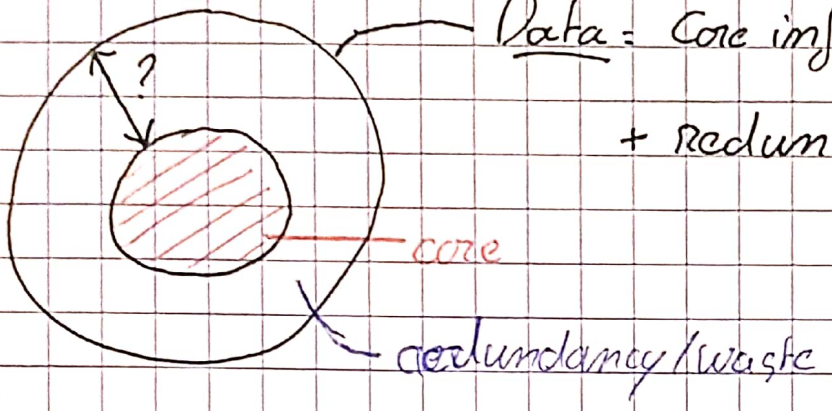
2. Compression de données



Data = Core information (incompressible)
+ Redundancy (waste (compressible)

=> La compression fonctionne bien quand la data est ordonnée

Self-information :

→ Plus un message est "rare", plus il contient d'information

→ Un message qui apparaît avec certitude ne contient pas d'info

→ $q(m) = -\log_2(p(m))$ Sh ← unité (Shanon)

Entropie : Ce qu'il faut au minimum pour compresser l'info sans perte de données

→ $H = -\sum_{i=1}^{N_s} p_i \log_2(p_i)$ Sh/symb

→ C'est la quantité moyenne de self-info des symboles de l'alphabet

→ Symboles équiprobables => full random => entropie max => incompressible

→ Equiprobables => $N_s = 2^m$ => $H = m$

entropie MAX => incompressible
entropie MIN => compressible

→ Taux de compression = $\frac{\text{Volume final}}{\text{Volume initial}}$

→ Ex: $H = 2$ Sh/symb => Chaque symb. peut être encodé sur 2 bits

Lossless compression: Data identique après décompression

→ étape d'analyse pour établir des statistiques

→ fonctionne bien sur du texte, images synthétiques, etc

→ symb. probable (=) encodage court et inversement

Huffman:

S_i	p_i
E	0,4
A	0,3
C	0,1
D	0,1
B	0,06
F	0,04

N_1 = B, F

S_i	P_i
E	0,4
A	0,3
C	0,1
D	0,1
N_1	0,1

N_2 = D, N_1

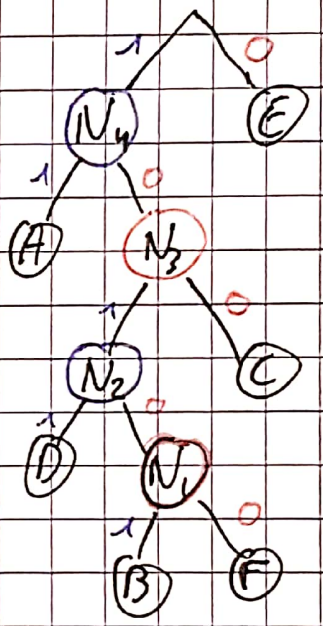
S_i	P_i
E	0,4
A	0,3
N_2	0,2
C	0,1

N_3 = N_2, C

S_i	P_i
E	0,4
A	0,3
N_3	0,3

N_4 = A, N_3

S_i	P_i
N_4	0,6
E	0,4



→ $L = \sum p_i \cdot l(s_i)$ longueur moyenne

→ Désavantages: char/char (pas de mots) et dépend de probas

Burrows-Wheeler: Faciliter la compression en regroupant les occurences (=> RLE)

b	a	n	a	n	a	$
$	b	a	n	a	n	a
a	$	b	a	n	a	n
n	a	$	b	a	n	a
a	n	a	$	b	a	n
n	a	n	a	$	b	a
a	n	a	n	a	$	b

Tri lexico.

1	$	b	a	n	a	n	a
2	a	$	b	a	n	a	n
3	a	n	a	$	b	a	n
4	a	n	a	n	a	$	b
5	b	a	n	a	n	a	$
6	n	a	$	b	a	n	a
7	n	a	n	a	$	b	a

BWT = dernière ligne de la chaîne initiale + dernière colonne = 5annb$aa

Move-to-front: (MTF) ammb$aa ⇒ $\alpha_0 = [a, b, m, \$]$

$a \to \alpha_0[0]$ // a est toujours le 1er symbole de α ⇒ pas de changement
$m \to \alpha_0[2]$ $\alpha_0 \to \alpha_1 = [m, a, b, \$]$
$m \to \alpha_1[0]$
$b \to \alpha_1[2]$ $\alpha_1 \to \alpha_2 = [b, m, a, \$]$ ⇒ MTF(ammb$aa) = 02023300
$\$ \to \alpha_2[3]$ $\alpha_2 \to \alpha_3 = [\$, b, m, a]$
$a \to \alpha_3[3]$ $\alpha_3 \to \alpha_4 = [a, \$, b, m]$
$a \to \alpha_4[0]$

bzip2: Généralement meilleure compression mais plus long à compresser

1) BWT → Tri par symboles identiques
2) MTF → Encodage en digits
3) Huffman → Grosse compression

