COMRESSION

I. Introduction
II. Méthodes statistiques
    Introduction à la théorie de l'information
    Codes à taille variable
        Codage d'Huffman
        Codage de Shannon-Fano
III. Méthodes de compression avec mémoire
    Codage arithmétique
    Méthodes à base de dictionnaire
        LZ77
        LZW

www.univ-rouen.fr/psi/paquet/

Introduction (1)

1. Compression

Pour communiquer plus vite

Téléphonie:	**1972**	**1984**	**1991**	**1994**
	64 Kbit/s	**32 Kbit/s**	**16 Kbit/s**	**8 Kbit/s**
GSM:	**1989**			
	13 Kbit/s			
UMTS:	**8, 4, 2, 1 Kbit/s**			

Pour archiver davantage

Fichiers textes ⇒ **Sans perte**

Voix, Musique & Vidéo ⇒ **Avec perte,**

Perception satisfaisante

Introduction (2)

2. Traitements numériques

Fiabilité, Coûts, Intégration (VLSI, DSP)

Quantification des échantillons 16bit ⇒ **Bande large**

Réduire la bande utile ⇒ **Compression**

3. Pourquoi la compression est-elle possible?

Redondance dans les données qu'on peut Éliminer/Coder

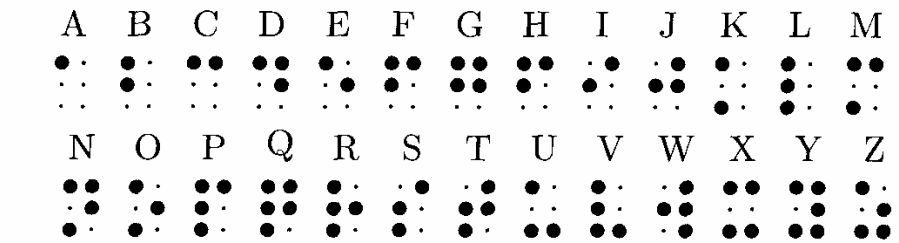
sans dégrader le message

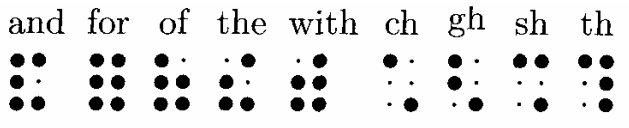
Texte ⇒ **Fréquences, Dictionnaires**

Voix, Images ⇒ **Codage différentiel**

Quelques Exemples…

1. Code Braille 1820: 6 bits par symbole





2. Code Baudot (Télégraphe) 1880: 5 bits par symbole
1950 ordinateurs de première génération

Letters	Code	Figures	Letters	Code	Figures
A	10000	1	Q	10111	/
B	00110	8	R	00111	-
C	10110	9	S	00101	SP
D	11110	0	T	10101	na
E	01000	2	U	10100	4
F	01110	na	V	11101	'
G	01010	7	W	01101	?
H	11010	+	X	01001	,
I	01100	na	Y	00100	3
J	10010	6	Z	11001	:
K	10011	(	LS	00001	LS
L	11011	=	FS	00010	FS
M	01011	)	CR	11000	CR
N	01111	na	LF	10001	LF
O	11100	5	ER	00011	ER
P	11111	%	na	00000	na

Code plus de 32 caractères grâce à LS et FF

LS, Letter Shift; FS, Figure Shift.
CR, Carriage Return; LF, Line Feed.
ER, Error; na, Not Assigned; SP, Space.

Quelques exemples…

3. «Compression avant » pour une liste lexicographique

a	a
aardvark	1ardvark
aback	1back
abaft	3ft
abandon	3ndon
abandoning	7ing
abasement	3sement
abandonment	3ndonment
abash	3sh
abated	3ted
abate	5
abbot	2bot
abbey	3ey
abbreviating	3reviating
abbreviate	9e
abbreviation	9ion

Quelques exemples…

4. Run Length Encoding (RLE): *Longueurs de plages*

Codage des répétitions de symboles

ddddddddddd → *10d*

Nécessité d'un caractère spécial pour annoncer une répétition

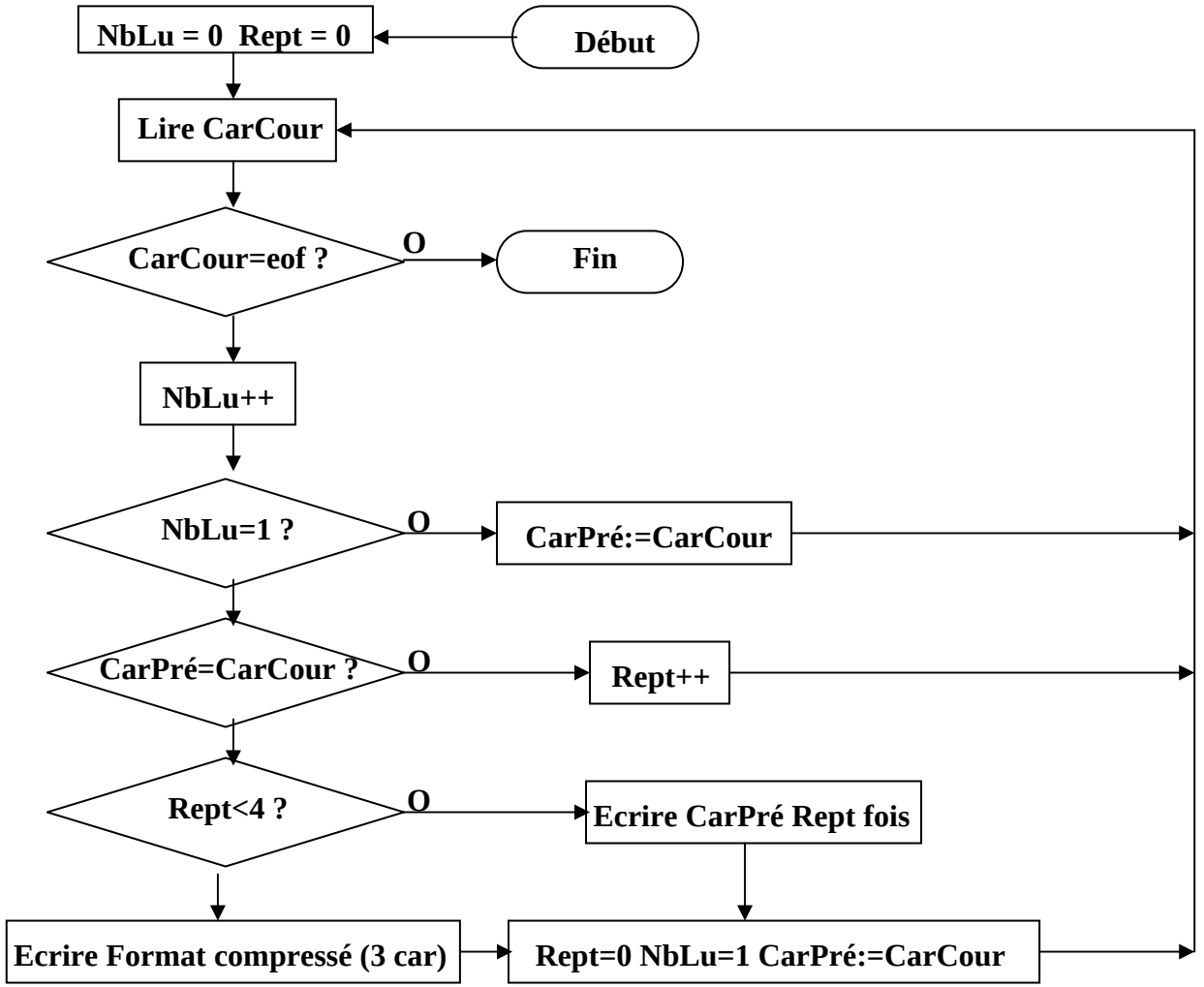
ddddddddddd → *@10d*

Compression pour plus de 3 répétitions

Répétition < 255

Caractère spécial dans le message?

Compression RLE: principe



Compression RLE: propriétés

Suppression du Caractère Spécial

Norme MNP5 *(Microcom Network Protocol class 5: modem)*

Répétition de n octets	***aaan***
c	**c**
cc	**cc**
ccc	**ccc0**
cccc	**ccc1**

Facteur de compression

pour une chaîne de N caractères

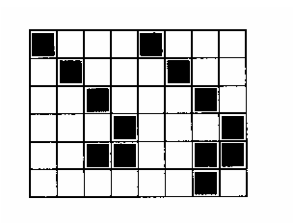
comportant M répétitions de L caractères en moyenne

$$fc = \frac{N}{N - ML + 4M} = \frac{N}{N - M(L-4)} = \frac{Taille\_Entrée}{Taille\_Sortie}$$

Compression RLE pour les Images

Cas des images binaires:

Répétitions Blancs, Répétition Noirs… lignes après lignes
Suppose qu'on débute sur un pixel blanc (0 si absent)
Il faut connaître la taille du bitmap pour le décompresser



