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 eventuellement 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 compresse pas, même d'un seul caractère.*

Démonstration.

- Il y a 256^n chaînes de longueur n.
- Il y a $1+256+\cdots+256^{n-1} = \frac{256^n-1}{256-1}$ chaînes de longueur strictement inférieur à n.
- $256^n > \frac{256^n-1}{256-1}$, donc si le compresseur compresse 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 ;

1 Compression RLE (Run Length Encoding)

Principe

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

Exemple

```
qqqqqqqooooooabppppppppqmyyyyyyyyyyyyyyyyyyyyyyya
→
(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 occurences 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

```
qqqqqqqooooooabppppppppqmyyyyyyyyyyyyyyyyyyyyyyya
→
qg      oe    aababpg    qamayw                      aa
```

Méthode 1

- Dans le meilleur des cas, un texte de n octets peut se retrouver compressé en $\ldots 2n/31$ octets
- Dans le pire des cas, un texte de n octets peut se retrouver compressé en $\ldots 2n$ 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 occurences de la lettre x par $!xw$ où w représente n en binaire.

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

```
qqqqqqqooooobpppppppqmyyyyyyyyyyyyyyyyyyyyyyya
→
!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 $\ldots 3n/31$ octets
- Dans le pire des cas, un texte de n octets peut se retrouver compressé en $\ldots 2n$ 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

```
qqqqqqqooooobpppppppqmyyyyyyyyyyyyyyyyyyyyyyya
→
qqqd   ooob abpppd   qmyyyt                    a
```

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 $\ldots 4n/34$ octets
- Dans le pire des cas, un texte de n octets peut se retrouver compressé en $\ldots 4n/3$ 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 poussiere 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
01100101110001000001101011001011010111000111000 11011

- 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	101	u	110
␣	111						

o n ␣ m e t ␣ u n ␣ p e u
01100101111010000011011111100101111011100011 10

- Taille du texte : 259 bits soit 52 octets

Compression statistique

L'objectif de la compression statistique est

- De construire un code préfixe
- Qu'il soit le plus possible adapté

Il faut donc s'arranger pour obtenir des codes petits pour des lettres fréquentes.

Deux algorithmes :

- Shannon-Fano
- Huffman

Shannon-Fano

- Calculer la fréquence de chaque lettre
- Classer les lettres par ordre décroissant de fréquences
- Diviser le tableau en deux parties, de fréquences à peu près égales. La première partie sera codé par des mots commençant par 0, la deuxième par 1.
- Subdiviser les parties obtenues récursivement, toujours de la même façon

Shannon-Fano

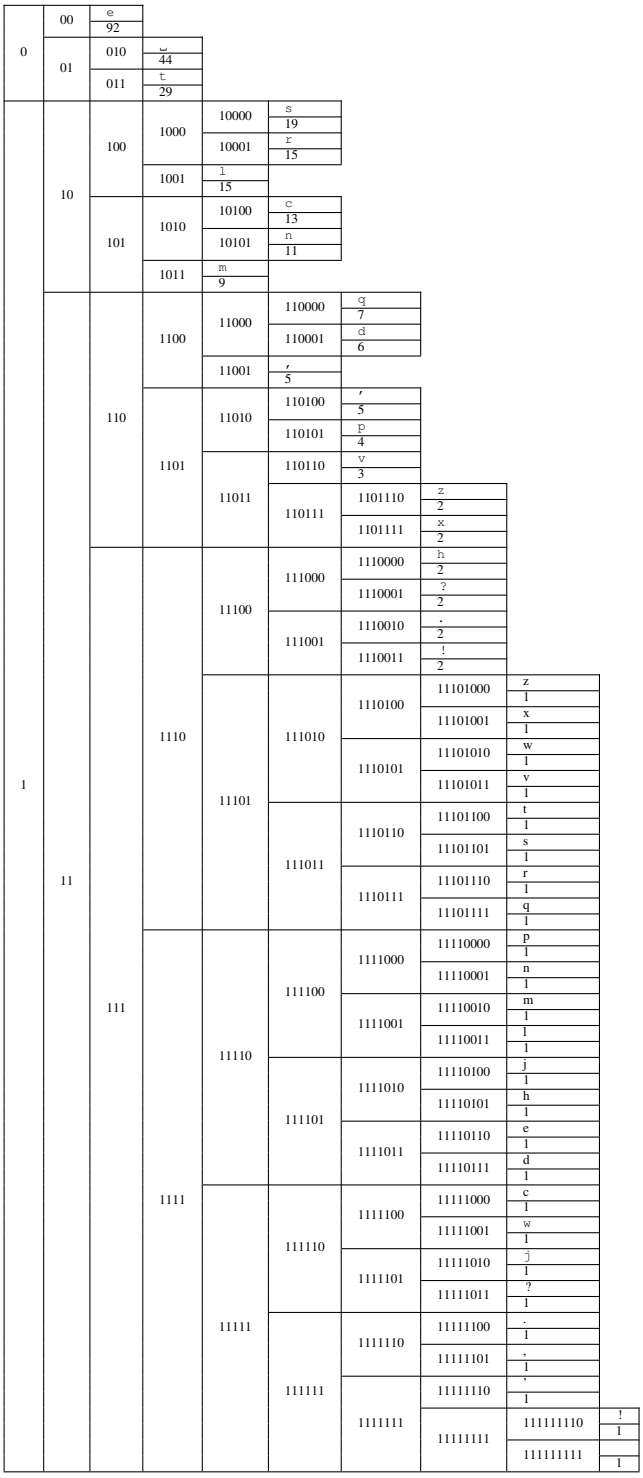
␣	44	h	2	p	4	x	2
a	0	i	0	q	7	y	0
b	0	j	1	r	15	z	2
c	13	k	0	s	19	.	2
d	6	l	15	t	29	,	5
e	92	m	9	u	0	’	5
f	0	n	11	v	3	!	2
g	0	o	0	w	1	?	2

