Compression de texte

Michel LAFOND, Lycée Le Castel à Dijon

Nous allons voir un algorithme (de David Huffman 1925-1999), qui permet de compresser n'importe quel texte écrit en français d'environ 15% en moyenne, et ceci sans perdre le moindre caractère ! C'est-à-dire qu'après décompression on retrouve le texte de départ sans perte d'information !

- Ceci semble paradoxal, car si à partir d'un texte de longueur N on peut en obtenir un de longueur 0,85 N, alors en itérant le processus de compression on pourrait parvenir à un texte arbitrairement court !

Le paradoxe est résolu en ce sens :
La quasi totalité de la littérature "ordinaire" sera effectivement compressée de 15% en moyenne.
Mais certains textes "inhabituels" ceux qui sont par exemple composés de mots contenant beaucoup de lettres rares comme WHISKY, seront peu compressés ou pire seront allongés par l'algorithme.
Les algorithmes de Huffman sont spécifiques à chaque langue, et celui qu'on va exposer pour le français ordinaire, compressera bien de 15% environ la quasi totalité des textes qu'on lit.
Voici une situation type :

	"TEL|PERE|TEL|FILS"	17 caractères
deviendra	"Z|ZE|PETCUIIGL"	14 caractères.

En musique l'algorithme MP3 (ou ses frères) supprime d'une musique une grande quantité d'informations complètement inutiles parce que l'oreille ne les perçoit pas ou peu. Mais ce qu'on obtient n'est pas la musique intégrale de départ qui est inaccessible à partir du seul fichier compressé.

Pour les images, l'algorithme JPEG (ou ses cousins) supprime également de l'information non pertinente avec évidemment une altération de l'image plus ou moins grande selon le taux de compression choisi.

- Pour un texte, les contraintes sont plus fortes, car il ne faut pas en perdre la moindre miette.

Pensons qu'un seul caractère erroné peut modifier complètement le sens d'un message.

D'abord, fixons l'alphabet utilisé : il est commode (pour le codage binaire) de prendre un alphabet dont le nombre de caractères est une puissance de 2. Habituellement c'est $2^8 = 256$ comme dans le code ASCII. Ce code permet à l'aide de 8 chiffres binaires d'écrire tous les entiers de 00000000 (le zéro décimal) à 11111111 (255 décimal). Ce qui permet de coder l'alphabet usuel des 26 lettres minuscules et majuscules, la ponctuation, les 10 chiffres etc. En général ça ne suffit pas : vérifiez par exemple sur votre ordinateur que dans la police "Times New Roman" il y a plus de 1100 caractères dont au moins 40 sortes de "a" !!! :

à á â ã ä å ā ă ą ậ ắ ẵ ấ ằ ẳ ẫ ặ ạ ả ấ ầ ẩ Ạ Ả Ấ Ầ Ẩ Ẫ Ậ Ắ Ằ Ẳ Ẵ Ặ A Á Â Ã Å Ā Ă Ą

Pour simplifier dans notre exemple, nous prendrons l'alphabet majuscule des 26 lettres non accentuées plus les 6 caractères :

"*BLANC*"	noté	|	servant de séparateur,
"*APOSTROPHE*"	noté	'	
"*POINT*"	noté	.	fin de phrase,
"*VIRGULE*"	noté	,	
"*TIRET*"	noté	-	le tiret peut servir aussi de parenthèse.
"*POINT D'INTERROGATION*"	noté	?	

Cela fait 32 caractères qu'on va coder dans un premier temps par un codage binaire ordinaire à 5 bits puisque $2^5 = 32$. Appelons ce codage BINAIRE-1

caractère	Code BINAIRE-1	caractère	Code BINAIRE-1
A	00000	P	10000
blanc	00001	Q	10001
B	00010	R	10010
C	00011	S	10011
D	00100	T	10100
E	00101	U	10101
F	00110	V	10110
G	00111	W	10111
H	01000	X	11000
I	01001	Y	11001
J	01010	Z	11010
K	01011	*apostrophe*	11011
L	01100	*virgule*	11100
M	01101	*tiret*	11101
N	01110	*point*	11110
O	01111	*interrogation*	11111

- C'est ici qu'intervient l'idée géniale de David Huffman, idée qu'il a eue lorsqu'il était étudiant au Massachussets Institute of Technology.