0 1 3 1
4 1 3 1
4 1 3 1
4 1 3 1
2 2 2 2
6 1 1

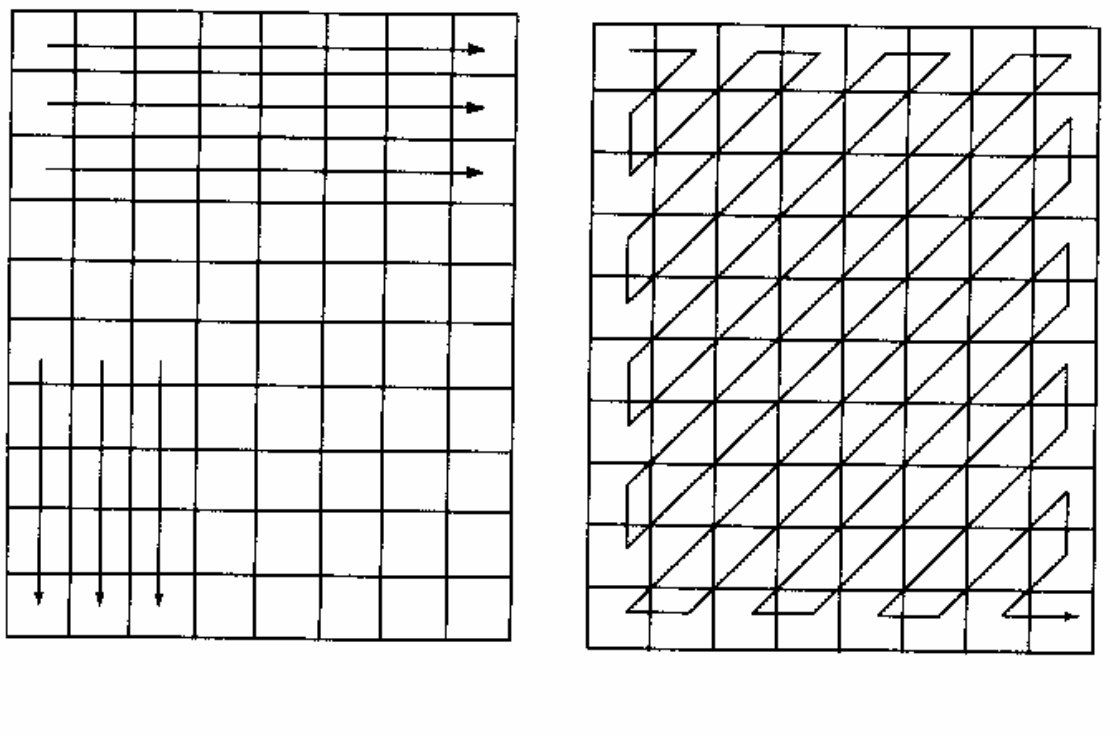
0131413141314131222261 1

Taux de compression pour une région uniforme

$$Tx = \frac{DemiPérimètre}{Surface} = \frac{TailleSortie}{TailleEntrée}$$

Compression RLE pour les Images

Influence du sens de parcours



Tester les 3 et appliquer le meilleur Tx de compression

1. Méthodes Statistiques

Idée Générale

Mettre en œuvre des codes de longueur variable pour réaliser des codes de longueur moyenne minimale.

et réaliser des codes qui ne soient pas ambigüs.

Un vieil exemple : Alphabet Morse 1843

A	**.-**	**N**	**-.**	**1**	**.----**	**Period**	**.-.-.-**
B	**-. . .**	**0**	**---**	**2**	**. .---**	**Comma**	**--. .--**
C	**-. -.**	**P**	**. --.**	**3**	**…--**	**Colon**	**---...**
Ch	**----**	**Q**	**--.-**	**4**	**....-**	**Question mark**	**. . --. .**
D	**-..**	**R**	**.-.**	**5**	**.....**	**Apostrophe**	**.----.**
E	**.**	**S**	**...**	**6**	**-....**	**Hyphen**	**-....-**
F	**..-.**	**T**	**-**	**7**	**--...**	**Dash**	**-..-.**
G	**--.**	**U**	**.--**	**8**	**---..**	**Parentheses**	**-.--.-**
H	**....**	**V**	**... -**	**9**	**----.**	**Quotation marks**	**.-..-.**

Il faut être capable de mesurer le « poids » de chaque symbole dans le message transmis : Mesurer l'Information (Shannon)

Théorie de l'information : C. Shannon 1948-1959

Exemple

On souhaite transmettre une chaîne de L symboles $a^1a^2a^3...a^{L-2}a^{L-1}a^L$

chaque symbole appartient à l'alphabet A
et peut donc être codé sur log2(n) bits $a^i \in A = \{a_1,...,a_n\}$

si la transmission est faite à s symboles par seconde
alors l'information transmise en bit par seconde est $H = s\log_2(n)$

Si les symboles a_i surviennent avec la probabilité P
(équiprobables) $n = \frac{1}{P}$

Alors $H = s\log_2(\frac{1}{P}) = -s\log_2(P)$ en bit/s

Principes de la théorie de l'information (2)

Si les symboles ne sont pas équiprobables, chacun avec une probabilité P_i il y a en moyenne sP_i symbole a_i par seconde

Donc en sommant sur les n symboles de l'alphabet on a

$$H = -\sum_{i=1}^{n} sP_i \log_2(P_i) = -s\sum_{i=1}^{n} P_i \log_2(P_i)$$

c'est la quantité d'information transmise en moyenne en bits par seconde

c'est un débit

L'information contenue dans un symbole est donc H/s soit

$$E = \frac{H}{s} = -\sum_{i=1}^{n} P_i \log_2(P_i)$$ Entropie des données (bits)

$$e = -P_i \log_2(P_i)$$ Entropie du symbole a_i

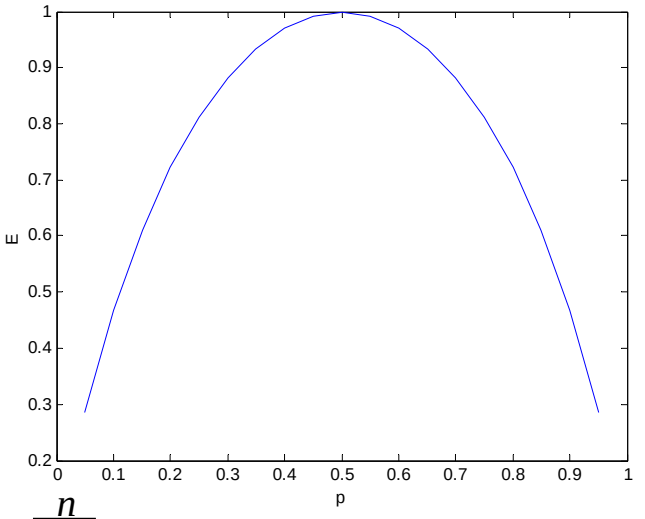
Principes de la théorie de l'information (3)

L'entropie est maximale quand les symboles sont équiprobables

$$0 \leq E \leq \log_2(L)$$

Cas d'une source binaire à deux symboles

a1	a2	entropie (E)
0.5	0.5	1
0.25	0.75	0.81
0.1	0.9	0.47
0	1	0



Redondance du message $R = \log_2(n) - (-\sum_{i=1}^{n} p_i \log_2(p_i))$

Un message parfaitement compressé n'est pas redondant

si $\log_2(n) = -\sum_{i=1}^{n} p_i \log_2(p_i)$ alors $R = 0$

Codes de Taille Variable (1)

Exemple: 4 symboles

	pi	**pi**	**code 0**	**code1**	**code2**
a1	0.25	0.49	00	1	1
a2	0.25	0.25	01	01	01
a3	0.25	0.25	10	010	000
a4	0.25	0.01	11	001	001
E	2bits	1.57bits			
R	0	0.43			

Le code 0 est redondant dans le deuxième exemple: on peut le compresser

Avec le code 1: le nombre de bits moyen par symbole est

1*0.49+2*0.25+3*0.25+3*0.01=1.77 R=1.77-1.57=0.2

20 symboles: a1a3a2a1a3a3a4a2a1a1a2a2a1a1a3a1a1a2a3a101

37 bits: 1,010,01,1,010,010,001,01,1,1,01, 1,1,010,1,1,01,010,1

Code de taille variable (2)

Mais le décodage du code1 est *Ambigu*

	1,0 1 0, 0 1	a1a3a2
ou	1,01,001	a1a2a4

	c1	c2
a1	1	1
a2	01	01
a3	010	000
a4	001	001

Le code 2 est non ambigu au décodage

1 0 1 0 0 1 a1a2a4

⇒ Un code est non ambigu si aucun symbole n'est le préfixe d'un autre

Pour construire un code de taille variable il faut appliquer les 2 principes

1) attribuer les codes les plus courts aux symboles les plus fréquents
2) interdire les symboles préfixes d'un autre

Inégalité de Kraft-MacMillan

Un code de n symboles a_i de taille L_i bits est non ambigu si et seulement si

$$\sum_{i=1}^{n} 2^{-L_i} \leq 1$$

Or la longueur de chaque code est supérieure à son entropie

$L_i = -\log_2(p_i) + E_i$ *Ei* est la longueur supplémentaire pour que L_i soit entier

Alors $2^{-L_i} = 2^{(\log(p_i) - E_i)} = \frac{p_i}{2^{E_i}}$ Si les Ei sont identiques Ei=E

$$\sum_{i=1}^{n} 2^{-L_i} = \frac{\sum_{i=1}^{n} p_i}{2^E} = \frac{1}{2^E} \leq 1$$ si E>=1

Code optimal

Lorsque chaque code est d'une longueur égale à la valeur de son entropie dans le message alors l'erreur est nulle et la longueur moyenne du code est la plus faible possible.

