

Représentation et codage de l'information

**7-Codage de vidéos,  
Compression sans pertes**

L1 Informatique, Université d'Orléans

Florent Foucaud, 2019

# Codage de vidéos

# Qu'est-ce qu'une vidéo ?

Vidéo = piste sonore + séquence d'images (25-30/seconde)

codées/décodées par un **codec** ("codeur-décodeur")

éventuellement : diverses pistes audio, sous-titres...

Pour contenir tout cela il faut un **format conteneur**

On a donc :

- Différents standards pour le format de la vidéo
- Différents codecs (pour l'audio et la vidéo)
- Différents formats conteneurs

# Historique : quelques standards

- H.120 (1984) et **H.261** (1988)

*ITU*



- MPEG-1 (1991)

*Moving Picture Expert Group : ISO + IEC*



- MPEG-2 Part 2 / H.262 (1994)

*ISO + IEC + ITU*

- MPEG-4 Part 2 / H.264, “Advanced Video Coding” (2001)

*ISO + IEC + ITU*

streaming, TNT, blu-ray...

- H.265, “High Efficiency Video Coding” (2013)

*ISO + IEC + ITU*

Ces standards définissent la résolution d'image, la méthode de compression, le débit (taille/seconde), etc.

# Quelques formats conteneurs

- VOB (Video Object) : DVD, 1995 
- AVI (Audio Video Interleave) : Microsoft, 1992 
- WMV (Windows Media Video) : Microsoft, 2003 
- OGG : Xiph.org foundation, ~2005 
- MKV (Matroska Video) : 2002 (format libre)  
- FLV (Flash Video) : Adobe Systems, ~2002 
- MP4 (MPEG-4 Part 14) : MPEG, 2003 
- M4V : Apple (version de MP4 avec des DRM) 

# Codecs vidéo

- HuffYUV (Ben Rudiak-Gould, 2000)  
compression sans pertes





Ben Rudiak-Gould



Fabrice Bellard

## Quelques codecs H.264 / MPEG4-Part 2 :

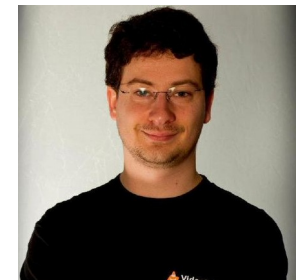
- Suite FFmpeg (Fabrice Bellard, 2000)  **FFmpeg**
- DivX :- ) / DivX (Jérôme Rota, 1998/2001) **DivX**
- Xvid (“fork” libre de DivX, 2001)  **xvid**  
codec
- X264 (VideoLAN, 2004)  



Jérôme Rota  
aka “Gej”

## Codec H.265 :

- X265 (MulticoreWare, 2013) **x265**



Jean-Baptiste Kempf

# **Compression sans pertes**

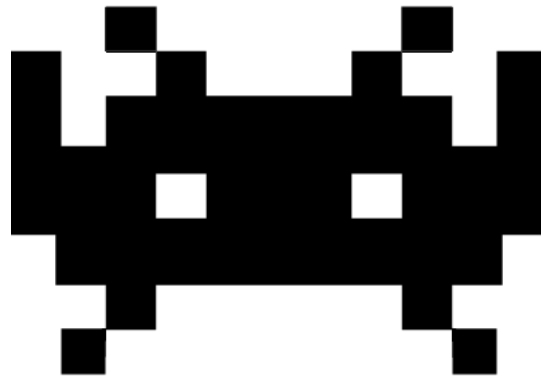
# Run-Length Encoding, RLE

## (Codage par plages)

Principe : tirer parti des répétitions

*Exemple* : images en noir (1) et blanc (0)

On écrit successivement le nombre de 0, le nombre de 1, etc.



```
11      # largeur
7       # hauteur
00100000100
10010001001
10111111101
11101110111
01111111110
00100000100
01000000010
```

Codage sans compression : 77 valeurs

```
0010000010010010001001101111111011110111011101111111110001000001001000000010
```

Codage avec compression RLE : 31 valeurs

*(MAIS : à coder sur 4 bits chacune !!!)*

```
2 1 5 1 2 1 2 1 3 1 2 2 1 7 1 4 1 3 1 3 1 9 3 1 5 1 3 1 7 1 1
```

