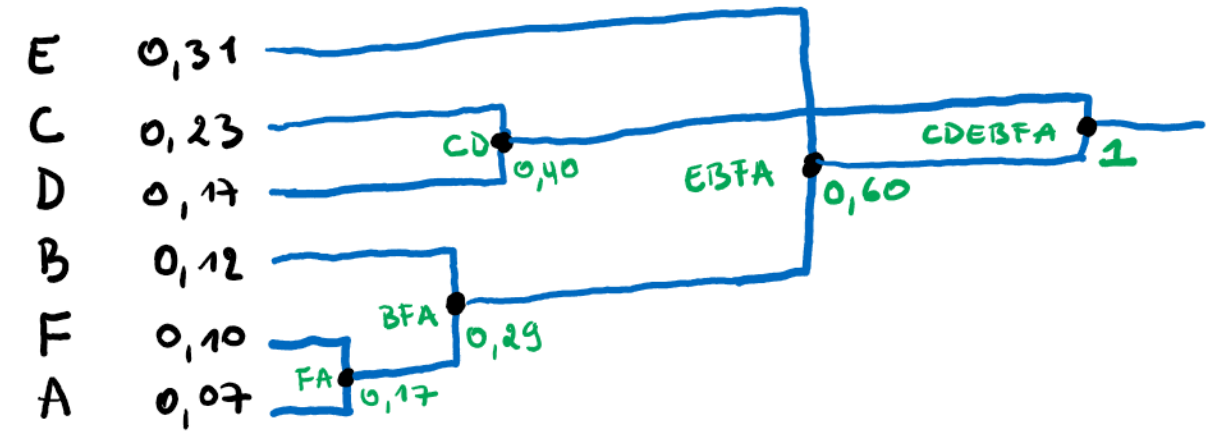
Itenation 3



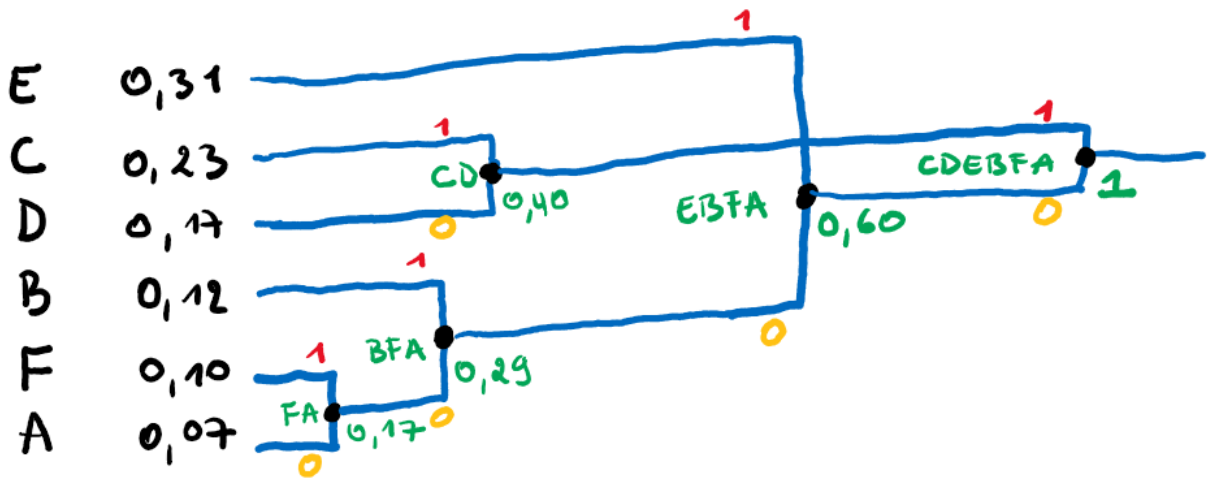
Itenation 4



Itenation 5



3- Parcourir l'arbre jusqu' aux feuilles en attribuant 1 à la branche la plus haute et 0 à la branche la plus basse



Nous avons donc pour chaque caractère le code suivant :