Elle est égale à l'entropie du message

$$\bar{L} = \sum_{i=1}^{n} P_i \, L_i = -\sum_{i=1}^{n} P_i \log_2(P_i) = E$$

Cette valeur ne peut pas être atteinte si on code les symboles individuellement car il faut un nombre entier de bit par symbole dans ce cas

Théorème du codage sans bruit d'une source

On montre que pour toute source discrète sans mémoire il existe un code représentant exactement cette source et non ambigu vérifiant

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

Source sans mémoire:

Les symboles sont émis indépendamment les uns des autres

Codage de Shannon-Fano

On connaît les symboles de l'alphabet et leur fréquence

1- Les classer dans l'ordre décroissant des fréquences

2- Construire deux ensembles de symboles équiprobables

3- les symboles du premier ensemble auront un code débutant par 1
les symboles du second ensemble auront un code débutant par 0

4- Aller en 2 pour chaque sous-ensemble tant qu'il en reste

La taille moyenne des symboles est

0.25*2+0.2*2+0.15*3+0.15*3+0.1*3+0.1*4+0.05*4=2.7 bits/symbole

	Proba.	Etapes			Codage	
1.	0.25	1 1			:11	
2.	0.20	1 0			:10	
3.	0.15	0	1 1		:011	
4.	0.15	0	1 0		:010	
5.	0.10	0	0	1	:001	E=2.67bits
6.	0.10	0	0	0 1	:0001	R=0.03
7.	0.05	0	0	0 0	:0000	

Codage de Shannon-Fano (Suite)

Cas optimale lorsque les sous-ensembles sont équiprobables

	Proba.	Étapes			code
1.	0.25	1	1		:11
2.	0.25	1	0		:10
3.	0.125	0	1	1	:011
4.	0.125	0	1	0	:010
5.	0.125	0	0	1	:001
6.	0.125	0	0	0	:000

Taille moyenne = 2.5 = E (Limite de l'inégalité de Kraft-MacMillan)

Codage d'Huffman

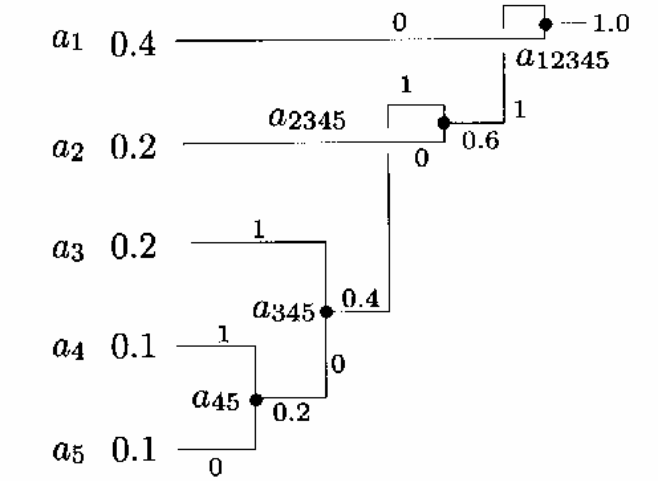
Construction du code dans le sens des bits de poids décroissant

1- Classer les symboles selon leur fréquence

2- Regrouper les deux moins fréquents et les coder
par un même symbole (respecter l'ordre des fréquences)

3- Répéter 2 jusqu'à n'avoir qu'un symbole

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



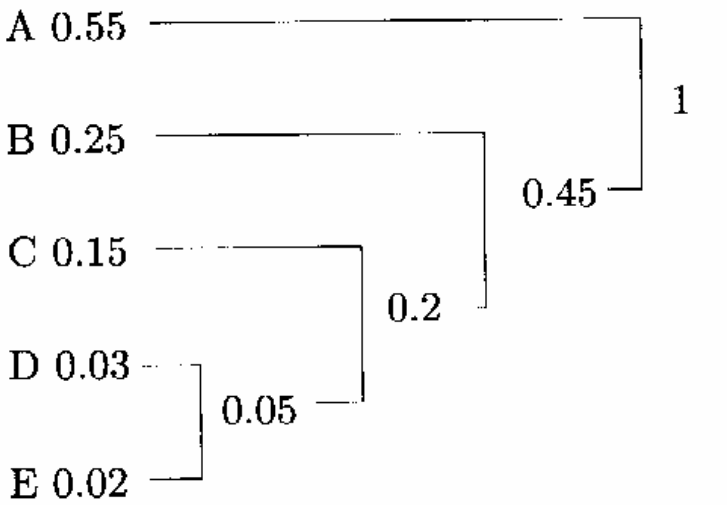
a1= 0
a2=10
a3=111
a4=1101
a5=1100

2.2 bits/symbole

Arbre d'Huffman: Le poids de chaque nœud est la somme des poids des fils

Codage d'Huffman (2)

Taille moyenne du code



(a)

Calcul directe

1*0.55+2*0.25+3*0.15+4*(0.02+0.03)=1.7 bits

Propriété d'un arbre d'Huffman

La somme des valeurs des feuilles pondérées par leur distance à la racine est égale à la somme des valeurs des nœuds: 1+0.45+0.2+0.05=1.7

Décodage

Il faut reconstruire l'arbre d'Huffman

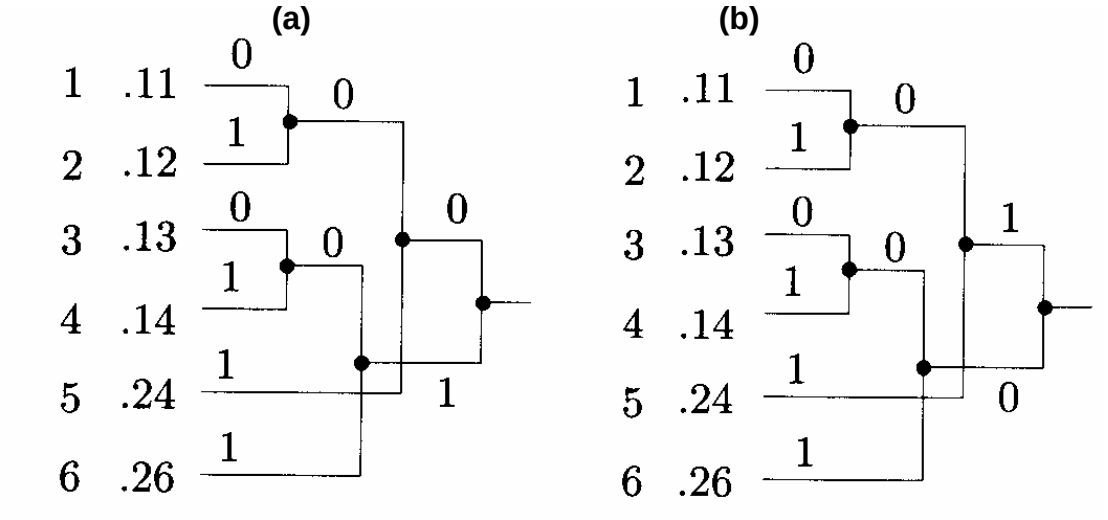
à partir des probabilités, fréquences

Décodage

parcourir l'arbre à partir de la racine pour chaque bit lu lorsqu'on atteint une feuille on trouve le code ASCII du symbole, puis on reprend à partir de la racine

Codage d'Huffman (3)

Différents codes



1: 000	100	000
2: 001	101	001
3: 100	000	010
4: 101	001	011
5: 01	11	10
6: 11	01	11
(a)	(b)	(c)

Codes Canoniques (c) : DECODAGE SANS ARBRE

- Les codes de n bits doivent avoir des valeurs supérieures aux préfixes des codes de longueur supérieure
- les préfixes de n bits des codes de longueur > n doivent être inférieurs aux valeurs des code de n bits

1: 000	100	**000**
2: 001	101	**001**
3: 100	000	**010**
4: 101	001	**011**
5: 01	11	**10**
6: 11	01	**11**
(a)	(b)	(c)

Exemple pour un code de 3 bits au plus

Les codes de 3 bits varient de 0 à 3 soit **000 001 010 011**
Les codes de 2 bits sont alors **10 11**
Aucun code de 1 bits

Longueur(bits)	1	2	3
Nb codes	0	2	4
première valeur	2	2	0

Codes Canoniques (5)

Règle de construction d'un code canonique (arbre d'Huffman est contruit)

nb[l]: nombre de code pour l bits

Initialisation

l = lmax: longueur maximale du code

PremVal[l]=0

Pour l=lmax-1 **jusqu'à** 1 par pas de –1

PremVal[l]=Arondi((PremVal[l+1]+nb[l+1])/2);

Exemple pour 6 bits: lmax=6, nb[lmax]=max+1=7, PremVal[lmax]=0

Le nombre de code nb par longueur doit être connu (construction de l'arbre)

l	1	2	3	4	5	6
nb	0	0	4	0	5	7
PremVal	2	4	3	5	4	0

Décodage (6)

Décodage pour un code canonique

```
l=1, lire c
tant que c < PremVal[l]
        Concaténer le bit suivant avec c, l=l+1,
fin
symbole = c – PremVal[l] /* parmi les symboles de l bits
```

Permet de travailler vite avec des alphabet de grande taille

Codage d'une source sans bruit avec mémoire

Idée générale

Souvent les symboles ne surviennent pas indépendamment les uns des autres dans le message qu'on veut coder

On peut s'approcher de la borne entropique en cherchant à coder ensemble plusieurs symboles. On parvient alors à attribuer en moyenne un nombre non entier de bit par symbole

Pour cela il faut déterminer l'information apportée par une séquence de symboles

Codage d'une source sans bruit avec mémoire (2)

Pour une séquence de deux symboles : XY

on introduit la probabilité jointe d'observer ensemble les deux symboles

$$P_{XY}(i,j)=P(X=a_i,Y=a_j)$$

et la probabilité conditionnelle d'observer un certain symbole après un autre

$$P_{Y|X}(i|j)=P(Y=a_i|X=a_j)$$

On définit alors l'entropie jointe des deux symboles consécutifs par:

$$E(X,Y)=-\sum_{i=1}^{n}\sum_{j=1}^{n}P_{XY}(i,j)\log 2(P_{XY}(i,j|))$$

C'est l'information apportée en moyenne par un couple de symboles consécutifs

Codage d'une source sans bruit avec mémoire (3)

définit alors l'entropie conditionnelle de Y quand X

$$E(Y|X) = -\sum_{i=1}^{n}\sum_{j=1}^{n} P_{XY}(i,j)\log 2(P_{Y|X}(i|j))$$

C'est l'information apporté en moyenne par le second symbole quand on connaît le premier

On montre que

$$E(X,Y) = E(X) + E(Y|X)$$

L'information apportée en moyenne par un couple de symbole est l'information apportée en moyenne par le premier plus celle apporté en moyenne par le second quand on connaît le premier

Codage d'une source sans bruit avec mémoire (4)

on généralise à des séquences de N symboles

$$E(X_N) = -\sum_{i=1}^{n} P_{Xn}(i_1, i_2 \ldots i_{N-1}, i_N) \log_2\left(P_{Xn}(i_1, i_2 \ldots i_{N-1}, i_N)\right)$$

Débit entropique d'une source

C'est l'information moyenne apportée par une séquence de longueur tendant vers l'infini

$$\bar{E} = \lim_{N \to \infty} \frac{1}{N} E(X_N)$$

On montre qu'on peut coder une source sans ambiguïté pourvu que le nombre de bit moyen par symbole soit supérieur ou égal au débit entropique

…On peut donc faire mieux en tenant compte de la mémoire de la source

Compression des FAX

Norme de l'International Telecommunication Union (ITU)

depuis 1993 CCITT:

Comité Consultatif International Télégraphique et Téléphonique

Norme T4 (Groupe3): Réseaux commutés standards: 9600 baud

Norme T6 (Groupe 4): Réseaux numériques 64Kbits

Groupe 3: codage mono-dimensionnel

Résolution Horizontale

Scan à 8.05 points par millimètres largeurs de l'image: 1664 pixels

Résolution Verticale

3.85 lignes par mm 7.7 (fine mode) 15.4 (very fine mode)

Nbr de Lignes	Pixels/ligne	Pixels/page	Temps
978	1664	1.67 M	170
1956	1664	3.255 M	339
3912	1664	6.51 M	678

Compression Groupe 3 (1)

Compression: Évaluation des Run-Length sur 8 documents Différents

les plus fréquents: 2,3,4 pixels noirs ⟹ les codes les pus courts

2 à 7 pixels blancs

RLE + Huffman

Run Length = 1 à 63 pixels (codes terminaux) +

+ multiples de 64 (codes de mise en page= « make-up codes »

Exemple: 12 pixels blancs = 001000

76 pixels blancs = 11011,001000 = 128+12

64 pixels noirs = 0000001111,0000110111 = 64+0

2561 pixels noirs = 000000011111,010 = 2560+1

Chaque ligne est codée séparément et se termine par EOL= 000000000001

1 pixel blanc est placé systématiquement en début de ligne et éliminé au décodage

Chaque page comporte 1 caractère EOL en début et 6 EOL en fin

Codes RLE Compression Groupe 3 et 4 (2)

Codes terminaux

Run length	White code-word	Black code-word	Run length	White code-word	Black code-word
0	00110101	0000110111	32	00011011	000001101010
1	000111	010	33	00010010	000001101011
2	0111	11	34	00010011	000011010010
3	1000	10	35	00010100	000011010011
4	1011	011	36	00010101	000011010100
5	1100	0011	37	00010110	000011010101
6	1110	0010	38	00010111	000011010110
7	1111	00011	39	00101000	000011010111
8	10011	000101	40	00101001	000001101100
9	10100	000100	41	00101010	000001101101
10	00111	0000100	42	00101011	000011011010
11	01000	0000101	43	00101100	000011011011
12	001000	0000111	44	00101101	000001010100
13	000011	00000100	45	00000100	000001010101
14	110100	00000111	46	00000101	000001010110
15	110101	000011000	47	00001010	000001010111
16	101010	0000010111	48	00001011	000001100100
17	101011	0000011000	49	01010010	000001100101
18	0100111	0000001000	50	01010011	000001010010
19	0001100	00001100111	51	01010100	000001010011
20	0001000	00001101000	52	01010101	000000100100
21	0010111	00001101100	53	00100100	000000110111
22	0000011	00000110111	54	00100101	000000111000
23	0000100	00000101000	55	01011000	000000100111
24	0101000	00000010111	56	01011001	000000101000
25	0101011	00000011000	57	01011010	000001011000
26	0010011	000011001010	58	01011011	000001011001
27	0100100	000011001011	59	01001010	000000101011
28	0011000	000011001100	60	01001011	000000101100
29	00000010	000011001101	61	00110010	000001011010
30	00000011	000001101000	62	00110011	000001100110
31	00011010	000001101001	63	00110100	000001100111

Codes de mise en page

Run length	White code-word	Black code-word	Run length	White code-word	Black code-word
64	11011	0000001111	1344	011011010	0000001010011
128	10010	000011001000	1408	011011011	0000001010100
192	010111	000011001001	1472	010011000	0000001010101
256	0110111	000001011011	1536	010011001	0000001011010
320	00110110	000000110011	1600	010011010	0000001011011
384	00110111	000000110100	1664	011000	0000001100100
448	01100100	000000110101	1728	010011011	0000001100101
512	01100101	0000001101100	1792	00000001000	same as
576	01101000	0000001101101	1856	00000001100	white
640	01100111	0000001001010	1920	00000001101	from this
704	011001100	0000001001011	1984	000000010010	point
768	011001101	0000001001100	2048	000000010011	
832	011010010	0000001001101	2112	000000010100	
896	011010011	0000001110010	2176	000000010101	
960	011010100	0000001110011	2240	000000010110	
1024	011010101	0000001110100	2304	000000010111	
1088	011010110	0000001110101	2368	000000011100	
1152	011010111	0000001110110	2432	000000011101	
1216	011011000	0000001110111	2496	000000011110	
1280	011011001	0000001010010	2560	000000011111	

Codage bi-dimensionnel Groupe 3 et 4 (1)

Groupe 3 optionnel, Groupe 4 toujours

Meilleur pour Images en Niveaux de Gris

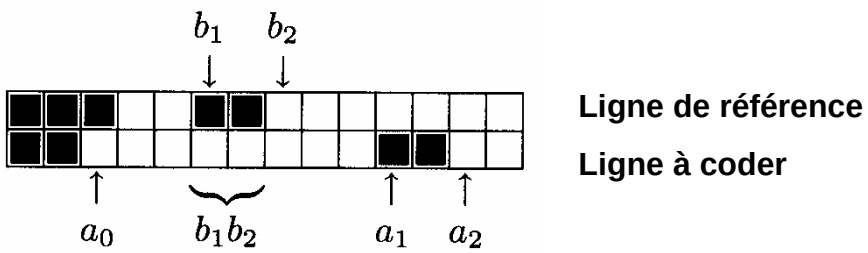
Codage relatif de chaque ligne par rapport à la précédente

La première ligne du document est supposée blanche
Chaque ligne commence par un pixel blanc

Moins robuste que mono-dimensionnel par ligne
G3: Codage mono-dimensionnel toutes les 2 ou 4 lignes
G4: Bi-dimensionnel sur le document complet

Codage bi-dimensionnel Groupe 3 et 4 (2)

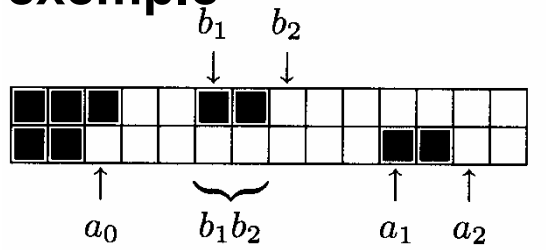
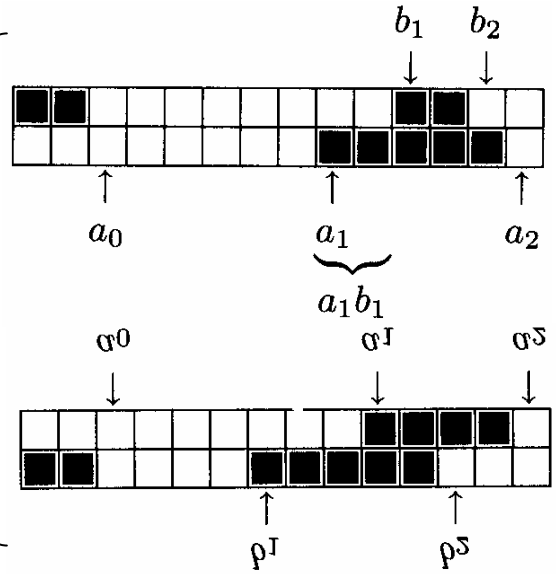
Principe:
a0a1 est le run courant à coder et le suivant est a1a2



On code en fonction de b1b2 sur le ligne de référence
b1 1er pixel à droite de a0 de couleur différente

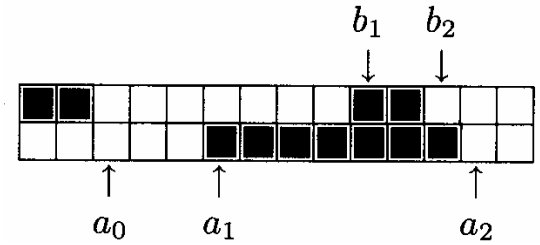
Mode Pass: Quand b_1b_2 à gauche de a_1a_2

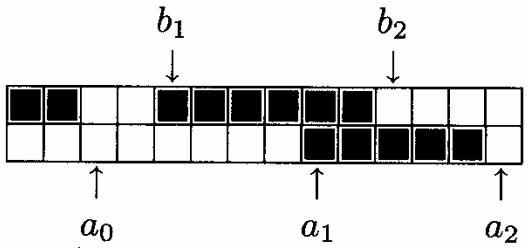
Codage bi-dimensionnel Groupe 3 et 4 (2)

Mode	exemple	condition	élément codé
Mode pass		pas de recouvrement de b1b2 sur a1a2 b2 à gauche de a1	Coder le run b1b2 puis a0=b2.
Mode Vertical		Recouvrement de moins de 3 pixels de b1b2 sur a1a2 a1b1 = -3,-2,-1,0,1,2,3	Coder le run a1b1 puis a0=a1
		2	**000010**
		-3	**0000011**

Codage bi-dimensionnel Groupe 3 et 4 (3)

Mode horizontal





b1b2 recouvre a1a2 de plus de 3 pixels
a1b1 > 3 pixels

Coder les run a0a1 et a1a2 puis déplacer a0 en a2

3 7

6 5

Codage bi-dimensionnel Groupe 3 et 4 (4)

Mode	Run length to be encoded	Abbre-viation	Codeword
Pass	b_1b_2	P	0001+coded length of b_1b_2
Horizontal	a_0a_1, a_1a_2	H	001+coded length of a_0a_1 and a_1a_2
Vertical	$a_1b_1 = 0$	V(0)	1
	$a_1b_1 = -1$	VR(1)	011
	$a_1b_1 = -2$	VR(2)	000011
	$a_1b_1 = -3$	VR(3)	0000011
	$a_1b_1 = +1$	VL(1)	010
	$a_1b_1 = +2$	VL(2)	000010
	$a_1b_1 = +3$	VL(3)	0000010
Extension			0000001000

