

Compression

Compression statistique

E. Jeandel

Compression

- Un outil de compression est la donnée de deux outils :
 - Un programme de compression qui prend un flux de données et qui le convertit en un flux compressé ;
 - Un programme de décompression qui prend un flux compressé et le décompresse.

Deux types de compression

- Une fois décompressé, on réobtient le flux d'origine (compression sans pertes) ;
- Une fois décompressé, on obtient un flux très semblable au flux d'origine, mais éventuellement différent (compression avec pertes) ;

Dans les premiers cours, on ne fait que de la compression sans pertes.

Compression sans pertes

Proposition 1. *Aucun outil de compression n'est efficace : On peut trouver pour tout n une chaîne de longueur n qu'il ne compresses pas, même d'un seul caractère.*

- Démonstration.* – Il y a 256^n chaînes de longueur n .
- Il y a $1 + 256 + \dots + 256^{n-1} = \frac{256^n - 1}{256 - 1}$ chaînes de longueur strictement inférieure à n .
 - $256^n > \frac{256^n - 1}{256 - 1}$, donc si le compresseur compresses toutes les chaînes de longueur n , il va en compresser deux de la même façon.
 - Contradiction. □

Ce qu'on peut faire

- Un outil de compression ne peut donc pas toujours compresser, ne serait-ce que d'un octet
- Les outils disponibles s'efforcent donc de ne produire des bons résultats que dans des cas particulier :
 - Documents textes ;
 - Documents produisant des structures repérables ;

Méthode 2 (exemple)

␣	00000	0	h	01000	8	p	10000	16	x	11000	24
a	00001	1	i	01001	9	q	10001	17	y	11001	25
b	00010	2	j	01010	10	r	10010	18	z	11010	26
c	00011	3	k	01011	11	s	10011	19	.	11011	27
d	00100	4	l	01100	12	t	10100	20	,	11100	28
e	00101	5	m	01101	13	u	10101	21	'	11101	29
f	00110	6	n	01110	14	v	10110	22	!	11110	30
g	00111	7	o	01111	15	w	10111	23	?	11111	31

qqqqqqqqooooabpppppppqqmYYYYYYYYYYYYYYYYYYYYYYY
 → !qg !oe ab!pg qm!yw a

Méthode 2

- Pour coder un seul !, il faut écrire ! ! a
- Si une lettre n'apparaît que deux fois de suite, il vaut mieux l'écrire deux fois.
- Dans le meilleur des cas, un texte de n octets peut se retrouver compressé en $\dots 3n/31$ octets
- Dans le pire des cas, un texte de n octets peut se retrouver compressé en $\dots 2n$ octets

Format utilisé en pratique dans de nombreux formats de compression (PCX pour les images)

Méthode 3

- Pour coder un seul x , on utilise x
- Pour coder deux x , on utilise xx
- Pour coder plus que 3 fois x , on utilise $xxxy$ où y représente le nombre de fois où x apparaît dans la suite.

Méthode 3 (exemple)

␣	00000	0	h	01000	8	p	10000	16	x	11000	24
a	00001	1	i	01001	9	q	10001	17	y	11001	25
b	00010	2	j	01010	10	r	10010	18	z	11010	26
c	00011	3	k	01011	11	s	10011	19	.	11011	27
d	00100	4	l	01100	12	t	10100	20	,	11100	28
e	00101	5	m	01101	13	u	10101	21	'	11101	29
f	00110	6	n	01110	14	v	10110	22	!	11110	30
g	00111	7	o	01111	15	w	10111	23	?	11111	31

qqqqqqqqooooabpppppppqqmYYYYYYYYYYYYYYYYYYYYYYY
 → qqgd ooob abpppd qmyyt a

1 Compression RLE (Run Length Encoding)

Principe

- Il arrive souvent qu'un même caractère apparaisse plusieurs fois consécutivement
- On remplace 10 occurrences du caractère a par la séquence $(a, 10)$

Exemple

qqqqqqqqooooabpppppppqqmYYYYYYYYYYYYYYYYYYYYYYY
 → (q, 6) (o, 5) (a, 1) (b, 1) (p, 6) (q, 1) (m, 1) (y, 23) (a, 1)

Il reste à expliquer comment coder des couples (x, i)

Méthode 1

- On représente n occurrences de la lettre x par xw où w représente n en binaire.