Le code BINAIRE-1 précédent utilise 5 bits par caractère ce qui est un peu bête quand on sait qu'en français ordinaire, 17 % des lettres sont des "E". Donc ce serait plus intelligent de coder les lettres fréquentes par des codes courts (comme 01 ou 010) et les lettres rares par des codes plus longs (comme 011011101) en jouant sur la fréquence des lettres de la langue qu'on veut coder, et en s'arrangeant pour qu'en moyenne le nombre de bits par caractère soit inférieur à 5.

En français ordinaire les lettres ont les fréquences suivantes (qui varient légèrement selon les textes qui ont été utilisés pour la statistique), classées de la plus fréquente à la moins fréquente :

	fréquence		fréquence		fréquence
E	0,144	D	0,033	*Virgule*	0,007
blanc	0,128	C	0,026	*tiret*	0,007
S	0,062	M	0,024	H	0,007
A	0,062	P	0,021	J	0,007
N	0,062	V	0,013	K	0,007
I	0,058	*apostrophe*	0,013	X	0,006
R	0,057	*point*	0,008	Y	0,006
T	0,057	Q	0,008	*interrogation*	0,006
U	0,049	B	0,007	Z	0,005
O	0,048	F	0,007	W	0,005
L	0,043	G	0,007		

La fréquence du "E" n'est que de 14 % et non de 17 % comme on le lit partout, parce qu'ici on prend en compte la ponctuation et le blanc.

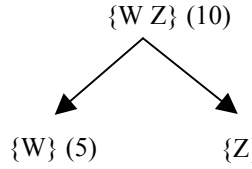
L'affectation des codes courts aux lettres fréquentes est optimisée lorsqu'on utilise l'algorithme de Huffman qui fonctionne ainsi :

On construit un arbre (dont les sommets seront des caractères ou groupes de caractères). Cet arbre sera binaire parce que de chaque sommet partiront deux branches. C'est cet arbre qui servira au codage compressé.
Chaque sommet de l'arbre sera accompagné de sa fréquence comptée en millièmes.
Au départ, les 32 sommets sont isolés (non connectés). On les connectera progressivement par fréquences croissantes. (Rappelons qu'un arbre est connexe par définition !).

Ainsi au départ on a {E} (144) {*blanc*} (128) {S} (62) -------- {Z} (5) {W} (5).

En bas de l'arbre **on groupe les deux caractères les moins fréquents** : ici W (5) et Z (5). Ces deux sommets fils sont reliés à un sommet père qui prend le nom de la réunion des 2 sommets fils et dont la fréquence est la somme de leurs fréquences :

Départ de l'algorithme :

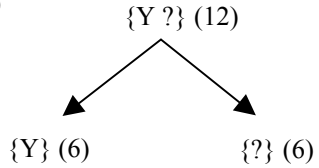


{*interrogation*} (6) {Y} (6)
{X} (6) {K} (7) etc.

