La Compression Des Données

Jalal Fadili

Centenaire de la naissance de Claude Shannon

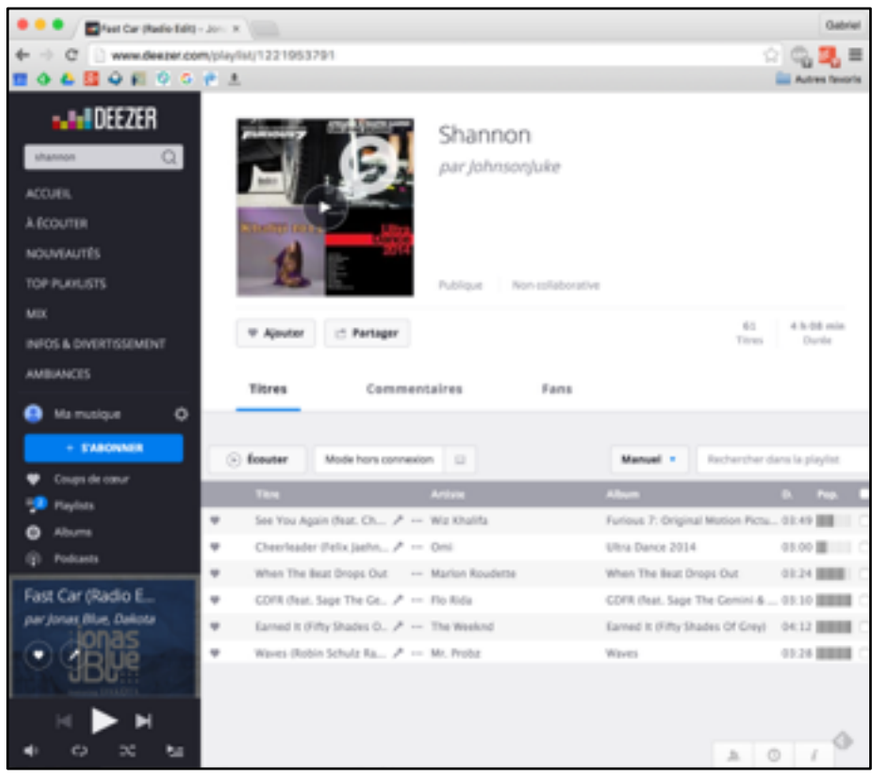
1916-2001

Normandie Université

ENSICAEN
ÉCOLE NATIONALE SUPÉRIEURE D'INGÉNIEURS DE CAEN
& CENTRE DE RECHERCHE

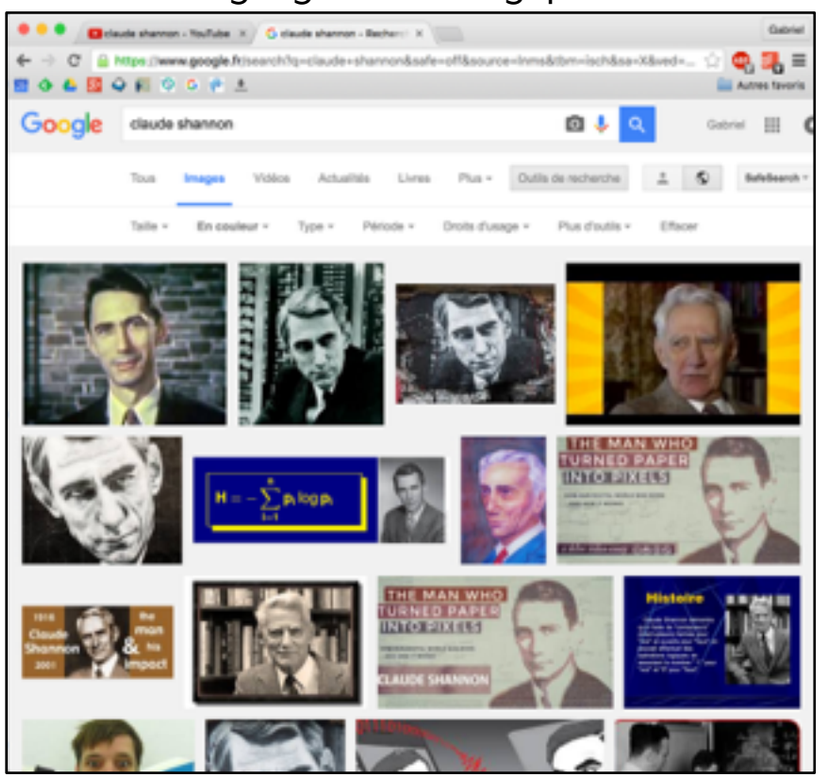