␣	00000	0	h	01000	8	p	10000	16	x	11000	24
a	00001	1	i	01001	9	q	10001	17	y	11001	25
b	00010	2	j	01010	10	r	10010	18	z	11010	26
c	00011	3	k	01011	11	s	10011	19	.	11011	27
d	00100	4	l	01100	12	t	10100	20	,	11100	28
e	00101	5	m	01101	13	u	10101	21	'	11101	29
f	00110	6	n	01110	14	v	10110	22	!	11110	30
g	00111	7	o	01111	15	w	10111	23	?	11111	31

qqqqqqqqooooabpppppppqqmYYYYYYYYYYYYYYYYYYYYYYY
 → qq oe aabapq qamayw a a

Méthode 1

- Dans le meilleur des cas, un texte de n octets peut se retrouver compressé en $\dots 2n/31$ octets
- Dans le pire des cas, un texte de n octets peut se retrouver compressé en $\dots 2n$ octets
- Comment améliorer ?

Méthode 2

- On choisit dans l'alphabet une lettre quelconque (dans l'exemple le point d'exclamation : !)
- On représente une occurrence de la lettre x par x
- On représente n occurrences de la lettre x par $!xw$ où w représente n en binaire.

Méthode 3

- Pour coder trois c , il faut écrire $ccc␣$
 - Dans le meilleur des cas, un texte de n octets peut se retrouver compressé en $\dots 4n/34$ octets
 - Dans le pire des cas, un texte de n octets peut se retrouver compressé en $\dots 4n/3$ octets
- Format utilisé en pratique (première étape de MNP5, dans les modems)

2 Compression statistique

Compression statistique

Principe 2. *La compression statistique utilise comme principale source d'information pour compresser la répartition des lettres dans le texte, et plus précisément leur fréquence d'apparition.*

on met un peu la poussière sur le tapis et on la laisse pour les autres

a	5	e	9	i	3	l	5
m	1	n	3	o	4	p	4
r	4	s	8	t	4	u	6
␣	15						

- Taille du texte : 71 octets
- Comment compresser ce texte ?

Comment compresser ce texte ?

- Il n'y a que 13 caractères, donc au lieu de représenter chacun sur 5 bits, on peut les représenter seulement sur 4 bits :

a	0000	e	0001	i	0010	l	0011
m	0100	n	0101	o	0110	p	0111
r	1000	s	1001	t	1010	u	1011
␣	1100	b	1101	c	1110	d	1111

o n ␣ m e t ␣ u n p e u
 01100101110001000001101010010110101100011100011011

- Taille du texte : 4×71 bits, soit 57 octets

Comment compresser ce texte ?

- On peut encore changer le codage :

a	0000	e	0001	i	0010	l	0011
m	0100	n	0101	o	0110	p	0111
r	1000	s	1001	t	1010	u	1100
␣	1111						

o n ␣ m e t ␣ u n p e u
 01100101111010000011011111001011101100011011

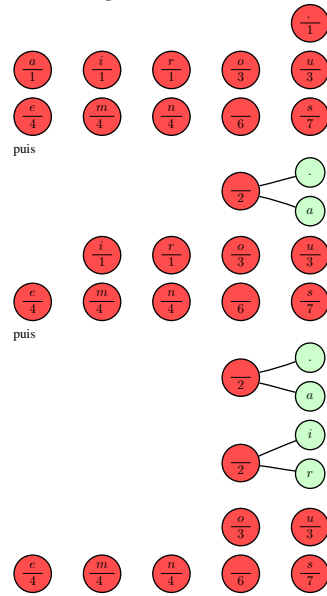
- Taille du texte : 259 bits soit 52 octets

Codage de Huffman

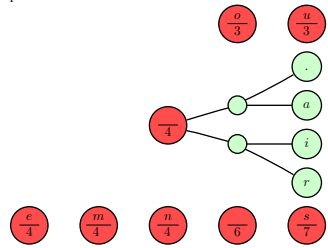
Principe de l'algorithme

- On représente chaque lettre par un sommet. Le poids d'un sommet est la fréquence de la lettre.
- A chaque étape, on relie ensemble les deux sommets les plus petits pour former un nouveau sommet dont le poids est la somme des poids.