e	␣	t	s	r	l	c	n	m	q	d	,	’	p	v	z	x	h	?	.	!	w	j
92	44	29	19	15	15	13	11	9	7	6	5	5	4	3	2	2	2	2	2	2	1	1

0	e	␣	t																			
	92	44	29																			
1	s	r	l	c	n	m	q	d	,	’	p	v	z	x	h	?	.	!	w	j		
	19	15	15	13	11	9	7	6	5	5	4	3	2	2	2	2	2	2	1	1		

0	00	e																				
		92																				
	01	␣	t																			
		44	29																			
1	10	s	r	l	c	n																
		19	15	15	13	11																
	11	m	q	d	,	’	p	v	z	x	h	?	.	!	w	j						
		9	7	6	5	5	4	3	2	2	2	2	2	2	1	1						

0	00	e										
		92										
	01	010	␣									
			44									
		011	t									
			29									
1	10	100	s	r	l							
			19	15	15							
		101	c	n								
			13	11								
	11	110	m	q	d	,						
			9	7	6	5						
		111	'	p	v	z	x	h	?	.	!	w	j
			5	4	3	2	2	2	2	2	2	1	1


0
00
e
92
01
010
␣
44
011
t
29
1
10
100
1000
10000
s
19
10001
r
15
1001
l
15
101
1010
10100
c
13
10101
n
11
1011
m
9
11
110
1100
11000
110000
q
7
110001
d
6
11001
,
5
1101
11010
110100
'
5
110101
p
4
11011
110110
v
3
110111
1101110
z
2
1101111
x
2
111
1110
11100
111000
1110000
h
2
1110001
?
2
111001
1110010
.
2
1110011
!
2
11101
111010
1110100
11101000
z
1
11101001
x
1
1110101
11101010
w
1
11101011
v
1
111011
1110110
11101100
t
1
11101101
s
1
1110111
11101110
r
1
11101111
q
1
1111
11110
111100
1111000
11110000
p
1
11110001
n
1
1111001
11110010
m
1
11110011
l
1
111101
1111010
11110100
j
1
11110101
h
1
1111011
11110110
e
1
11110111
d
1
11111
111110
1111100
11111000
c
1
11111001
w
1
1111101
11111010
j
1
11111011
?
1
111111
1111110
11111100
.
1
11111101
,
1
1111111
11111110
'
1
11111111
111111110
!
1
111111111
1

e	00	␣	010	t	011	s	10000
r	10001	l	1001	c	1010	n	1011
m	11000	q	11001	d	11010	,	11011
’	111000	p	111001	v	111010	z	111011
x	1111000	h	1111001	?	111101	.	1111100
!	1111101	w	1111110	j	1111111		

␣	44	h	2	p	4	x	2
a	0	i	0	q	7	y	0
b	0	j	1	r	15	z	2
c	13	k	0	s	19	.	2
d	6	l	15	t	29	,	5
e	92	m	9	u	0	’	5
f	0	n	11	v	3	!	2
g	0	o	0	w	1	?	2