Données, information, communication

www.deezer.com



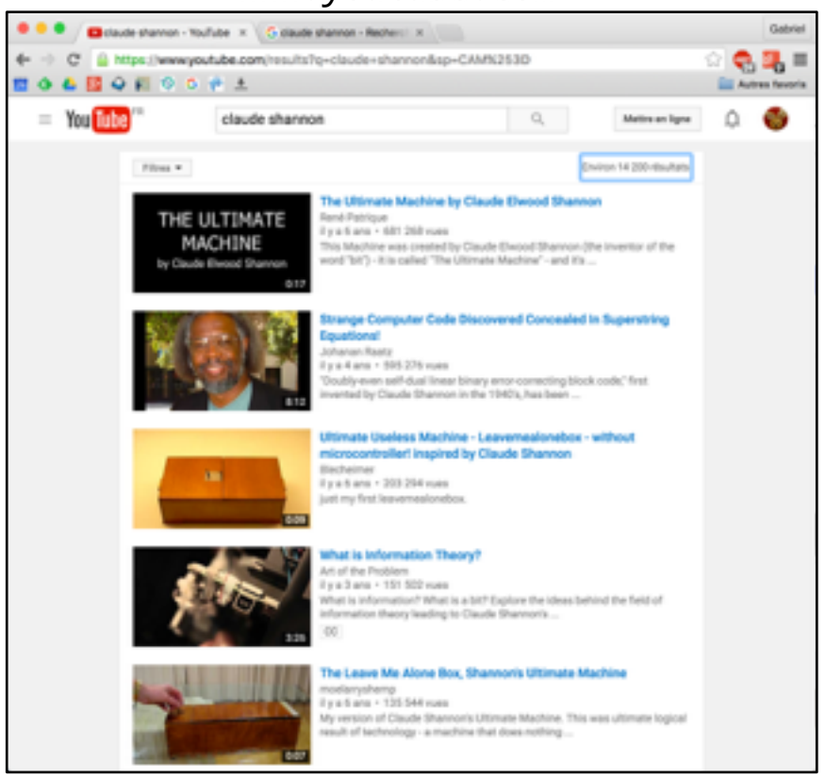
musique

www.google.com/imghp



images

www.youtube.com



vidéos

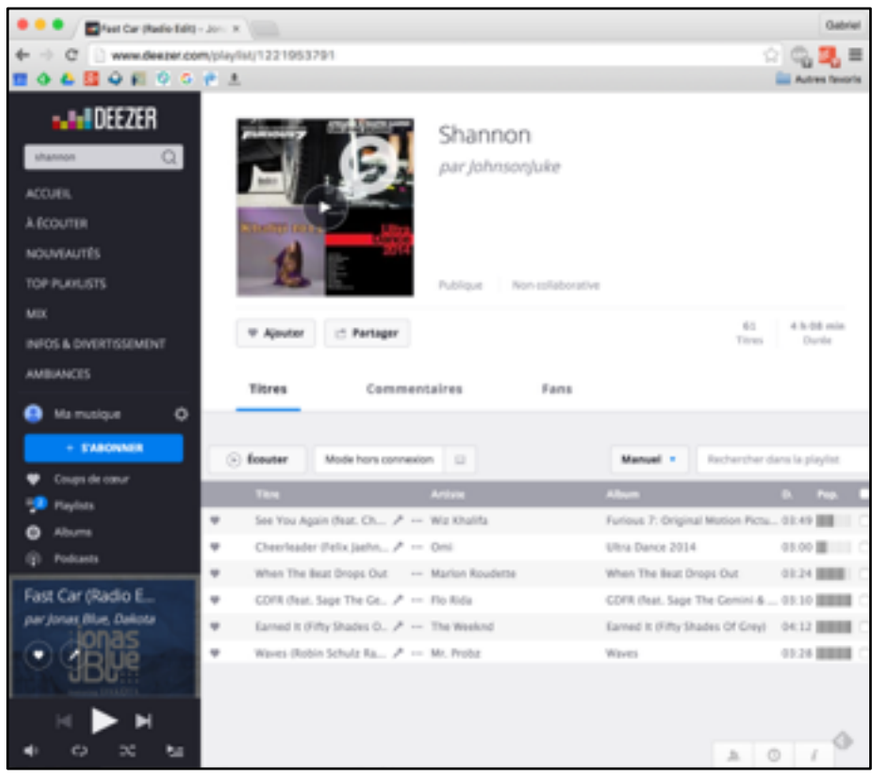
3D, cinéma, jeux vidéos

réseaux sociaux

scanner, imagerie

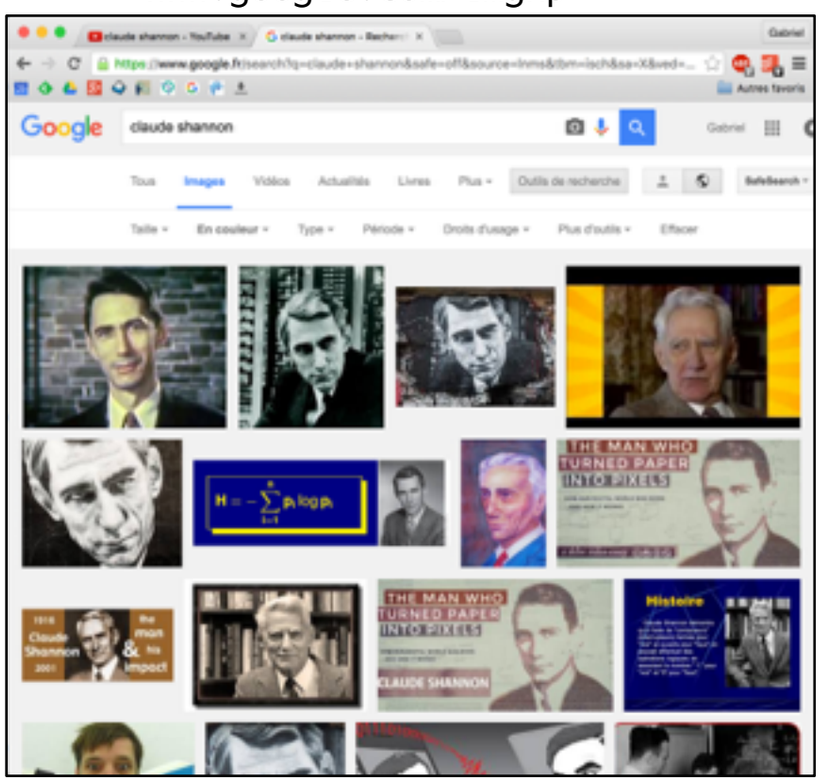