Initialisation

a0 = pixel blanc à gauche de l'image

a1 = premier point noir

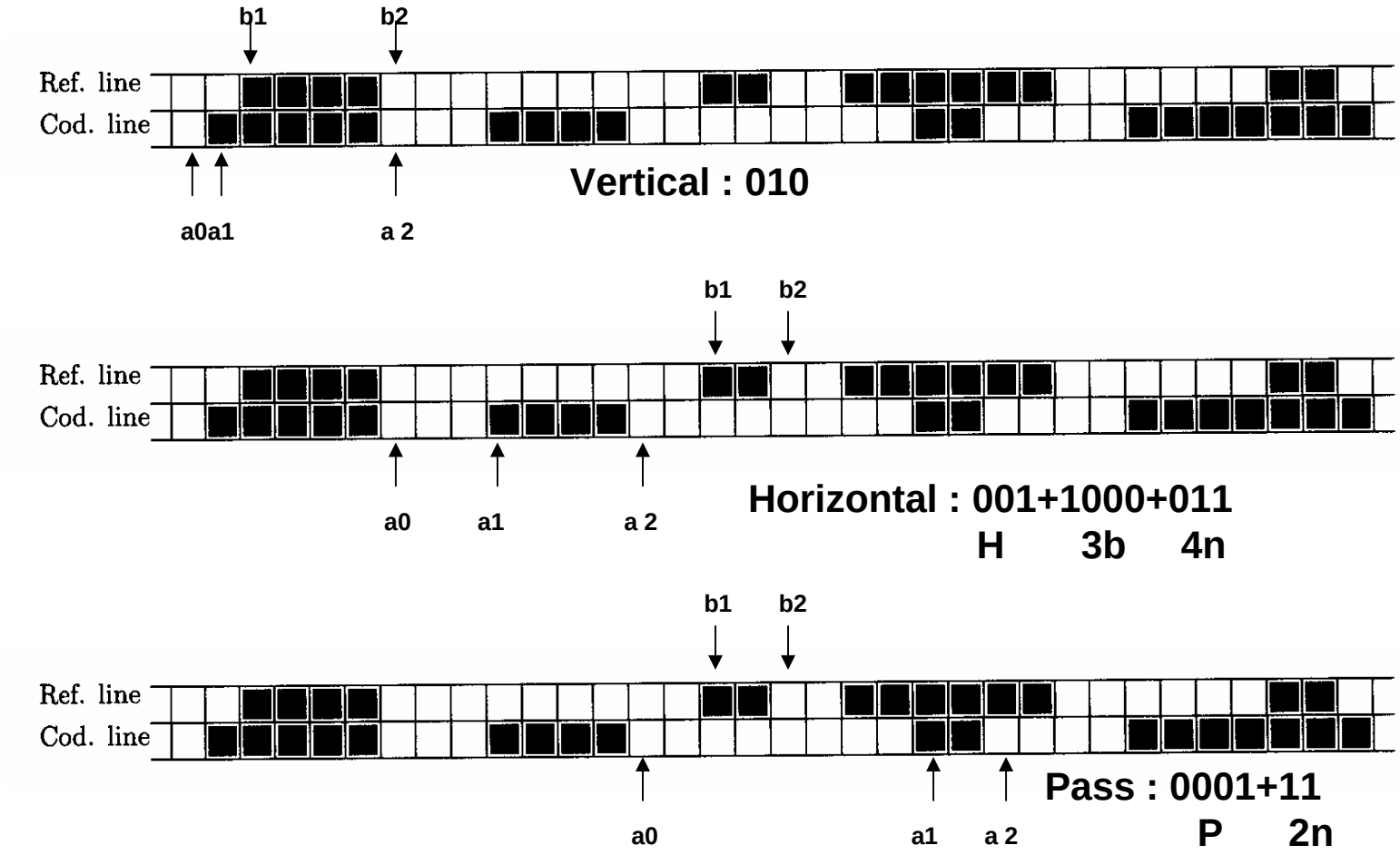
a2 premier blanc suivant

et abs(a0a1)=-1

b1 premier pixel à droite de a0 de couleur différente

b2 premier pixel à droite de b1 et de couleur différente

Codage bi-dimensionnel Groupe 3 et 4 (5)



2. Méthodes de compression avec mémoire

Les méthodes statistiques que nous venons de voir ne permettent pas d'atteindre la borne entropique car elles codent les symboles les uns indépendamment des autres.

Nous allons voir maintenant deux méthodes qui codent des séquences de symboles, voire la séquence dans son intégralité.

Codage Arithmétique

Méthodes précédentes efficaces pour des symboles de probabilité $p=2^{-n}$

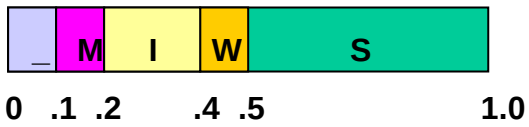
Pour un symbole de proba 0.4 Il faudrait pouvoir lui attribuer $-\log_2(0.4)=1.32$bit exactement

On code une séquence de symboles dans sa globalité pas les symboles individuellement

Exemple: codage de la suite de caractères **SWISS MISS**

	Freq	Prob.	Intervalle
S	5	5/10	[0.5 1[
W	1	1/10	[0.4 0.5[
I	2	2/10	[0.2 0.4[
M	1	1/10	[0.1 0.2[
_	1	1/10	[0.0 0.1[

[0 1[est fractionné en sous-intervalles



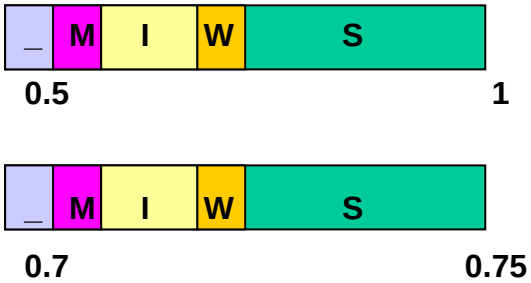
Codage Arithmétique (2)

On définit Low(0) = 0 High(0) = 1 Range(0) = 1

S est dans [0.5 1[**donc** **Low(1)** = 0.5

High(1) = 1

Range(1) = 0.5



W [0.4 0.5[**donc** Low(2) = Low(1) + Range(1) * **Inf(W)** = 0.7

High(2) = Low(1) + Range(1) * **Sup(W)** = 0.75

Range(2) = Range(1) * **Echelle(W)** = Range(1) * (Sup(W)-Inf(W)) = High(2) – Low(2) = 0.05 = Echell(W) * Echelle(S)

I [0.2 0.4[**donc** Low(3) = Low(2) + Range(2) * Inf(I)=0.71

High(3) = Low(2) + Range(2) * Sup(I)=0.72

Range(3) = 0.01=Echelle(I)*Echelle(W)*Echelle(S)

…….

S [0.5 1[**donc** Low(n) = Low(n-1) + Range(n-1) * Inf(S)= 0.71753375

High(n) = Low(n-1) + Range(n-1) * Sup(S)=0.717535

On conserve **Low(n) = 0.71753375** on code **71753375**

Codage Arithmétique (3)

Le code final correspond au schémas de factorisation suivant :

Low(n) = Low(n-1) + Range(n-1) * Inf(n)

= Low(n-2) + Range(n-2) * Inf(n-1) + Range(n-1) * Inf(n)

= Low(n-2) + Range(n-2) * (Inf(n-1) + Echelle(n-1) * Inf(n))

……..

= Low(1) + Echelle(1) * (Inf(2) + Echelle(2) * (Inf(3) + Echelle(3)* (…..)))

Low(n) = Low(1) + Echelle(1) * Code(2) < High(1)

Où code(2) ne dépend que des caractères à partir du second dans la chaîne

On déduit de même

Code(2) = Inf(2) + Echelle(2) * (Inf(3) + Echelle(3)* (…..))

= Inf(2) + Echelle(2) * Code(3)

Codage Arithmétique (3)

Low(n) = Low(1) + Echelle(1) * Code(2) < High(1)

Code(2) = Inf(2) + Echelle(2) * (Inf(3) + Echelle(3)* (…..))
= Inf(2) + Echelle(2) * Code(3) < High(2)

Donc au décodage il suffit d'effectuer les opérations inverses

Déduire caractère 1 à partir de Low(n)

Code(2) = (Low(n) – Low(1)) / Echelle(1)

Déduire caractère 2 à partir de code(2)

Code(3) = (Code(2) – low(2)) / Echelle(2)

……..

Codage Arithmétique (4)

Décodage

On doit connaître la table des symboles et leurs plages (début du fichier)

S [0.5 1[W [0.4 0.5[I [0.2 0.4[M [0.1 0.2[_ [0.0 0.1[

Soit à décoder 71753375

1- le code est dans l'intervalle [0.5 1[donc premier symbole S

2- déterminer le nouveau code: (0.71753375-0.5)/0.5=0.4350675

new_code = (code - Low(S)) / range(S)

3- le code est dans l'intervalle [0.4 0.5[donc second symbole W

4-nouveau code (0.4350675-0.4)/0.1=0.350675

5- **I**, nouveau code (0.350675-0.2)/0.2=0.753375
6- **S**, code (0.753375-0.5)/0.5=0.50675
7- **S**, code (0.50675-0.5)/0.5=0.0135
8- **_**, code (0.0135-0.0)/0.1=0.135
9- **M**, code (0.135-0.1)/0.1=0.35
10- Symbole **I**, code (0.35-0.2)/0.2=0.75
11- Symbole **S**, code (0.75-0.5)/0.5=0.5
12- Symbole **S**, code 0

Codage Arithmétique (5)

Problème

pour des fichiers de taille importante, le code résultat peut être très long, et les divisions et soustractions impossibles

Solution

Représenter les limites des intervalles High et Low par des entiers

Quand les digits les plus significatifs de High et Low deviennent identiques, ils ne changent plus ensuite.
Il est alors inutile de les conserver, on les élimine en les écrivant dans le fichier compressé, on décale d'un digit à gauche Low et High et on ajoute à droite de Low un zéro à droite de High 9

Low= xxxxx0…. High=yyyyy9…..

car low=0 high=0.99999999999

Codage Arithmétique (6)