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
␣	15						

Shannon-Fano - Correction

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

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

0	00	␣	e			
		15	9			
	01	s	u			
		8	6			
1	10	l	a	t	r	
		5	5	4	4	
	11	p	o	n	i	m
		4	4	3	3	1

0	00	000	␣			
			15			
		001	e			
			9			
	01	010	s			
			8			
		011	u			
			6			
1	10	100	l	a		
			5	5		
		101	t	r		
			4	4		
	11	110	p	o		
			4	4		
		111	n	i	m	
			3	3	1	

0	00	000	␣			
			15			
		001	e			
			9			
	01	010	s			
			8			
		011	u			
			6			
1	10	100	1000	l		
				5		
			1001	a		
				5		
		101	1010	t		
				4		
			1011	r		
				4		
	11	110	1100	p		
				4		
			1101	o		
				4		
		111	1110	n		
				3		
			1111	11110	i	
					3	
				11111	m	
					1	

␣	000	e	001	s	010	u	011
l	1000	a	1001	t	1010	r	1011
p	1100	o	1101	n	1110	i	11110
m	11111						

250 bits (50 octets)

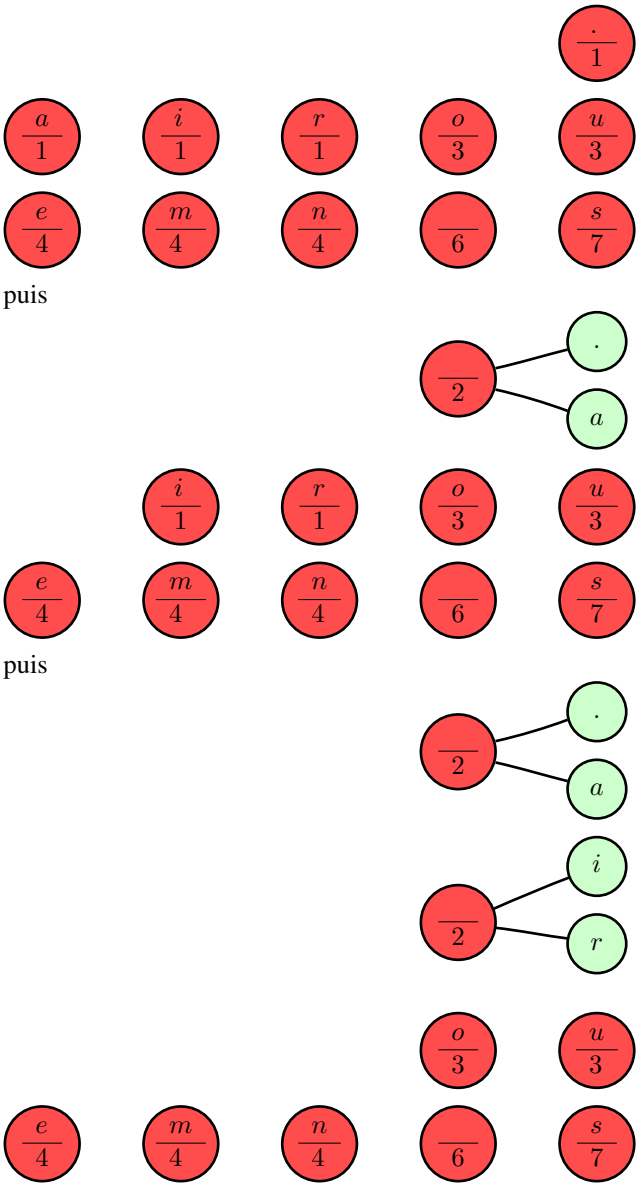
a	5	e	9	i	3	l	5
m	1	n	3	o	4	p	4
r	4	s	8	t	4	u	6
␣	15						

Codage de Huffman

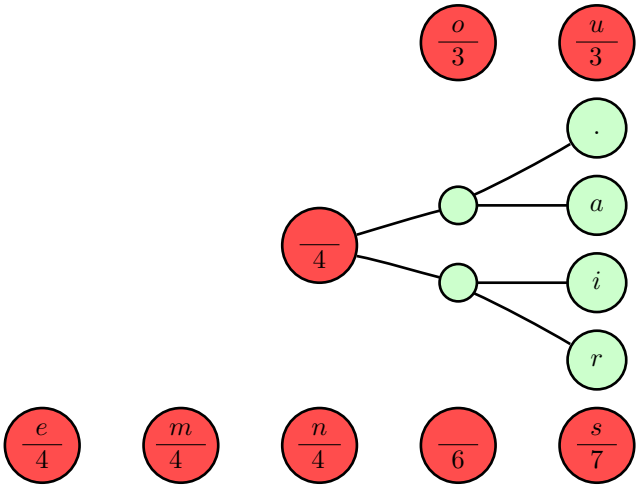
Principe de l'algorithme

- 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.