Données, information, communication

www.deezer.com



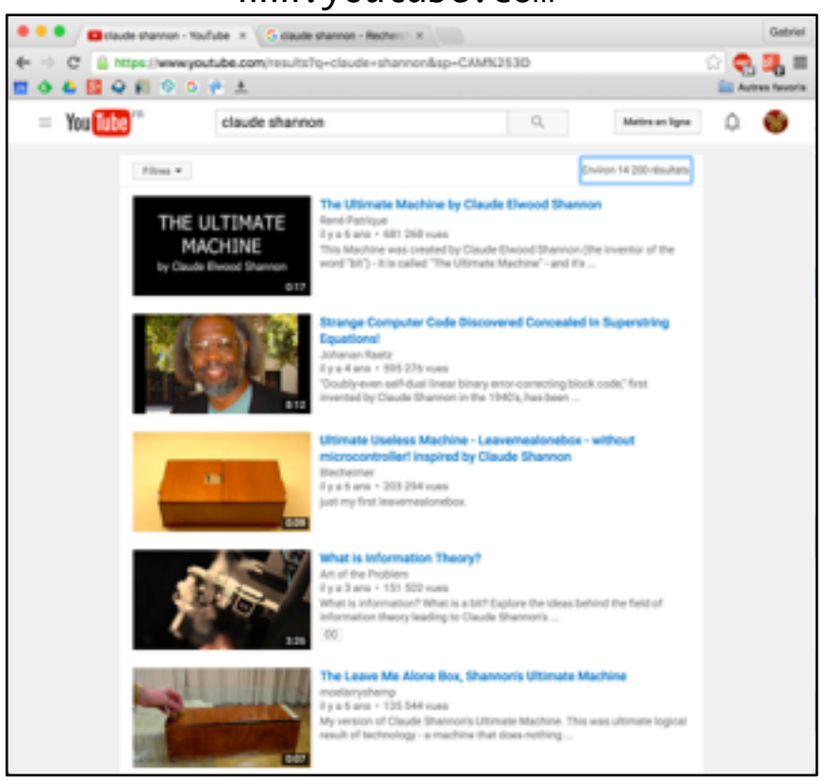
musique

www.google.com/imghp



images

www.youtube.com



vidéos

3D, cinéma, jeux vidéos

réseaux sociaux

scanner, imagerie

3 problèmes : acquérir / stocker / transmettre ces données.