L'alphabet transformé n'a donc plus que 31 "caractères" : {E} (144) ... {W Z} (10)

On réitère cette étape tant qu'il reste des lettres isolées :

Les 2 caractères les moins fréquents sont maintenant {Y} (6) et {*interrogation*} (6) d'où le nouveau sommet {Y ?} (12)



Les 2 caractères les moins fréquents sont maintenant {K} (7) et {X} (6) d'où le nouveau sommet {K X} (13) ;

ensuite on a {H} (7) et {J} (7) d'où le nouveau sommet {H J} (14);
ensuite on a {F} (7) et {G} (7) d'où le nouveau sommet {F G } (14);
ensuite on a {*virgule*} (7) et {*tiret*} (7) d'où le nouveau sommet {, -} (14);
ensuite on a {Q} (8) et {B}(7) d'où le nouveau sommet {Q B} (15);

On a la situation provisoire suivante (les fréquences sont classées au fur et à mesure) :

	Fréquence ×1000		Fréquence ×1000		Fréquence ×1000
E	144	O	48	{, -}	14
Blanc	128	L	43	V	13
S	62	D	33	*apostrophe*	13
A	62	C	26	{K X}	13
N	62	M	24	{Y ?}	12
I	58	P	21	{W Z}	10
R	57	{Q B}	15	*point*	8
T	57	{F G}	14		
U	49	{H J}	14		

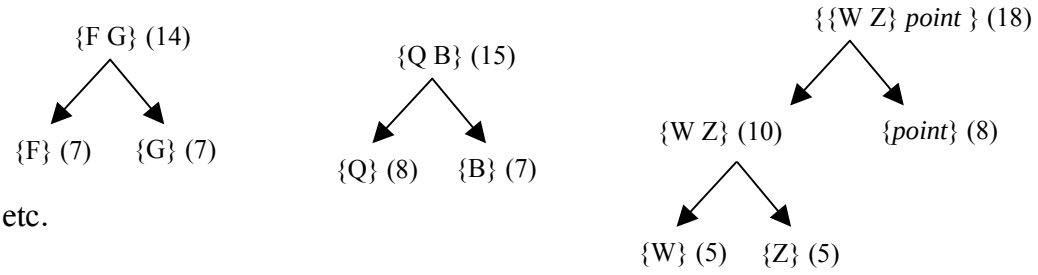
Attention aux étapes suivantes :

Les deux "caractères" les moins fréquents (voir la fin du tableau ci-dessus) sont maintenant {*point*} (8) et {W Z} (10) d'où le nouveau sommet {{W Z} {*point*}} (18);
Les deux "caractères" les moins fréquents sont désormais {Y ?} (12) et {K X} (13) d'où le nouveau sommet {{Y ?} {K X}} (25);
Les deux "caractères" les moins fréquents sont ensuite {V} (13) et {*apostrophe*} (13) d'où le nouveau sommet {V *apostrophe*} (26) = {V '} (26);
Les deux "caractères" les moins fréquents sont ensuite {, -} (14) et {H J} (14) d'où le nouveau sommet {{H J}{, -}} (28);

On a la situation intermédiaire suivante (les fréquences sont classées) :