E = 01
C = 11
D = 10
B = 001
F = 0001
A = 0000

Enfin nous concluons avec quelques caractéristiques tel que l'entropie, et la longueur moyenne qui est une propriété de l'arbre de Huffman :

- Soit « Li », la longueur d'un caractère en partant de la racine à la feuille (caractère) et, « Pi », la fréquence relative à ce caractère, on note :

$$\bar{l} = \sum_i p_i l_i$$

La longueur moyenne qui n'est rien d 'autre que la sommes des poids des nœuds.
Ici la longueur moyenne :

L = 2 * (0.31 + 0.23 + 0.17) + 3 * 0.12 + 4 * (0.1 + 0.07)
= 1.42 + 0.36 + 0.68
= 2.46 bits

- Enfin comme à nos habitudes, pour calculer l'entropie il suffit de faire

$$H = -\sum_{i=1}^{6} P_i \log_2 P_i$$

H = 2.2174 bits ou Shannon

Enfin nous partirons de ces deux caractéristiques afin de tirer une conclusion sur la relation entre la distribution des Probabilités et, la borne entropique.

Etant donnée la relation suivante :

$$E \leq \bar{\ell} < E+1$$

Nous avons E = 2.2477 bits, L = 2.46 bits et E + 1 = 3.2477 bits

Il est donc évident que nos distributions ne sont pas très éloignées des puissances de 2 car on est proche de la borne entropique :

$$2,247 \leq 2,46 < 3,24$$

$$E \leq \bar{\ell} < E+1$$

Implémentation de l’algorithme de Huffman

A. Implémentation

Nous allons à présent répéter le même procédé, mais cette fois ci, en utilisant des DataFrame python à la place des listes tel que recommandés dans l’énoncé, mais tel que permis par le responsable du cours.

Dans le respect des étapes ci-dessous, nous avons abouti à des résultats satisfaisants, notons aussi que durant cette expérience nous avons principalement utiliser le fichier « OlicerTwist2.txt » avant de faire les tests avec les 3 premiers chapitres :

```
1- Calcul des statistiques textuelles
2- Initialisation de la liste des feuilles de l’arbre binaire
3- Initialisation de la liste ordonnée des nœuds à traiter
4- Tant que la liste des nœuds n’est pas vide
      a. Créer un nœud parent avec les deux premiers nœuds de la liste
      b. L’ajouter en fin de liste des nœuds de l’arbre binaire
      c. L’ajouter à la liste des nœuds
      d. Retirer les deux premiers nœuds de la liste
      e. Ordonner la liste dans l’ordre des poids croissants
5- Parcourir toutes les branches de l’arbre depuis la racine jusqu’aux
   feuilles et construire le code binaire
```

Dans ce texte, on retrouve :

alphabet :
[' ' '!' '"' "'" ',' '-' '.' ':' ';' 'A' 'C' 'D' 'E' 'H' 'I' 'L' 'M' 'O' 'P' 'Q' 'R' 'S' 'T' 'U' 'a' 'b' 'c' 'd' 'e' 'f' 'g' 'h' 'i' 'j' 'l' 'm' 'n' 'o' 'p' 'q' 'r' 's' 't' 'u' 'v' 'w' 'x' 'y' 'z' 'à' 'â' 'ç' 'è' 'é' 'ê' 'ï' 'î' 'ô' 'ù' 'û']

Taill de l’alphabet : 59

Distribution des caracteres :
[479 1 32 36 1 9 4 2 2 1 1 3 1 2 3 1 8 3 2 3 1 3 1 157 19 70 90 348 26 20 10 172 10 114 59 166 107 83 35 141 143 170 159 37 2 10 5 1 9 2 2 11 38 3 1 1 4 2 3]

Longueur du texte : 2829

- On a alors une suite de 60 caractères classé selon l'ordre décroissant des fréquences

	Symbole	Occurence
0		479
27	e	348
31	i	172
41	t	170
35	n	166
42	u	159
23	a	157
40	s	143
39	r	141
33	l	114
36	o	107
26	d	90
37	p	83
25	c	70
34	m	59
52	é	38
43	v	37
3	,	36
38	q	35
2	'	32
28	f	26
29	g	20