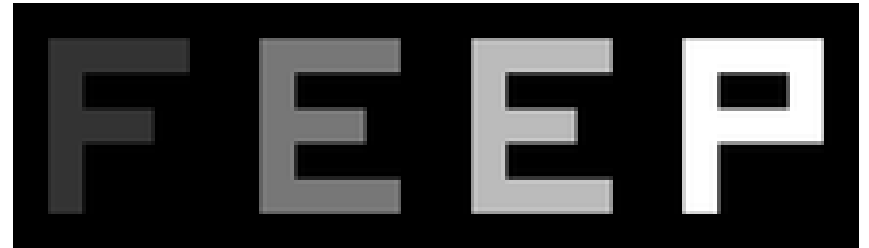


# Run-Length Encoding, RLE

Comment faire si on a plusieurs valeurs ?

séquence de : <nombre de répétitions> <valeur répétée>

Exemple avec un fichier PGM :



```
P2
24 7          # largeur, hauteur
15           # nombre de niveaux de gris entre noir (=0) et blanc (=valeur)
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 3 3 3 3 0 0 7 7 7 7 0 0 11 11 11 11 0 0 15 15 15 15 0
0 3 0 0 0 0 0 7 0 0 0 0 0 11 0 0 0 0 0 15 0 0 15 0
0 3 3 3 0 0 0 7 7 7 0 0 0 11 11 11 0 0 0 15 15 15 15 0
0 3 0 0 0 0 0 7 0 0 0 0 0 11 0 0 0 0 0 15 0 0 0 0
0 3 0 0 0 0 0 7 7 7 7 0 0 11 11 11 11 0 0 15 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

168 valeurs

On obtient :