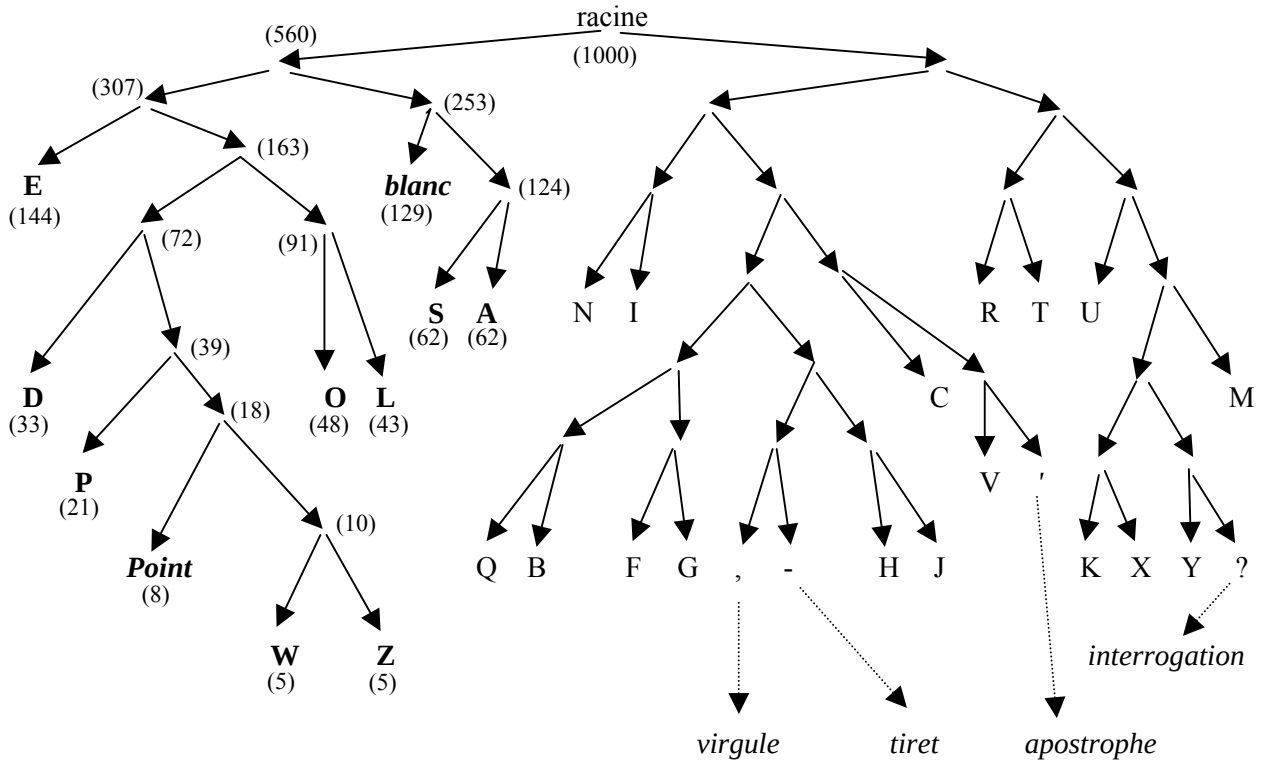
	Fréquence ×1000		Fréquence ×1000		Fréquence ×1000
E	144	O	48	{{W Z} *point}*	18
Blanc	128	L	43	{Q B}	15
S	62	D	33	{F G}	14
A	62	{{H J}{, -}}	28		
N	62	*{V apostrophe}*	26		
I	58	C	26		
R	57	{{K X}{Y ?}]	25		
T	57	M	24		
U	49	P	21		

L'arbre, se construit donc au fur et à mesure à partir du bas.
Actuellement, le bas de l'arbre a l'allure suivante :



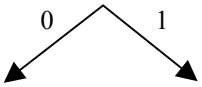
etc.

Voici ce qu'on obtient quand tout est fini :



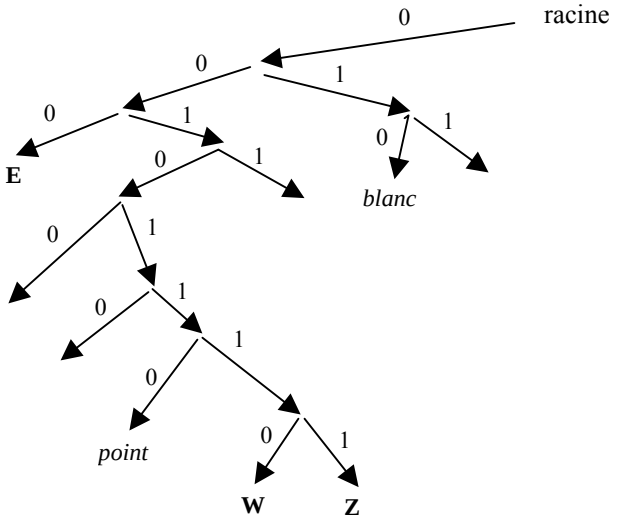
J'ai indiqué les fréquences (× 1000) uniquement dans la partie gauche de l'arbre.