(…)

44	w	2
50	ç	2
49	â	2
57	ù	2
8	A	2
7	;	2
13	I	2
18	Q	2
9	C	1
47	z	1
10	D	1
12	H	1
15	M	1
4	-	1
1	!	1
54	î	1
55	ï	1
20	S	1
22	U	1

```
type(df_huff.Code)
```

On peut par exemple constater que l'espace est présent 479 fois dans le texte, comme dans le TP précédent, nous savons comment calculer la probabilité d'apparition de ce caractère.

- Nous avons ensuite après avoir appliqué l'étape itérative et, le codage, obtenu les paires suivantes :

	Symbole	Code
0	ï	"1111111001111"
1	S	"1111111001110"
2	-	"1111111001101"
3	î	"1111111001100"
4	z	"1111111001011"
5	U	"1111111001010"
6	D	"1111111001001"
7	M	"1111111001000"
8	H	"1111111000111"
9	C	"1111111000110"
10	â	"11111100010"
11	ç	"11111100001"
12	;	"11111100000"
13	Q	"11111001111"
14	w	"11111001110"
15	ù	"11111001101"
16	I	"11111001100"
17	A	"11111000011"
18	!	"11111000010"
19	ô	"1111100101"
20	:	"1111100100"
21	û	"1111100000"
22	L	"1011110111"
23	ê	"1011110110"
24	E	"1011110101"
25	R	"1011110100"
26	T	"1011110011"
27	P	"1011110010"
28	à	"111111011"
29	.	"111111010"
30	O	"111110001"
31	y	"101111000"
32	è	"10111011"
33	h	"10111010"
34	x	"10111001"
35	j	"10111000"
36	q	"1111111"
37	'	"1111101"
38	f	"1011111"
39	g	"0101011"
40	b	"0101010"
41	m	"111011"
42		"111010"
43	é	"010100"
44	,	"001010"
45	l	"11110"
46	o	"11100"
47	d	"10110"
48	p	"01011"
49	c	"00100"
50	i	"1010"
51	t	"0111"
52	n	"0110"
53	u	"0100"
53	u	"0100"
54	a	"0011"
55	s	"0001"
56	v	"0010"
57	r	"0000"
58		"110"
59	e	"100"

Remarquons qu'il nous faut « 110 » pour coder un espace et, «111010 » pour coder un double espace (ce qui est tout à fait normal car rappelons-nous que l'on conserve des codes longs pour les caractères moins probables et, des codes plus courts pour les caractères les plus probables, ce principe est le même lorsqu'il s'agit des mots, etc.).

B. Caractéristiques

Enfin, pour chaque fichier, nous avons dans les mêmes conditions, déterminées les informations suivantes :

Fichier	Taille initiale (Bits)	Taille après compression (Bits)	Entropie (Bits/Shannon)
OlivierTwist2	22632	12537	4.38
OlivierTwistChap1	49552	27628	4.32
OlivierTwistChap2	80000	44910	4.36
OlivierTwistChap3	80000	44941	4.40

Tout comme dans le TP précédent, on peut constater que le contenu des chapitres 2 et 3 sont presque similaires, en termes de compression, nous voyons que Huffman nous rapproche de 50% de compression.

Conclusion :

L'objectif de ce TP était d'implémenter la méthode de Huffman afin de construire l'arbre binaire tel que vu plus haut. Objectif atteint, mais la seule contrainte ici est de constater que le temps ne nous a pas permit d'aller jusqu'au bout afin de déterminer par exemple la longueur moyenne.

Cependant, nous restons convaincus que l'avantage de la méthode de Huffman, réside dans son principe, alors qu'avec Shannon Fanon, nous codions chaque symbole pris individuellement, Huffman nous dit le contraire, en effet, ce dernier est beaucoup plus optimale car comme vu plus haut, nous avons codé des alternatives avec un seul bit étant donnée les symboles (ai) formés à partir d'un regroupement.