```
25 0 4 3 2 0 4 7 2 0 4 11 2 0 4 15 2 0 1 3 5 0 1 7 5 0 1 11 5 1 1
15 2 0 1 15 2 0 3 3 3 0 3 7 3 0 3 11 3 0 4 15 2 0 1 3 5 0 1 7 5 0
1 11 5 0 1 15 5 0 1 3 5 0 4 7 2 0 4 11 2 0 1 15 26 0
```

86 valeurs

# Run-Length Encoding, RLE

Peut-on compresser du **texte** avec RLE ?

Inefficace : chaque caractère est répété au plus 2 fois !

Un texte “compressé” avec RLE serait en fait plus long...

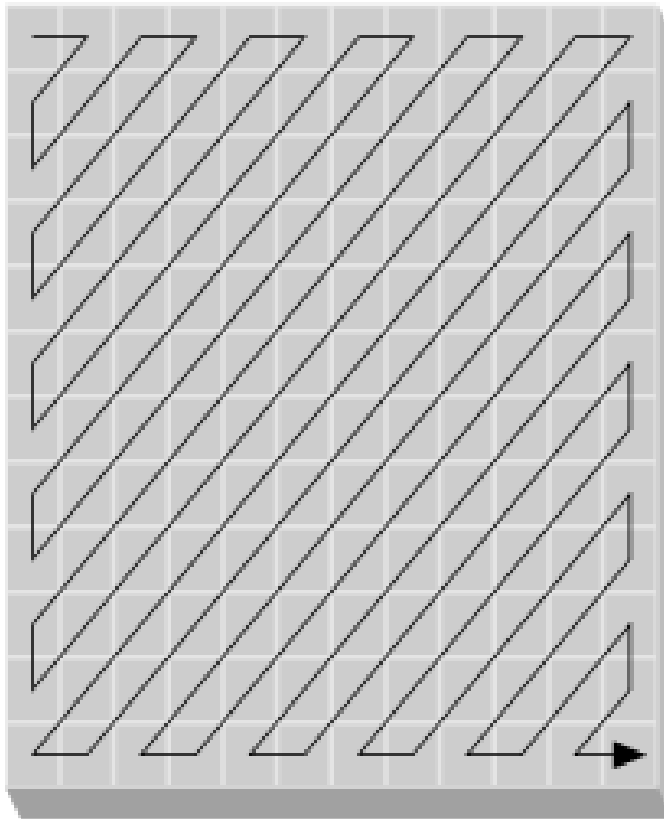
Solution possible :

- Compresser les données binaires

# Run-Length Encoding, RLE

Autres méthodes possibles :

- Compression par blocs de longueur fixée
- encodage “zig-zag” (utilisé dans les formats JPEG et MPEG) :



# Run-Length Encoding, RLE

## **Conclusion :**

- Méthode simple
- Efficace pour les données avec beaucoup de répétitions
- PAS efficace pour les textes !

## **Utilisation :**

- utilisé dans les FAX, où il est très efficace (documents Noir & Blanc avec beaucoup de lignes blanches)

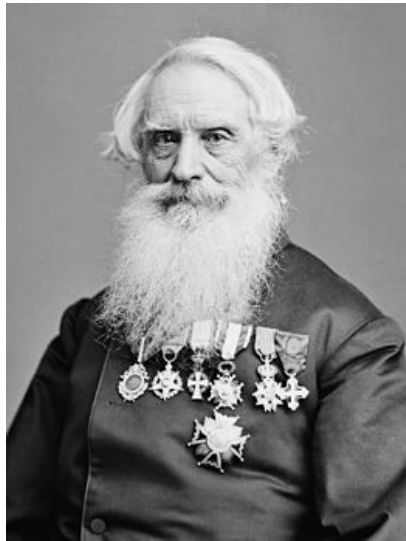
Combiné avec d'autres méthodes :

- BMP
- PNG
- JPEG
- ZIP

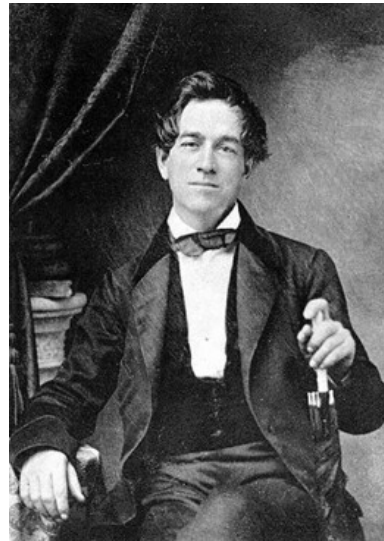


# Code Morse (1838)

Code binaire à longueur variable, inventé pour le télégraphe



Samuel Morse



Alfred Vail

## Code morse international

1. Un tiret est égal à trois points.
2. L'espace entre deux éléments d'une même lettre est égal à un point.
3. L'espace entre deux lettres est égal à trois points.
4. L'espace entre deux mots est égal à sept points.

A	● —	U	● ● —
B	— ● ● ●	V	● ● ● —
C	— ● — ●	W	● — —
D	— ● ●	X	— ● ● —
E	●	Y	— ● — —
F	● ● — ●	Z	— — ● ●
G	— — ●		
H	● ● ● ●		
I	● ●		
J	● — — —		
K	— ● —		
L	● — ● ●		
M	— —		
N	— ●		
O	— — —		
P	● — — ●		
Q	— — ● —		
R	● — ●		
S	● ● ●		
T	—		
		1	● — — —
		2	● ● — —
		3	● ● ● —
		4	● ● ● ●
		5	● ● ● ● ●
		6	— ● ● ● ●
		7	— — ● ● ●
		8	— — — ● ●
		9	— — — — ●
		0	— — — — —



# Code Morse (1838)

Code binaire à longueur variable, inventé pour le télégraphe

## Code morse international

1. Un tiret est égal à trois points.
2. L'espace entre deux éléments d'une même lettre est égal à un point
3. L'espace entre deux lettres est égal à trois points.
4. L'espace entre deux mots est égal à sept points.