Exemple: Codage de SWISS_MISS

1	2	3	4	5
S	$L = 0+(1-0)\times 0.5 = 0.5$	5000		5000
	$H = 0+(1-0)\times 1.0 = 1.0$	9999		9999
W	$L = 0.5+(1-.5)\times 0.4 = 0.7$	7000	7	0000
	$H = 0.5+(1-.5)\times 0.5 = 0.75$	7499	7	4999
I	$L = 0+(0.5-0)\times 0.2 = 0.1$	1000	1	0000
	$H = 0+(0.5-0)\times 0.4 = 0.2$	1999	1	9999
S	$L = 0+(1-0)\times 0.5 = 0.5$	5000		5000
	$H = 0+(1-0)\times 1.0 = 1.0$	9999		9999
S	$L = 0.5+(1-0.5)\times 0.5 = 0.75$	7500		7500
	$H = 0.5+(1-0.5)\times 1.0 = 1.0$	9999		9999
␣	$L = .75+(1-.75)\times 0.0 = 0.75$	7500	7	5000
	$H = .75+(1-.75)\times 0.1 = .775$	7749	7	7499
M	$L = 0.5+(.75-.5)\times 0.1 = .525$	5250	5	2500
	$H = 0.5+(.75-.5)\times 0.2 = 0.55$	5499	5	4999
I	$L = .25+(.5-.25)\times 0.2 = 0.3$	3000	3	0000
	$H = .25+(.5-.25)\times 0.4 = .35$	3499	3	4999
S	$L = 0.0+(0.5-0)\times 0.5 = .25$	2500		2500
	$H = 0.0+(0.5-0)\times 1.0 = 0.5$	4999		4999
S	$L = .25+(.5-.25)\times 0.5 = .375$	3750	3750	
	$H = .25+(.5-.25)\times 1.0 = 0.5$	4999		4999

71753 375

Initialiser Low=0000 High=9999
NewLow=Low+Range*Low(Symbol)
NewHigh=low+Range*High(Symbol)

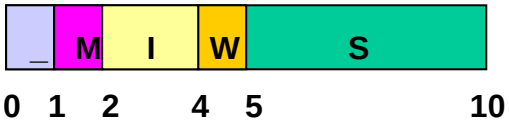
Codage Arithmétique (7)

Décodage de 71753375 (exemple sur 16 bits)

Low = 0000 High = 9999 code = 7175

utiliser les fréquences (entières) à la place des proba

NbSymb



1- index = round(((Code – Low+1)*NbSymb-1)/(High-Low+1)) 7.1759 7

2- Déterminer le symbole S à l'aide de l'index

3- Mettre à jour Low et High

Low=Low+(High-Low+1)*LowCumFreq(Symbol)/NbSymb

High=Low+(High-Low+1)*HighCumFreq(Symbol)/10 – 1

4- Si les digits les plus à gauche de High et Low sont identiques alors

décaler à gauche High, Low, code

ajouter 0 à droite de Low, 9 à gauche de High, le digit suivant à droite de code

Codage Arithmétique (8)

Décodage

0. Initialiser Low=0000, High=9999, and Code=7175.
1. index= [(7175 - 0 + 1) x 10 - 1]/(9999 - 0 + 1) = 7.1759 **conserver** 7 **sélectionner** S
   Low =0+(9999-0+l)x 5/10 =5000. High = 0+(9999-0+1) x 10/10-1 = 9999.
2. index= [(7175 - 5000 + 1) x 10 - 1]/(9999 - 5000 + 1) = 4.3518 **conserver** 4 **sélectionner** "W"
   Low = 5000 + (9999 - 5000 + 1) x 4/10 = 7000. High = 5000 + (9999 - 5000 + 1) x 5/10 - 1 = 7499.
   éliminer 7 puis affecter Low=0000, High=4999, Code=1753.
3. index= [(1753 - 0 + 1) x 10 - I]/(4999 - 0 + 1) = 3.5078 **conserver** 3 **sélectionner** I
   Low = 0+(4999-0+l)x 2/10= 1000. High = 0+(4999-0+1) x 4/10-1 = 1999.
   éliminer 1 puis affecter Low=0000, High=9999, and Code=7533.
4. index= [(7533 - 0 + 1) x 10 - 1]/(9999 - 0 + 1) = 7.5339 **conserver** 7 **sélectionner** S
   Low =0+(9999-0+l)x 5/10 =5000. High = 0+(9999-0+1) x 10/10-1 = 9999.
5. index= [(7533 - 5000 + 1) x 10 - 1]/(9999 - 5000 + 1) = 5.0678 **conserver** 5 **sélectionner** S
   Low = 5000 + (9999 - 5000 + 1) x 5/10 = 7500. High = 5000 + (9999 - 5000 + 1) x 10/10 - 1 = 9999.
6. index= [(7533 - 7500 + 1) x 10 - 1]/(9999 - 7500 + 1) = 0.1356 **conserver** O **sélectionner** u
   Low = 7500 + (9999 - 7500 + 1) x 0/10 = 7500. High = 7500 + (9999 - 7500 + 1) x 1/10 - 1 = 7749.
   éliminer 7 affecter Low=5000, High=7499, Code=5337.
7. index= [(5337 - 5000 + 1) x 10 - 1]/(7499 - 5000 + 1) = 1.3516 **conserver** 1 **sélectionner** M
   Low = 5000 4- (7499 - 5000 + 1) x 1/10 = 5250. High = 5000 + (7499 - 5000 + 1) x 2/10 - 1 = 5499.
   éliminer 5 affecter Low=2500, High=4999, Code=3375.
8. index= [(3375 - 2500 + 1) x 10 - 1]/(4999 - 2500 + 1) = 3.5036 **conserver** 3 **sélectionner** I
   Low = 2500 + (4999 - 2500 + 1) x 2/10 = 3000. High = 2500 + (4999 - 2500 + 1) x 4/10 - 1 = 3499.
   éliminer 3 affecter Low=0000, High=4999, Code=3750.
9. index= [(3750 - 0 + 1) x 10 - 1]/(4999 - 0 + 1) = 7.5018 **conserver** 7 **sélectionner** S
   Low =0+(4999-0+l)x 5/10 =2500. High = 0+(4999-0+1) x 10/10-1 =4999.
10. index= [(3750 - 2500 + 1) x 10 - 1]/(4999 - 2500 + 1) = 5.0036 **conserver** 5 **sélectionner** S
    Low = 2500 + (4999 - 2500 + 1) x 5/10 = 3750. High = 2500 + (4999 - 2500 + 1) x 10/10 - 1 = 4999.

Codage Arithmétique (9)

Analyse de la compression

Le codage en décimal peut être facilement transformé en un codage binaire
Il suffit de placer 1 à droite de High au lieu de 9

10 caractères initiaux SWISS_MISS **71753375**

Si on travaillait en binaire

On coderait low et high en binaire
soit low= 0.71753375 = 0.10110111101100000100101010111
high= 0.717535 = 0.10110111101100000101111111011

en appliquant le principe du codage arithmétique on conserverait le préfixe commun
en binaire 10110110111101100000100 soit 20 bits

L'entropie de la chaîne est 19,6 bits donc la compression obtenue est très bonne
Variante utilisée dans JPEG2000

Méthodes à base de Dictionnaire

Principe

Sauvegarder des chaînes de caractères dans un dictionnaire et les indexer

En phase de compression:

si la chaîne existe dans le dictionnaire écrire son index
sinon l'écrire dans le fichier

Il faut distinguer les index et les chaînes
avec 19 bits on index 2^{19}=524288 mots différents
le 20eme bit peut servir à distinguer un index et un mot

exemple:
si le mot « gel » est dans le dictionnaire avec pour index 1025
il sera codé sous la forme compressée 0 00000000010000000001
si le mot « xet » n'est pas dans le dictionnaire
il sera codé sous la forme 1 0000011 01111000 01100101 01110100
les 7 derniers bit du premier octet code la taille du mot qui suit en octets

Si on suppose que la taille moyenne d'un mot est de 5 caractères
il faut 6 octets pour le coder en non compressé, 20 bits pour le compresser

Méthodes à base de Dictionnaire (2)

Quelles conditions pour une compression?

- Le mot doit être présent dans le dictionnaire

- Soit P la probabilité de trouver le mot dans le dictionnaire

- Après avoir traité N mots, la taille du fichier compressé est

N*[P*20(bits) + (1-P)*48(bits)]

- Si on suppose des mots de 5 caractères,
la taille du fichier non compressé est de N*5*8 = 40*N bits

Il y a compression si
N*[48-28*P] < 40*N soit si P > 0.29

au moins 30% des mots du fichier doivent être présents dans le dictionnaire

Méthodes à base de Dictionnaire (3)

Nature du Dictionnaire

- doit dépendre des applications: Textes, binaires, codes source…

- peut être construit de façon adaptative au cours de la compression

ajout de mots au fur et à mesure

Jacob Ziv and Abraham Lempel (1970) compression LZ77 et LZ78

Méthodes à base de Dictionnaire (4)

Compression de chaînes plus efficace qu'une compression par symboles

exemple: **soit deux symboles a1 et a2 avec P(a1)=0.8 P(a2)=0.2**

Si on codait les symboles individuellement on aurait besoin de 1 bit / symbole

Si on code les séquences de 2 symboles, un codage d'Huffman donne

Chaînes	Proba.	Code
a_1a_1	$0.8 \times 0.8 = 0.64$	0
a_1a_2	$0.8 \times 0.2 = 0.16$	11
a_2a_1	$0.2 \times 0.8 = 0.16$	100
a_2a_2	$0.2 \times 0.2 = 0.04$	101

La taille moyenne d'une chaîne est 1*0.64+2*0.16+3*0.16+3*0.04=1.56 bits

soit 0.78 bits / symbole

Si on code de la même façon les chaînes de 3 symboles

Chaînes	Proba.	Code
$a_1a_1a_1$	$0.8 \times 0.8 \times 0.8 = 0.512$	0
$a_1a_1a_2$	$0.8 \times 0.8 \times 0.2 = 0.128$	100
$a_1a_2a_1$	$0.8 \times 0.2 \times 0.8 = 0.128$	101
$a_1a_2a_2$	$0.8 \times 0.2 \times 0.2 = 0.032$	11100
$a_2a_1a_1$	$0.2 \times 0.8 \times 0.8 = 0.128$	110
$a_2a_1a_2$	$0.2 \times 0.8 \times 0.2 = 0.032$	11101
$a_2a_2a_1$	$0.2 \times 0.2 \times 0.8 = 0.032$	11110
$a_2a_2a_2$	$0.2 \times 0.2 \times 0.2 = 0.008$	11111