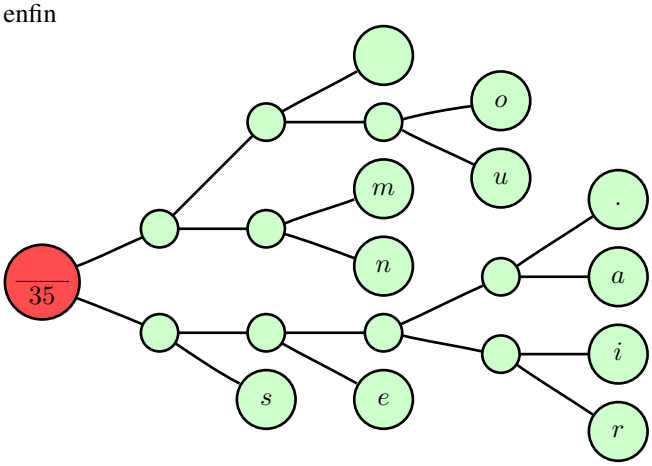
Déroulement de l'algorithme



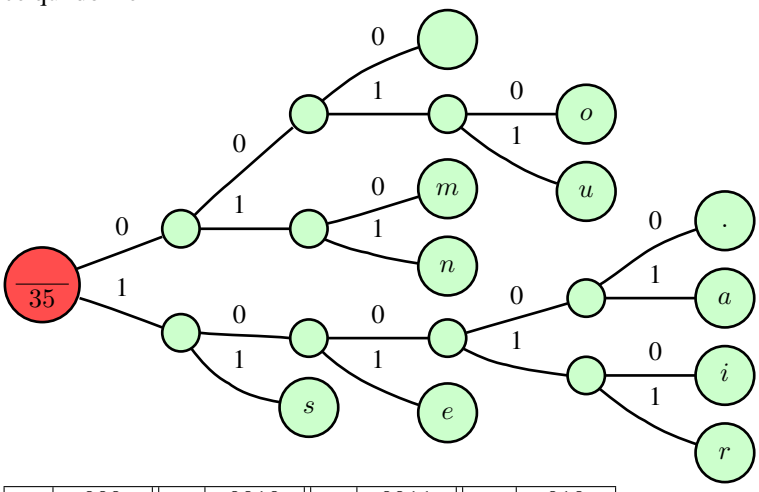
puis



enfin



ce qui donne



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

Codage de Huffman

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

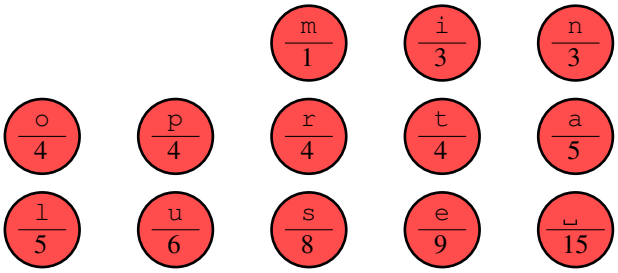
␣	6	n	4
a	1	u	3
e	4	r	1
i	1	s	7
m	4	.	1
o	3		

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.