## Inconvénient :

Certains codes sont des préfixes d'un autre !

### Exemples :

E préfixe de A, F, H, I, J...

D préfixe de X

M préfixe de G

...

A	● ■	U	● ● ■
B	■ ● ● ●	V	● ● ● ■
C	■ ● ■ ●	W	● ■ ■
D	■ ● ●	X	■ ● ● ■
E	●	Y	■ ● ■ ■
F	● ● ■ ●	Z	■ ■ ● ●
G	■ ■ ●		
H	● ● ● ●		
I	● ●		
J	● ■ ■ ■		
K	■ ● ■		
L	● ■ ● ●		
M	■ ■		
N	■ ●		
O	■ ■ ■		
P	● ■ ■ ●		
Q	■ ■ ● ■		
R	● ■ ●		
S	● ● ●		
T	■		
		1	● ■ ■ ■ ■
		2	● ● ■ ■ ■
		3	● ● ● ■ ■
		4	● ● ● ● ■
		5	● ● ● ● ●
		6	■ ● ● ● ●
		7	■ ■ ● ● ●
		8	■ ■ ■ ● ●
		9	■ ■ ■ ■ ●
		0	■ ■ ■ ■ ■

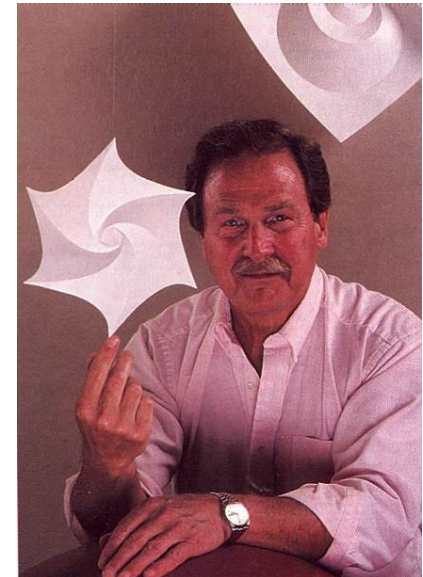
# Codage de Huffman (1952)

## Principes :

- Méthode statistique :  
plus un caractère est fréquent, plus son code est court
- Codage “par préfixe” pour faciliter le décodage :  
Pour deux symboles codés avec  $c1$  et  $c2$ ,
  - $c1$  n'est pas un préfixe de  $c2$
  - $c2$  n'est pas un préfixe de  $c1$
- Le codage se fait avec un “arbre de Huffman”

## Utilisation :

- méthode DEFLATE (ZIP, gzip, PNG,...)
- combiné avec compression à pertes : MP3, JPEG
- ...



David A. Huffman



# Codage de Huffman, algorithme

*“ceci est un petit exemple”*

1) calculer la fréquence de chaque symbole

e=6 ; espace=4 ; t=3 ; c=2 ; i=2 ; p=2 ; s=1 ; u=1 ; n=1 ; x=1 ; m=1 ; l=1

2) Calcul de l'arbre de Huffmann :

- Créer un noeud  $n_s$  pour chaque symbole  $s$
- Le poids  $p(n_s)$  est égal à la fréquence de  $s$
- Tant qu'il reste deux noeuds sans parent :
  - soient  $n1$ ,  $n2$  deux noeuds de poids minimum sans parent
  - créer un noeud  $n$  de poids  $p(n)=p(n1)+p(n2)$
  - $n$  devient le parent de  $n1$  et  $n2$

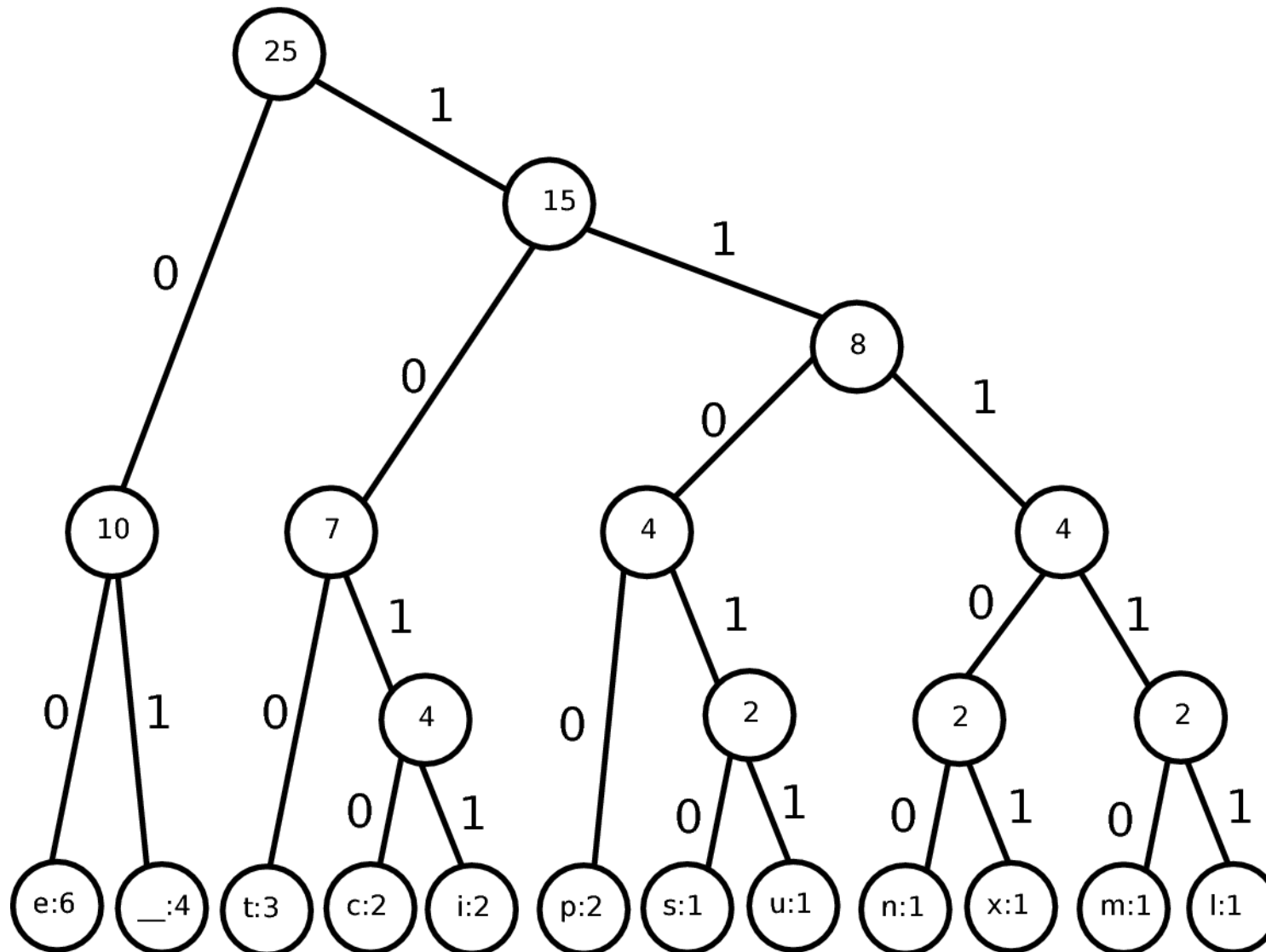
3) Codage d'un symbole : déterminé par le chemin de la racine vers la feuille (gauche=0, droite=1)

# Codage de Huffman, exemple

*“ceci est un petit exemple”*

e=6 ; espace=4 ; t=3 ; c=2 ; i=2 ; p=2 ; s=1 ; u=1 ; n=1 ; x=1 ; m=1 ; l=1

CODAGE :



e : 00  
espace : 01  
t : 100  
c : 1010  
l : 1011  
p : 1100  
s : 11010  
u : 11011  
n : 11100  
x : 11101  
m : 11110  
l : 11111

# Codage de Huffman, exemple

*“ceci est un petit exemple”*