Par construction la fréquence de chaque sommet non terminal est la somme des fréquences de ses deux fils, la fréquence de la racine est donc 1000, et plus un caractère est rare, plus il est situé bas dans l'arbre. On voit bien que pour économiser les chiffres binaires, il suffit de prendre comme longueur du code d'un caractère son niveau (profondeur) dans l'arbre.
C'est facile : on convient que pour chaque père, le fils gauche est codé 0 et le fils droit 1 :



Chaque branche est ainsi codée 0 ou 1, et on définit le **code BINAIRE-COMPRESSÉ** d'un caractère comme le nombre binaire obtenu en prenant dans l'ordre les bits des branches du chemin qui va de la racine de l'arbre au caractère (chemin qui est unique par définition d'un arbre).

Ainsi les 2 caractères les plus fréquents E et *blanc* seront codés respectivement 000 et 010.
W lui, sera codé 00101110.



Voici le tableau complet du codage binaire compressé :

Caractère	Code BINAIRE-COMPRESSÉ	caractère	Code BINAIRE-COMPRESSÉ
A	0111	P	001010
Blanc	010	Q	1010000
B	1010001	R	1100
C	10110	S	0110
D	00100	T	1101
E	000	U	1110
F	1010010	V	101110
G	1010011	W	00101110
H	1010110	X	1111001
I	1001	Y	1111010
J	1010111	Z	00101111
K	1111000	*apostrophe*	101111
L	00111	*virgule*	1010100
M	111111	*tiret*	1010101
N	1000	*point*	0010110
O	00110	*interrogation*	1111011

- On peut se demander pourquoi on ne prend pas le code 0 pour E puisque c'est le caractère le plus fréquent. De même on prendrait 1 pour *blanc* qui vient juste après etc.

Oui, mais il y a le problème du décodage !

Si pour compresser on prenait le codage disons : (E ; 0) (*blanc* ; 1) (S ; 00) (A ; 01) (N ; 11) etc. en suivant bêtement les fréquences, on serait bien embêté au moment de la décompression pour déchiffrer le message "0100110....." : en effet :
Le code commence par 0. Le début serait donc "E" ?
Mais ce même code commence par 01. Le début serait donc "A" ? etc.

On voit bien le problème : pour décompresser, il faut que dans le code BINAIRE-COMPRESSÉ, **aucun code ne soit le préfixe d'un autre code.**

Ceci est garanti par le fait que seuls les sommets terminaux de l'arbre (appelés aussi les feuilles de l'arbre) sont codés.

- Calculons le "Gain" réalisé par le codage BINAIRE-COMPRESSÉ.

Chacun des 32 caractères a la fréquence f_k et un nombre de bits B_k.

Donc le nombre moyen de bits par caractère est de $\sum_{k=1}^{k=32} f_k B_k = 4{,}306$.

Au lieu de 5 par le BINAIRE-1 ordinaire. Le gain est donc de 13,88 % en moyenne.
C'est-à-dire que pour tout texte respectant à peu près les fréquences données précédemment, (quasiment tous les textes en français) on aura un gain d'environ 14 % avec cette compression.

- RÉSUMONS : Le **protocole de la compression** est donc le suivant :

> - On part du texte clair (message M_0) supposé ne contenir que les 32 caractères sélectionnés.
>   On convertit chaque caractère de M_0 (avec les blancs) à l'aide de BINAIRE-COMPRESSÉ.
>   On obtient le message M_1 sous forme d'un nombre binaire.
> - On partage ce nombre M_1 en tranches de 5 bits à partir de la gauche.
>
> *Si le nombre de bits de la dernière tranche n'est pas multiple de 5, on la complète par des 0 ; cela donne un nombre binaire* M_1'
>
> - Enfin on convertit chaque tranche (5 bits) de M_1' en un caractère à l'aide de BINAIRE-1.
>   Cela donne le message final compressé : M_2.