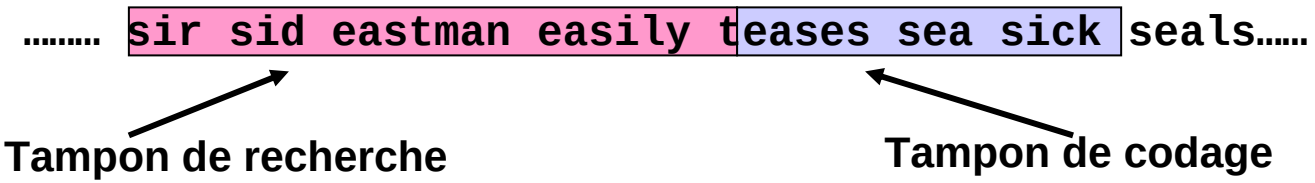
La taille moyenne d'une chaîne est alors 2.184 bits

soit 0.728 bits / symbole

Compression LZ77 (LZ1)

Le dictionnaire contient les chaînes les plus récentes

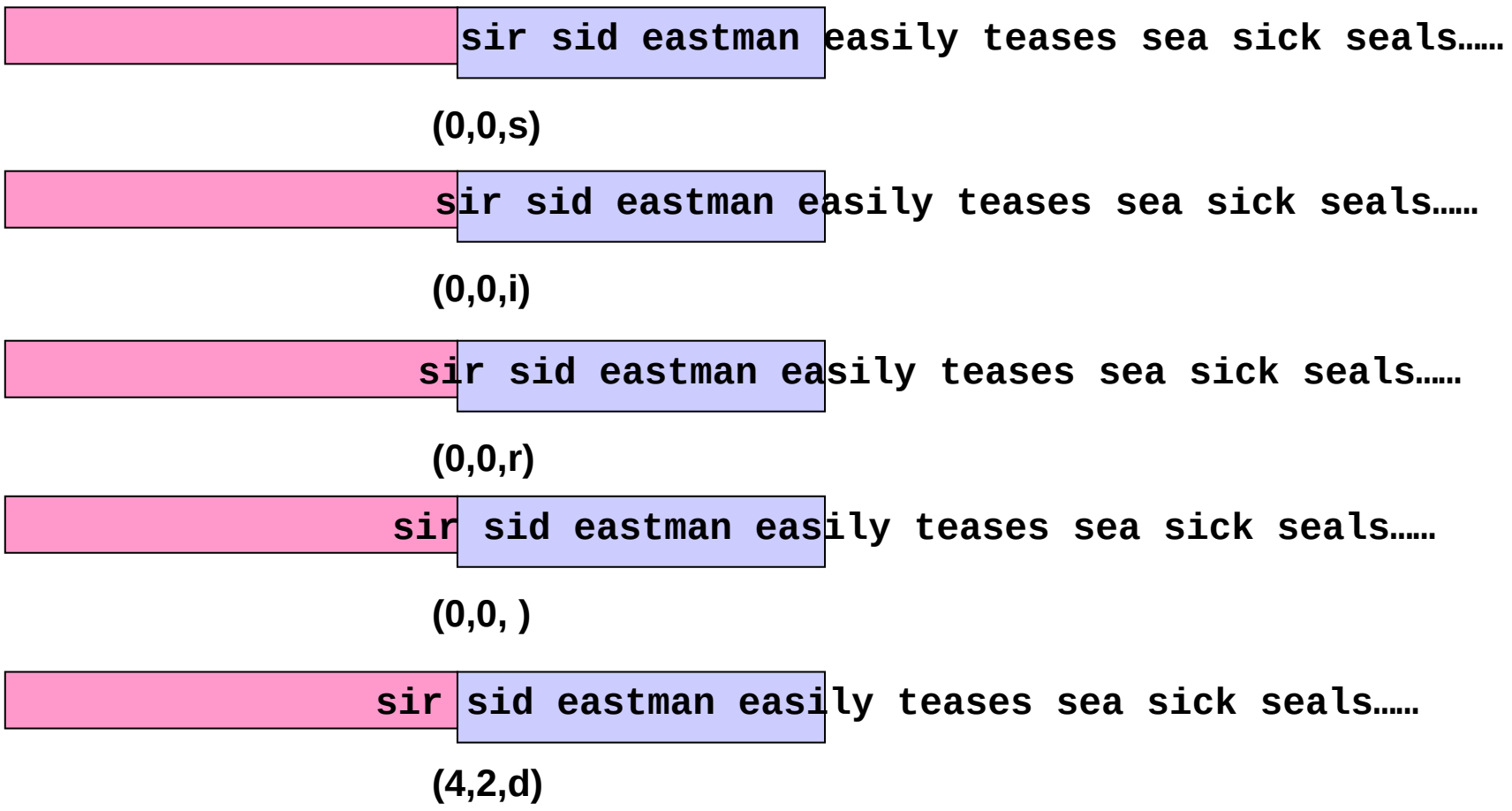
Celles présentes dans une fenêtre glissante



-Le premier caractère du tampon de codage est « e »
-On cherche de droite à gauche dans le tampon de recherche la présence d'un « e »
-Le premier est distant de 8 caractères
-On progresse alors de gauche à droite simultanément dans les deux buffers tant que les caractères sont identiques: on trouve 3 symboles identiques « eas »
-On reprend le déplacement vers la gauche dans le buffer de recherche pour examiner les autres match possibles: on trouve « eas » à une disance de 16 caractères
-On conserve le match le plus long (le plus récent s'ils sont identiques)
-Si aucun match on code le caractère

On code (offset, longueur, symbole suivant le match)

Compression LZ77 (2)



Compression LZ77 (3)

Propriétés

si S désigne la taille du buffer de recherche il faut log2(S) bit pour coder l'offset

entre 10 et 12 bits soit un buffer de 1024 à 4096 caractères

La longueur maximale de la chaîne est égale à la taille du buffer de codage -1

il faut Log2(L-1) bits pour la coder: 5 bits en général

Pour coder le symbole il faut 8 bits en général

Codage total du triplet: 11 + 5 + 8 = 24 bits

Longueur maximale de la chaîne

alf eastman easily yels A|**AAAAAAAAAAA**

taille du buffer 10 **codage (1,9,A)**

alf eastman easily grows alf|**alfa in his garden**

codage (3,4,) et non pas (28,3,a)

Compression LZ77 (4)

Décompression

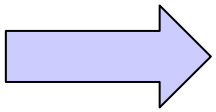
On gère un buffer de même taille que le buffer de recherche

On lit un triplet (offset,longueur,symbole)

On trouve dans le buffer la suite de symboles correspondante

On l'écrit dans le fichier décompressé à laquelle on ajoute le symbole

On écrit également ce résultat dans le buffer



Rapidité de la décompression.

Compression LZ77 (5)

Variantes et Améliorations

- fenêtre mise en œuvre par un barillet
- gérer le buffer de recherche par un arbre pour accélérer la recherche quand la taille augmente
- utiliser des offset et longueurs de taille variable

Remarques

- les fichiers où les chaînes se répètent de façon espacée seront mal compressées
- si on augmenta la taille du buffer de recherche on peut avoir une meilleure compression mais plus lente

Compression LZ78

Pas de buffer mais un Dictionnaire

Initialement de taille vide (chaîne nulle)
le code occupe deux champs: 1 pointeur dans le dico, le code d'un symbole

les chaînes sont sauvegardées dans le dictionnaire
on garde la mémoire de toutes les chaînes déjà rencontrées

1- index_courant = nul
2- chaîne courante = caractère courant
3- si chaîne courante n'existe pas dans le dictionnaire
l'ajouter en lui attribuant un index
écrire index_courant et le dernier caractère de
chaîne courante dans le fichier compressé
aller en 2
si elle existe
index_courant = index de la chaîne trouvée
chaîne_courante = chaîne_courante+caractère_courant
aller en 3

Compression LZ78 (2)

Exemple

```
sir sid eastman easily teases sea sick seals
```

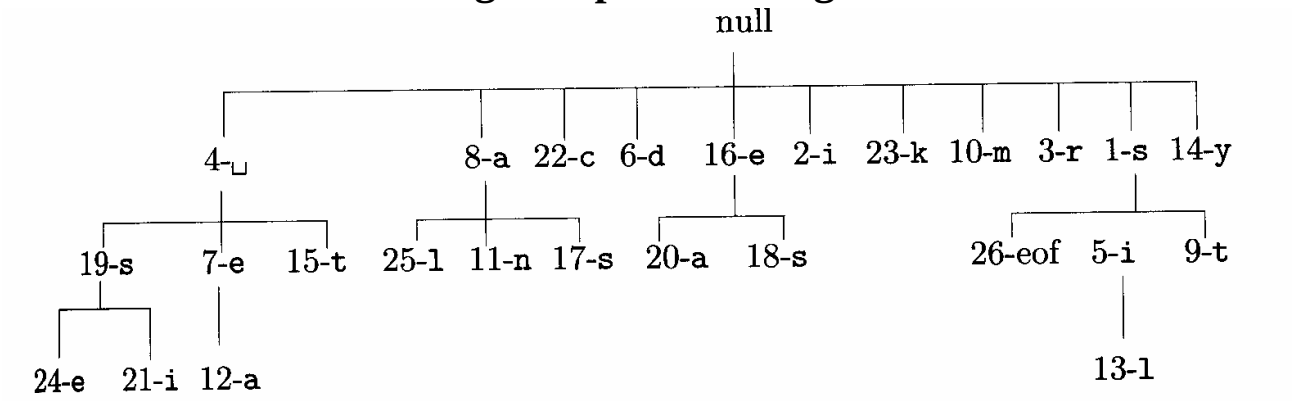
Contenu Dico		code compressé
0	Null	
1	s	(0,s)
2	i	(0,i)
3	r	(0,r)
4	_	(0,_)
5	si	(1,i)
6	d	(0,d)
7	_e	(4,e)
•	a	(0,a)
•	st	(1,t)
•	m	(0,m)
•	an	(8,n)
•	_ea	(7,a)
•	sil	(5,l)
•	y	(0,y)