e=6 ; espace=4 ; t=3 ; c=2 ; i=2 ; p=2 ; s=1 ; u=1 ; n=1 ; x=1 ; m=1 ; l=1

CODAGE :

On obtient :

1010 00 1010 1011 01 00 11010 100 01 11011  
11100 01 1100 00 100 1011 100 01 00 11101 00  
11110 1100 11111 00

Soit 83 bits

au lieu de  $4 \times 25 = 100$  bits pour un codage à longueur fixe (sur 4 bits/symbole)

ou même  $7 \times 25 = 175$  bits en ASCII (7bits/symbole)

e : 00  
espace : 01  
t : 100  
c : 1010  
i : 1011  
p : 1100  
s : 11010  
u : 11011  
n : 11100  
x : 11101  
m : 11110  
l : 11111

# Codage de Huffman, décodage

- Aucun code n'est le préfixe d'un autre :  
permet un décodage sans ambiguïté  
  
(d'où l'appellation "code par préfixe")
- On doit transmettre la table des valeurs (ou l'arbre lui-même)

# Codage de Huffman, conclusion

- **Inconvénients :**
  - pas de gain si tous les symboles ont la même fréquence
  - il faut lire les données 2 fois : calcul de fréquences + codage
- Variante “adaptative” pour les flux de données (“streaming”) l’arbre est alors modifié de façon dynamique

# Algorithmes de Lempel-Ziv

(1977, 1978)

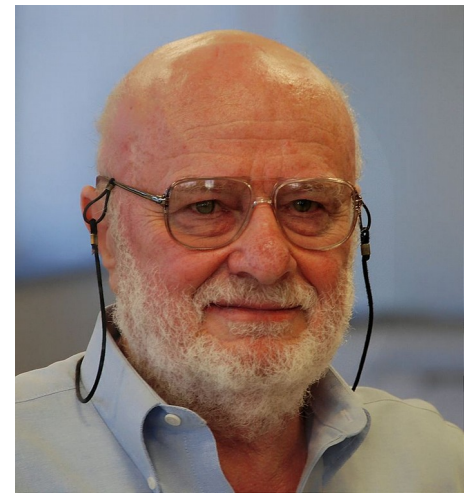
## Principes :

- Codage “à la volée” : une seule passe
- LZ77 : fenêtre glissante
- LZ78 : dictionnaire + codage “récuratif”

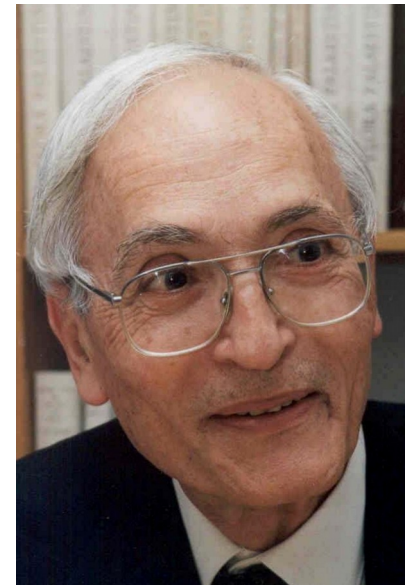
## Utilisations :

- méthode DEFLATE : ZIP, gzip, PNG, ...  
(LZ77 combiné avec Huffman)
- système de fichiers Microsoft NTFS
- jeux vidéo “Electronic Arts”

...



Abraham Lempel



Jacob Ziv

# Lempel-Ziv : exemple (LZ78)

Exemple :      a a b a a c a b c a b c b a b c a c b c b a

	valeur	code
0	NULL	
1		
2		
3		
4		
5		
6		
7		
8		
9		

# Lempel-Ziv : exemple (LZ78)

Exemple :      a a b a a c a b c a b c b a b c a c b c b a

	valeur	code
0	NULL	
1	a	0, a
2		
3		
4		
5		
6		
7		
8		
9		

Code obtenu :

0 a



# Lempel-Ziv : exemple (LZ78)

Exemple :     a **a** b a a c a b c a b c b a b c a c b c b a

	valeur	code
0	NULL	
1	a	0, a
2	a b	1, b
3		
4		
5		
6		
7		
8		
9		

Code obtenu :

0 a 1 b

# Lempel-Ziv : exemple (LZ78)

Exemple :      a a b a a c a b c a b c b a b c a c b c b a

	valeur	code
0	NULL	
1	a	0, a
2	a b	1, b
3	a a	1, a
4		
5		
6		
7		
8		
9		