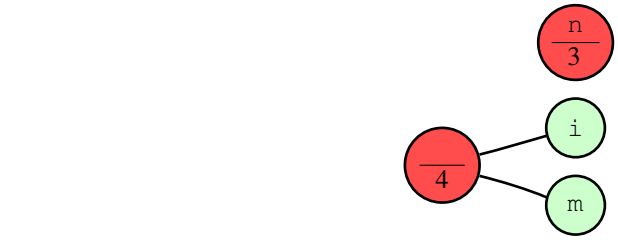
Exercice

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

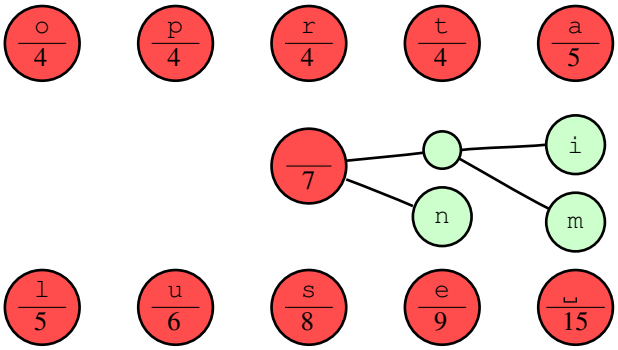
Correction


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

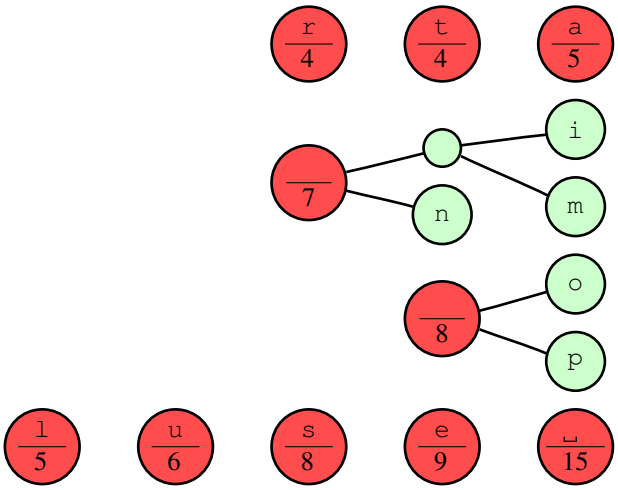
puis


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

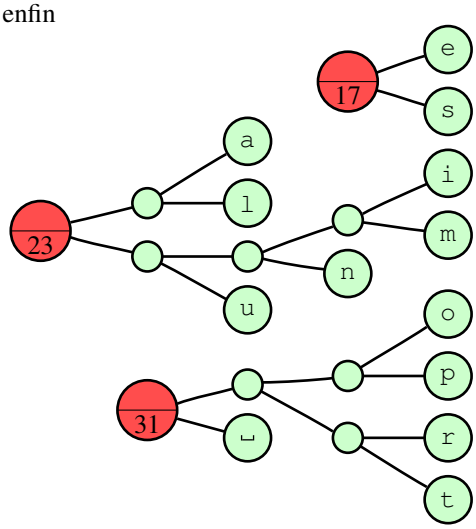
puis


o 4
p 4
r 4
t 4
a 5
7
i
n
m
l 5
u 6
s 8
e 9
␣ 15

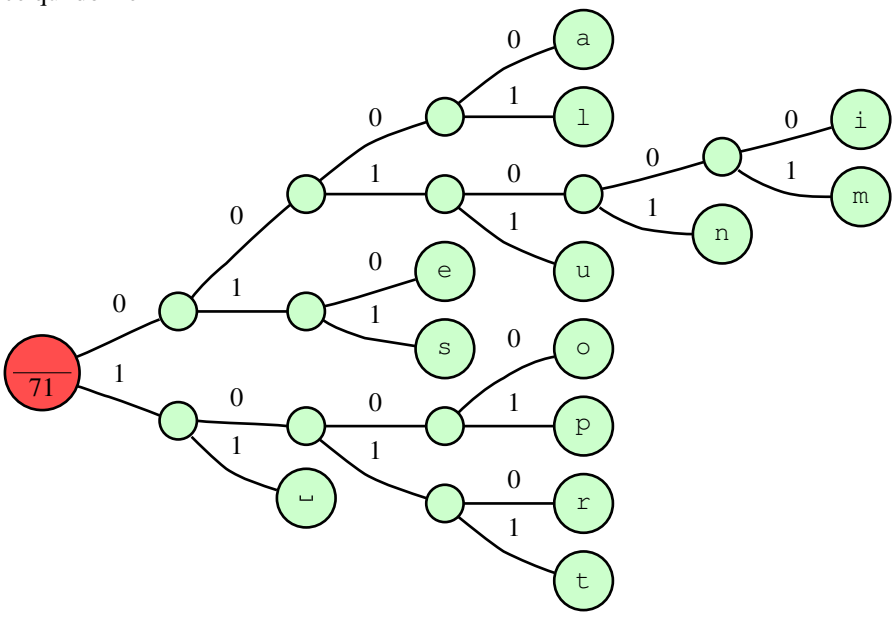
puis



enfin



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
␣	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
␣	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'apparence 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
␣	0.211	0.473843	a	0.070	0.269565
s	0.113	0.354901	e	0.127	0.377724
i	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 compresse 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 compresse 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 compresse 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.