Compression LZ78 (3)

Organisation du dictionnaire

Grand dico **meilleure compression**

grand pointeurs + grands codes + recherche lente



TRIE: ajout d'un élément = ajout d'un nœud terminal

Gestion du Dico

- sa taille ne peut qu'augmenter

1- Limiter sa taille à une valeur max et compresser moins bien au-delà

2- Détruire le dico et continuer avec un nouveau (travail par blocs)

Compression LZ78 (4)

Décodage

Construire le dico au fur et à mesure du décodage

Compression LZW

Proposé par Terry Welch (84) (sous licence)

1985 Dépôt de licence par Sperry Corp.
1986 Rachat de Sperry par Unisys
1987 CompuServe crée le format GIF et utilise LZW
1994 Après réclamation auprès du bureau des Licence US
tous les utilisateurs de LZW doivent payer la licence
CompuServe (GIF), Aldus (TIFF), Adobe (PostScript level II)
Compress d'UNIX…

Compression LZW (2)

Principe

Le code ne comporte qu'un seul champ: l'adresse dans le dictionnaire
On initialise le dico avec tous les symboles de l'alphabet 8bit = 256 symboles

1- Initialiser la chaîne de travail I à null
2- On lit un caractère courant x
3- Tantque la chaîne Ix existe dans le dictionnaire
I=Ix
x = caractère suivant
Fin
4- Écrire l'index de I dans le fichier compressé
5- ajouter Ix dans le dictionnaire
5- I=x
6- aller en 2

Compression LZW (3)

`sir sid eastman easily teases sea sick seals`

I	Dans dico?	Nouv. Ent	Code de sortie
s	Y		
si	N	256-si	115 (s)
i	Y		
ir	N	257-ir	105 (i)
r	Y		
r_	N	258-r_	114 (r)
_	Y		
_s	N	259-_s	32 (_)
s	Y		
si	Y		
sid	N	260-sid	256 (si)
d	Y		
d_	N	261-d_	100 (d)
_	Y		
_e	N	262-_e	32 (_)
e	Y		
ea	N	263-ea	101 (e)
a	Y		
as	N	264-as	97 (a)
s	Y		
st	N	265-st	115 (s)
t	Y		

Compression LZW (4)

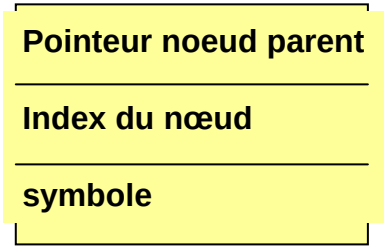
Principe du décodage

1- Initialiser le dico avec les caractères
2- Lire l'index courant dans le fichier compressé
3- Trouver la chaîne correspondante I dans le dico
4- l'Écrire dans le fichier décompressé
5- Lire l'index suivant dans le fichier d'entrée
6- Trouver la chaîne correspondante J dans le dico
7- l'écrire dans le fichier de sortie
8- extraire x le premier caractère de J
9- sauvegarder la chaîne Ix dans le dico
10- I=J
11- aller en 5

Compression LZW (5)

Organisation du dictionnaire

C'est un TRIE chaque nœud contient



- Pointeur Parent permet de remonter vers la racine
- On descend dans l'arbre par Hachage:
le pointeur du fils est calculé en fonction du pointeur courant
et du caractère courant. « index » code ce pointeur

Exemple: on suppose que la chaîne abc existe déjà dans le dictionnaire

adresse	97	266	284	299
parents	-	97	266	284
contenu	a	b	c	d
désigne	a	ab	abc	abcd

Le caractère suivant est un d

- on trouve la chaîne abc
- on cherche à la suite le caractère d : on hache (284 et d). On trouve 299
- si le nœud 299 n'est pas utilisé on l'initialise
- si le nœud existe et contient le même parent (284) et le même caractère (d)
abcd existe déjà dans le dictionnaire
- si le nœud existe mais contient autre chose il y a collision on incrémente le pointeur

Compression LZW (6)

Décodage

1- On initialise le dictionnaire avec les 256 caractères
2- Lire le premier code dans le fichier d'entrée
3- Retrouver la chaîne I dans le dictionnaire
4- Lire le code suivant dans le fichier
5- Retrouver la chaîne J avec son index (pas de hachage)
6- extraire x le premier caractère de J
7- sauvegarder Ix dans le dico
8- J=I
9- aller en 4

On trouve la chaîne J dans l'ordre inversé en remontant jusqu'à la racine

Format GIF

Format de fichier graphique utilisant une variante de LZW

pour des images comportant b niveaux de gris
le dictionnaire comporte initialement 2^{b+1} entrées possibles
sa taille est doublée quant il est plein, en restant < 4096
dans ce cas on peut décider de le détruire et repartir à 0
le code 2^{b} indique le changement de dictionnaire

pour sauvegarder les dictionnaires
les pointeurs comporte un bit de plus à chaque changement de dico
ils sont accumulés et codés dans des blocs de 8 bits

la compression des images est mauvaise car les images sont scannées par lignes

Exercice 1

La séquence suivante a été codée selon la méthode LZW. Le dictionnaire de base est uniquement constitué des 26 caractères de l'alphabet et de l'espace. L'indexation débute à partir de 0 et suit l'ordre alphabétique, l'espace est donc codé par l'index 26. On demande de décoder cette séquence en précisant à chaque étape le contenu du dictionnaire.

11 4 18 26 2 7 0 20 18 18 4 19 19 28 26 36 31 28

Dictionnaire initial

```
A 0
B 1
C 2
D 3
E 4
F 5
G 6
H 7
I 8
J 9
K 10
L 11
M 12
N 13
O 14
P 15
Q 16
R 17
S 18
T 19
U 20
V 21
W 22
X 23
Y 24
Z 25
_ 26
```

Construction du dictionnaire

```
LE  27
ES  28
S_  29
_C  30
CH  31
HA  32
AU  33
US  34
SS  35
SE  36
ET  37
TT  38
TE  39
ES_ 40
_S  41
```

Etapes de décompression

```
I = l
J = e
27 = le         I = e
J = s
28 = es         I = s
J = _
29 = s_         I = _
J = c
30 = _c         I = c
J = h
31 = ch         I = h
J = a
32 = ha         I = a
J = u
33 = au         I = u
J = s
34 = us         I = s
J = s
35 = ss         I = s
J = e
36 = se         I = e
J = t
37 = et         I = s
J = t
38 = tt         I = t
J = es
39 = te         I = es
J = _
40 = es         I = _
J = se
41 = _s         I = se
J = ch
42 = sec        I = ch
J = es
```

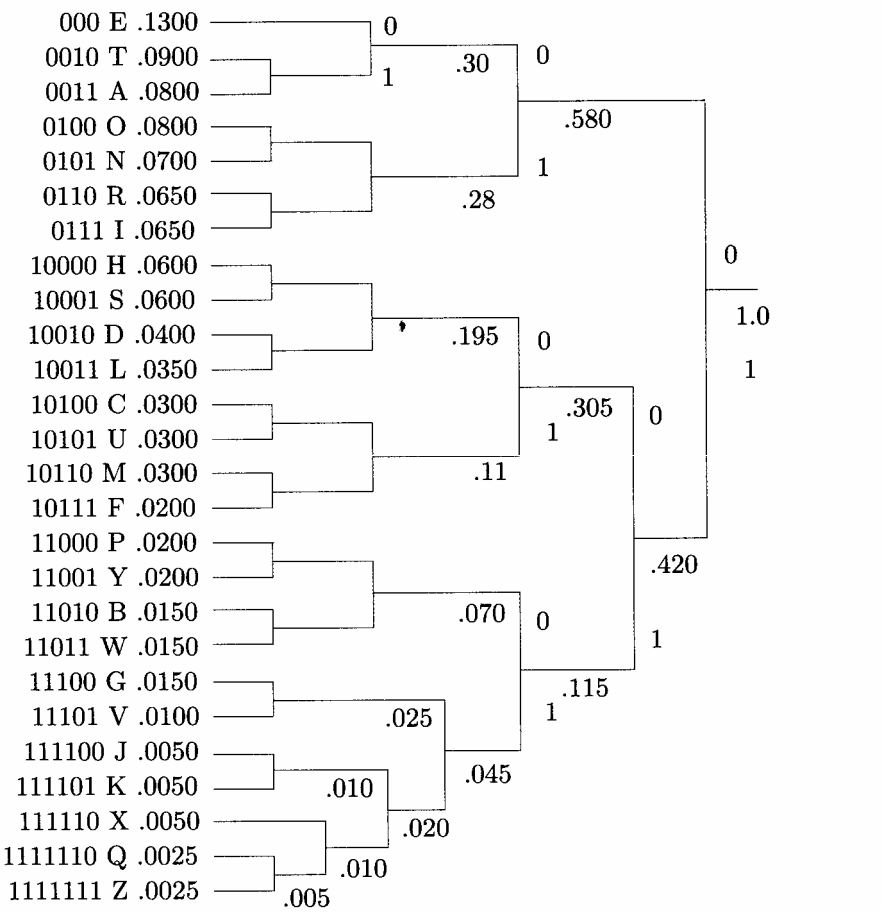
Message décompressé

?????.....

Exercice 2

Cet arbre représente l'arbre d'Huffman des 26 caractères de l'alphabet anglais.

Calculer la taille moyenne de ce code, son entropie et sa redondance



$$Taille\_Moyenne = \sum_{s=1}^{n} Nbit(s) * P_s = 4{,}315$$

$$E = -\sum_{s=1}^{n} P_s \log_2(P_s) = 3{,}9888$$

$$R = -\log_2(n) - \left(-\sum_{i=1}^{n} p_i \log_2(p_i)\right) = 0{,}7117$$

Bibliographie

Nicolas Moreau, Techniques de Compression des Signaux, Masson, 1994.

David Salomon, Data Compression, The Complete Reference, 2nd Edition, Springer 2000.