Code obtenu :

0 a 1 b 1 a

# Lempel-Ziv : exemple (LZ78)

Exemple :      a a b a a c a b c a b c b a b c a c b c b a

	valeur	code
0	NULL	
1	a	0, a
2	a b	1, b
3	a a	1, a
4	c	0, c
5		
6		
7		
8		
9		

Code obtenu :

0 a 1 b 1 a 0 c

# Lempel-Ziv : exemple (LZ78)

Exemple :      a a b a a c **a b c** a b c b a b c a c b c b a

	valeur	code
0	NULL	
1	a	0, a
2	a b	1, b
3	a a	1, a
4	c	0, c
5	a b c	2, c
6		
7		
8		
9		

Code obtenu :

0 a 1 b 1 a 0 c 2 c

# Lempel-Ziv : exemple (LZ78)

Exemple :     a a b a a c a b c **a b c b** a b c a c b c b a

	valeur	code
0	NULL	
1	a	0, a
2	a b	1, b
3	a a	1, a
4	c	0, c
5	a b c	2, c
6	a b c b	5, b
7		
8		
9		

Code obtenu :

0 a 1 b 1 a 0 c 2 c 5 b

# Lempel-Ziv : exemple (LZ78)

Exemple :      a a b a a c a b c a b c b **a b c a c b c b a**

	valeur	code
0	NULL	
1	a	0, a
2	a b	1, b
3	a a	1, a
4	c	0, c
5	a b c	2, c
6	a b c b	5, b
7	a b c a	5, a
8		
9		

Code obtenu :

0 a 1 b 1 a 0 c 2 c 5 b 5 a

# Lempel-Ziv : exemple (LZ78)

Exemple :      a a b a a c a b c a b c b a b c a **c b** c b a

	valeur	code
0	NULL	
1	a	0, a
2	a b	1, b
3	a a	1, a
4	c	0, c
5	a b c	2, c
6	a b c b	5, b
7	a b c a	5, a
8	c b	4, b
9		

Code obtenu :

0 a 1 b 1 a 0 c 2 c 5 b 5 a 4 b

# Lempel-Ziv : exemple (LZ78)

Exemple :      a a b a a c a b c a b c b a b c a c b **c b a**

	valeur	code
0	NULL	
1	a	0, a
2	a b	1, b
3	a a	1, a
4	c	0, c
5	a b c	2, c
6	a b c b	5, b
7	a b c a	5, a
8	c b	4, b
9	c b a	8, a

Code obtenu :

0 a 1 b 1 a 0 c 2 c 5 b 5 a 4 b 8 a

18 symboles (contre 22)



# Lempel-Ziv : codage (LZ78)

Exemple :      a a b a a c a b c a b c b a b c a c b c b a

	valeur	code
0	NULL	
1	a	0, a
2	a b	1, b
3	a a	1, a
4	c	0, c
5	a b c	2, c
6	a b c b	5, b
7	a b c a	5, a
8	c b	4, b
9	c b a	8, a

## Algorithme de codage pour l'entrée T :

- Initialiser un dictionnaire D: int / (int, char)
- $s := T[0]$  //chaîne courante
- Tant que la chaîne s n'est pas vide :
  - soit i l'indice de  $s[0] \dots s[|s|-1]$  dans D
  - ajouter la valeur (i,  $s[|s|]$ ) à D
  - écrire i  $s[|s|]$
  - $s :=$  prochaine chaîne de T pas dans D

# Lempel-Ziv : décodage (LZ78)

Exemple : 0 a 1 b 1 a 0 c 2 c 5 b 5 a 4 b 8 a