Exemple : le texte clair est M_0 = "**TEL** | **PERE** | **TEL** | **FILS**"
On convertit à l'aide de BINAIRE-COMPRESSÉ :

T E L P E R E T E L F I L S

M_1 = "1101 000 00111 **010** 001010 000 1100 000 **010** 1101 000 00111 **010** 1010010 1001 00111 0110"

On partage ce nombre en tranches de 5 bits à partir de la gauche, la dernière tranche de 4 bits est complétée par un 0 :
M_1' = "11010 00001 11010 00101 00001 10000 00101 10100 00011 10101 01001 01001 00111 0110**0**"
On convertit chaque tranche de M_1' en un caractère à l'aide de BINAIRE-1 :

M_1' = "11010 00001 11010 00101 00001 10000 00101 10100 00011 10101 01001 01001 00111 0110**0**"

Z *blanc* Z E *blanc* P E T C U I I G L

Cela donne le message final compressé : M_2 = "**Z|ZE|PETCUIIGL**"

On est passé de 17 caractères dans M_0 à 14 caractères dans M_2. Ici, on gagne 17,6 %.

- On pourrait essayer de compresser à nouveau M_2. Mais en général, la seconde compression ne donne rien, car la première est optimisée.

- Remarque :
  Le message final M_2 ne commence jamais par un *blanc* et ne finit jamais par un *blanc*.

En effet : le *blanc* de BINAIRE-1 est codé astucieusement 00001.

M_1 ne peut pas commencer par 00001 puisque aucun code de BINAIRE-COMPRESSÉ ne commence par quatre 0.
M_1 ne peut pas finir par 00001 puisque :
- ou bien M_1 a été complété par un ou plusieurs 0 donc ne se termine pas par 1 ;
- ou bien M_1 avait un nombre de bits multiple de 5, mais 00001 n'est pas un code de BINAIRE-COMPRESSÉ.

• La décompression s'effectue sans problème. Exemple :
Le message compressé M_2 = "Z|ZE|PETCUIIGL" est codé avec BINAIRE-1. On obtient :
M_1' = "11010 00001 11010 00101 00001 10000 00101 10100 00011 10101 01001 01001 00111 0110**0**"
M_1' est décodé avec BINAIRE-COMPRESSÉ. On a vu que c'était possible de manière unique.
Bien entendu, si à la fin, les derniers 0 de M_1' ne correspondent à aucun code, c'est qu'ils ont été ajoutés pour obtenir un nombre de bits multiple de 5 et on les ignore purement et simplement.
C'est plus facile si on a le tableau BINAIRE-COMPRESSÉ classé par codes binaires croissants :

caractère	Code BINAIRE-COMPRESSÉ	caractère	Code BINAIRE-COMPRESSÉ
000	E	111111	M
010	*blanc*	0010110	*point*
0110	S	1010000	Q
0111	A	1010001	B
1000	N	1010010	F
1001	I	1010011	G
1100	R	1010100	*virgule*
1101	T	1010101	*tiret*
1110	U	1010110	H
00100	D	1010111	J
00110	O	1111000	K
00111	L	1111001	X
10110	C	1111010	Y
001010	P	1111011	*interrogation*
101110	V	00101110	*W*
101111	*apostrophe*	00101111	Z

Allons-y de gauche à droite :

M_1' = "1101000001110100010100001100000010110100000111010101001010010011101100".

1 n'est pas un code.
11 non plus.
110 non plus.
1101 correspond à T.
La suite 000 correspond à E . Puis 00111 correspond à L. Soit le début : "TEL"
Il reste après "TEL" : "010 001010 000 1100 000 010 1101 000 00111 010 1010010 1001 00111 01100"
qui est converti en ... | PERE|TEL|FIL ...
Et enfin, en ignorant le 0 terminal, 0110 est converti en "S"
Nous retrouvons bien M_0 = "TEL | PERE | TEL | FILS".

Bibliographie : TANGENTE Hors série Numéro 12.