LZW encoding: TO BE OR NOT TO BE avec $\mathcal{D}$ = table ASCII

Buffer	Input	$\mathcal{D}$[·]	@$\mathcal{D}$[·]	output	
	T				// première lettre
T	O	TO	256	@$\mathcal{D}$[T]	// TO pas dans $\mathcal{D}$ ⇒ @ 256
O	-	O-	257	@$\mathcal{D}$[O]	
..	...	...	...	...	
-	T	-T	268	@$\mathcal{D}$[-]	
T	O			@$\mathcal{D}$[T]	// les @ d'output passent de 8 à 9 bits
TO	-	TO-	269	256	

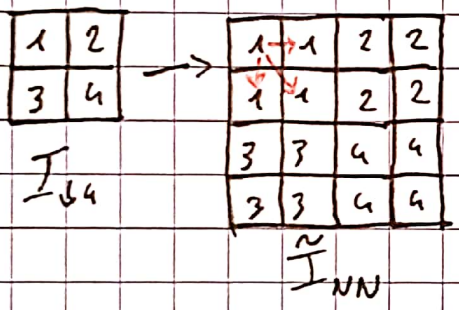
Lossy compression: Data différente après compression

→ On enlève certaines infos : [155][140][156] → [155]

→ Doit être imperceptible par l'utilisateur

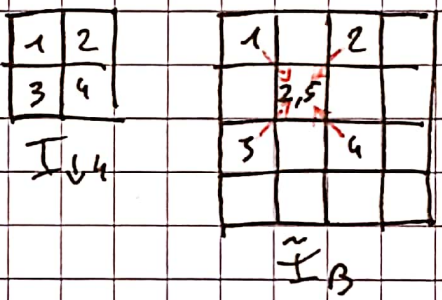
Downsampling / upsampling → diviser/multiplier le nb de pixels

Nearest-neighbor interpolation:



↳ blocking effect

Bilinear interpolation



↳ blurring effect

Hadamard Kernel matrix:

On pose $H_1 = (1)$ et $\forall n = 2^p$, $H_{2N} = \frac{1}{\sqrt{2}} \begin{pmatrix} H_N & H_N \\ H_N & -H_N \end{pmatrix} \Rightarrow H_2 = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$

$H_8 = \frac{1}{2\sqrt{2}} \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 & 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 & 1 & -1 & -1 & 1 \\ 1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 \\ 1 & -1 & 1 & -1 & -1 & 1 & -1 & 1 \\ 1 & 1 & -1 & -1 & -1 & -1 & 1 & 1 \\ 1 & -1 & -1 & 1 & -1 & 1 & 1 & -1 \end{pmatrix}$ # chgt. signe : 0, 7, 3, 4, 1, 6, 2, 5

transf. →

$H_8 = \frac{1}{2\sqrt{2}} \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 \\ 1 & 1 & -1 & -1 & -1 & -1 & 1 & 1 \\ 1 & 1 & -1 & -1 & 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 & 1 & -1 & -1 & 1 \\ 1 & -1 & -1 & 1 & -1 & 1 & 1 & -1 \\ 1 & -1 & 1 & -1 & -1 & 1 & -1 & 1 \\ 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 \end{pmatrix}$: 0, 1, 2, 3, 4, 5, 6, 7

Compression de données

Discrete Cosine Transform: (DCT)

$$D_N(j,k) = \alpha(j)\cos\left(\frac{\pi(2k+1)j}{2N}\right) \text{ avec } \alpha(j) = \begin{cases} \sqrt{\frac{1}{N}} & \text{si } j=0 \\ \sqrt{\frac{2}{N}} & \text{sinon} \end{cases}$$

Ex: $D_8 = \frac{1}{2}\begin{pmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & \cdots & \frac{1}{\sqrt{2}} \\ \cos\frac{\pi}{16} & \cos\frac{3\pi}{16} & \cdots & \cos\frac{15\pi}{16} \\ \vdots & & \ddots & \vdots \\ \cos\frac{7\pi}{16} & \cdots & \cdots & \cos\frac{105\pi}{16} \end{pmatrix}$ $\rightarrow I = D_N^T J_D D_N$

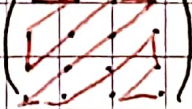
JPEG compression:

1) Découper en blocs 8x8 (padding pixels noirs ou miroir → artefacts)

2) Pour chaque bloc, on calcule le DCT: $J_{B_i} = DCT(B_i - 128)$

3) Quantization du DCT par une matrice Q donnée

↳ on arrondit ces valeurs au plus proche

4) On arrange ces valeurs en zigzag 

5) On utilise Huffman avec des tables de conversion données

=> Seule perte possible: arrondi de l'étape 3