	valeur	code
0	NULL	
1	a	0, a
2	a b	1, b
3	a a	1, a
4	c	0, c
5	a b c	2, c
6	a b c b	5, b
7	a b c a	5, a
8	c b	4, b
9	c b a	8, a

## Algorithme de décodage pour l'entrée T :

- Initialiser le dictionnaire D
- Pour chaque paire (i, c) de T faire :
  - ajouter la valeur au dictionnaire D
  - $j := i$
  - $s := c$
  - tant que  $i \neq 0$  :
    - $s := D[i][2] + s$
    - $i := D[i][j]$
  - écrire s

# Lempel-Ziv : variantes



James  
Storer



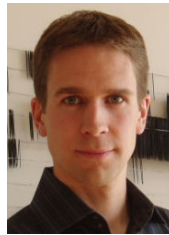
Thomas G.  
Szymanski

## Variantes de LZ77 (à "fenêtre glissante") :

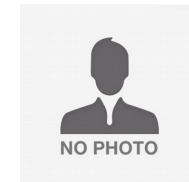
- Lempel-Ziv-Storer-Szymanski, 1982 :  
PKZIP – RAR – GameBoyAdvance  
*idée : ne compresser que quand cela vaut la peine*
- LZX, 1995 :  
AMIGA – Microsoft CAB+CHM+Xbox Live  
*idée : dictionnaire stocké de façon plus efficace*
- LZMA (Lempel-Ziv Markov Chain), 1996 : 7zip  
*idée : grand dictionnaire  
+ calcul de probabilités d'apparition*



Jonathan  
Forbes



Toumi  
Poutanen



Igor Pavlov



Terry A.  
Welch

## Variantes de LZ78 :

- LZW (Lempel-Ziv-Welch), 1984 : compress – GIF – PDF  
*idée : initialiser tous les mots de taille 1 par défaut*

# Lempel-Ziv : conclusion

- En pratique : taille du dictionnaire fixée
- **Avantages :**
  - une seule passe
  - optimal (en termes de théorie de l'information de Shannon)

# DEFLATE (LZ + Huffman)



Phillip W. Katz

- Double passe : LZ77 + Huffman
- Initialement conçu pour PKZIP (MS-DOS, 1989)
- Utilisé dans : PNG, ZIP, gzip (GNU zip)

# Transformée de Burrows-Wheeler (1994)

**Idée** : mélanger le texte pour créer des répétitions

(la transformation doit être réversible !)

**Attention** : ce n'est pas une méthode de compression (c'est une étape préliminaire)

Ensuite, on applique notre compression préférée :  
RLE, Huffman, Lempe-Ziv, etc.

Technique très utilisée en bio-informatique :  
Codage du génome avec 4 symboles – A, C, G, T



Michael Burrows



David J. Wheeler



# Burrows-Wheeler : codage



Banana for scale

Exemple de texte :  $t = BANANA$

1) ajouter des marqueurs de début et fin :  $\$ BANANA \#$

2) calculer les  $|t|$  "rotations" du texte