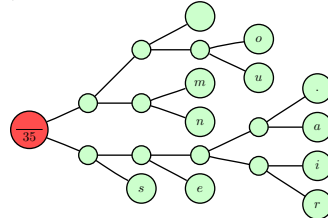
Déroulement de l'algorithme



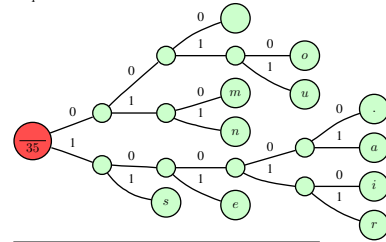
puis



enfin



ce qui donne



ω	000	o	0010	u	0011	m	010
n	011	.	10000	a	10001	i	10010
r	10011	e	101	s	11		

Codage de Huffman

ω	000	o	0010	u	0011	m	010
n	011	.	10000	a	10001	i	10010
r	10011	e	101	s	11		

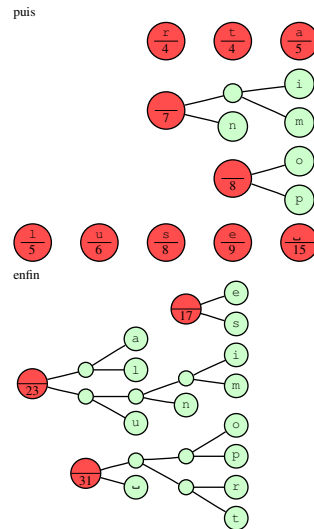
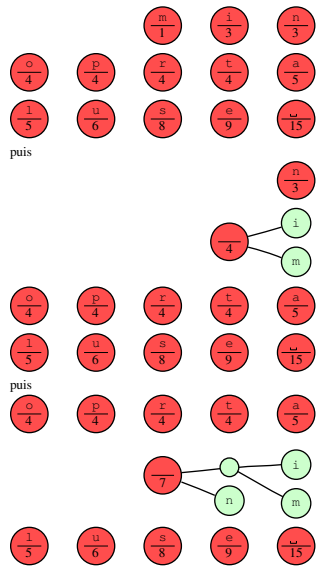
ω	6	n	4
a	1	u	3
e	4	r	1
i	1	s	7
m	4	.	1
o	3		

Taille du texte : $2 \times 7 + 3 \times (6 + 4 + 4 + 4) + 4 \times (3 + 3) + 5 \times (1 + 1 + 1 + 1) = 112$ bits, soit 22.4 octets.

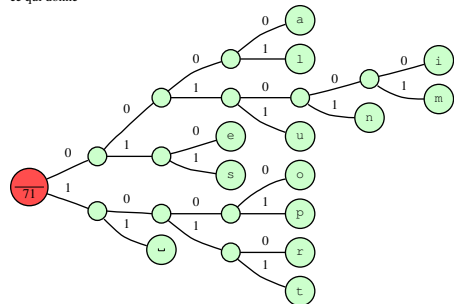
Exercice

a	5	e	9	i	3	l	5
m	1	n	3	o	4	p	4
r	4	s	8	t	4	u	6
ω	15						

Correction



ce qui donne



Correction

a	0000	l	0001	i	001000	m	001001
n	00101	u	0011	e	010	s	011
o	1000	p	1001	r	1010	t	1011
v	11						

248 bits (50 octets)

a	5	e	9	i	3	l	5
m	1	n	3	o	4	p	4
r	4	s	8	t	4	u	6
v	15						

Théorie

- Est-ce qu'on peut faire encore mieux ?

Théorème 3. Le(s) code(s) de Huffman est le meilleur code préfixe pour une fréquence d'apparition des caractères donnés.

Définition 1. On note p_c la fréquence d'apparition de c . L'entropie du message est $H = \sum_c -p_c \log_2 p_c$.

Théorème 4. Aucun code ne peut faire mieux que l'entropie.

Théorie - Retour à l'exemple

c	p_c	$-p_c \log p_c$	c	p_c	$-p_c \log p_c$
u	0.085	0.301249	o	0.056	0.233789
t	0.056	0.233789	n	0.042	0.192878
v	0.211	0.473843	a	0.070	0.269565
s	0.113	0.354901	e	0.127	0.377724
l	0.042	0.192878	m	0.014	0.086616
r	0.056	0.233789	l	0.070	0.269565
p	0.056	0.233789			

L'entropie est ici de 246 bits, soit 50 octets. Huffman n'est pas très loin !

Conclusion

- La compression statistique utilise les fréquences d'apparition des caractères pour compresser
- La méthode de Huffman est la meilleure méthode de compression statistique. ...
- ... mais ce n'est pas la meilleure méthode de compression. En pratique la compression statistique est toujours couplée à d'autres algorithmes dans des outils de compression (comme bzip2 et gzip). On l'utilise principalement comme dernière étape dans le déroulement de la compression.

Déroulement de Huffman

- On calcule la fréquence d'apparition de chaque caractère
- On construit l'arbre de Huffman
- On compress le texte en utilisant cet arbre

Problèmes ?

Huffman adaptatif

- On lit le texte caractère après caractère
- On construit au fur et à mesure la table des fréquences d'apparition des caractères lus avant
- On construit à chaque étape un arbre de Huffman en fonction des fréquences d'apparition des caractères
- On compress le texte en utilisant ces arbres

Problèmes ?

Prédiction

- On lit caractère après caractère du texte
 - On détermine, à partir des caractères lus avant, la probabilité que le prochain caractère soit x
 - On construit à chaque étape un arbre de Huffman en fonction de ces probabilités
 - On compress le texte en utilisant ces arbres
- L'efficacité de la compression est fonction de l'heuristique utilisée à la deuxième étape. On obtient ainsi la plupart des meilleurs algorithmes connus.