3) les trier dans l'ordre lexicographique

4) renvoyer la dernière **colonne**

$\$ BANANA \#$	$ANANA \# \$ B$
$\# \$ BANANA$	$ANA \# \$ BAN$
$A \# \$ BANAN$	$A \# \$ BANAN$
$NA \# \$ BANA$	$BANANA \# \$$
$ANA \# \$ BAN$	$NANA \# \$ B A$
$NANA \# \$ BA$	$NA \# \$ BAN A$
$ANANA \# \$ B$	$\$ BANANA \#$
$BANANA \# \$$	$\# \$ BANANA$

**Résultat :**

$BNN \$ AA \# A$

# Burrows-Wheeler : décodage



Banana for scale

## Algorithme pour un codage "r":

- Répéter  $|r|$  fois :
  - ajouter "r" comme première colonne
  - trier les lignes lexicographiquement
- Renvoyer la ligne qui commence par \$ (caractère de début)

**Exemple :  $r = B N N \$ A A \# A$**

B	A	BA	AN	BAN	ANA	BANA	ANAN	BANAN	AAA#\$
N	A	NA	AN	NAN	ANA	NANA	ANA#	NANA#	ANANA
N	A	NA	A#	NA#	A#\$	NA#\$	A#\$B	NA#\$B	A#\$BA
\$	B	\$B	BA	\$BA	BAN	\$BAN	BANA	\$BANA	BANAN
A	N	AN	NA	ANA	NAN	ANAN	NANA	ANANA	NANA#
A	N	AN	NA	ANA	NA#	ANA#	NA#\$	AA#\$	NA#\$B
#	\$	#\$	\$B	#\$B	\$BA	#\$BA	\$BAN	#\$BAN	\$BANA
A	#	A#	#\$	A#\$	#\$B	A#\$B	#\$BA	A#\$BA	#\$BAN

BAAA#\$	ANANA#	BANANA#	ANANANA	BANANANA	ANANA#\$B
NANANA	ANA#\$B	NANA#\$B	ANA#\$BA	NANA#\$BA	ANA#\$BAN
NA#\$BA	A#\$BAN	NA#\$BAN	A#\$BANA	NA#\$BANA	A#\$BANAN
\$BANAN	BAAA#\$	\$BAAA#\$	BANANA#	\$BANANA#	BANANANA
ANANA#	NANANA	ANANANA	NANA#\$B	ANANA#\$B	NANA#\$BA
ANA#\$B	NA#\$BA	ANA#\$BA	NA#\$BAN	ANA#\$BAN	NA#\$BANA
#\$BANA	\$BANAN	#\$BANAN	\$BAAA#\$	#\$BAAA#\$	<b>\$BANANA#</b>
A#\$BAN	#\$BANA	A#\$BANA	#\$BANAN	A#\$BANAN	#\$BAAA#\$



# Burrows-Wheeler : conclusion

- **Avantages :**

- simple à implémenter
- efficace pour du texte (syllabes qui réapparaissent souvent)



# Quelques autres méthodes

- **Codage de Shannon-Fano, 1969**

codage par préfixe, comme Huffman mais moins efficace  
*utilisé par la méthode IMplode (format ZIP)*

- **Codage arithmétique, années 1960**

Idée : chaque symbole est codé par un intervalle de  $[0,1]$   
un texte est représenté par un flottant  
*utilisé dans JPEG2000*

- **Codage de Rice-Golomb, années 1960**

Idée : codage d'une valeur avec son quotient et son reste  
dans la division euclidienne.

*utilisé pour des formats audio (FLAC, Apple lossless, MPEG4-ALS)*