Types de compression

- Soit d une donnée à compresser.
- Un algorithme de codage/compression $C : d \mapsto C(d)$.
- Un algorithme de décodage/décompression $D : c \mapsto D(c)$.
- Compression conservative si $D(C(d)) = d$.
- Compression non-conservative si $D(C(d)) \neq d$ mais $D(C(d)) \approx d$.

Types de compression

- Soit d une donnée à compresser.
- Un algorithme de codage/compression $C : d \mapsto C(d)$.
- Un algorithme de décodage/décompression $D : c \mapsto D(c)$.
- Compression conservative si $D(C(d)) = d$.
- Compression non-conservative si $D(C(d)) \neq d$ mais $D(C(d)) \approx d$.

PNG

Types de compression

- Soit d une donnée à compresser.
- Un algorithme de codage/compression $C : d \mapsto C(d)$.
- Un algorithme de décodage/décompression $D : c \mapsto D(c)$.
- Compression conservative si $D(C(d)) = d$.
- Compression non-conservative si $D(C(d)) \neq d$ mais $D(C(d)) \approx d$.

PNG

$\approx$

JPEG (Q=25%)

Codage uniforme : représentation binaire

Ecriture binaire (base 2):

$[\ldots a_3 a_2 a_1 a_0]_2$ représente $a_0 2^0 + a_1 2^1 + a_2 2^2 + a_3 2^3 + \ldots$

Exemple: $[110]_2 = [1 \times 4 + 1 \times 2 + 0 \times 1]_{10} = [6]_{10}$.

Codage uniforme : représentation binaire

Ecriture binaire (base 2):

$[\ldots a_3 a_2 a_1 a_0]_2$ représente $a_0 2^0 + a_1 2^1 + a_2 2^2 + a_3 2^3 + \ldots$

Exemple: $[110]_2 = [1 \times 4 + 1 \times 2 + 0 \times 1]_{10} = [6]_{10}$.

Codage des nombres $\{0, 1, 2, 3\}$:

$[0]_{10} = [0]_2 \quad \rightarrow \text{code} = 00$

$[1]_{10} = [1]_2 \quad \rightarrow \text{code} = 01$

$[2]_{10} = [10]_2 \rightarrow \text{code} = 10$

$[3]_{10} = [11]_2 \rightarrow \text{code} = 11$

Codage uniforme : représentation binaire

Ecriture binaire (base 2):

$[\ldots a_3 a_2 a_1 a_0]_2$ représente $a_0 2^0 + a_1 2^1 + a_2 2^2 + a_3 2^3 + \ldots$

Exemple: $[110]_2 = [1 \times 4 + 1 \times 2 + 0 \times 1]_{10} = [6]_{10}$.

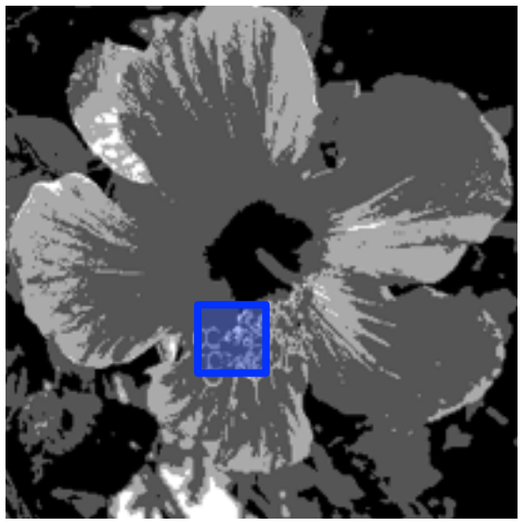
Codage des nombres $\{0, 1, 2, 3\}$:

$[0]_{10} = [0]_2 \quad \rightarrow \text{code} = 00$

$[1]_{10} = [1]_2 \quad \rightarrow \text{code} = 01$

$[2]_{10} = [10]_2 \rightarrow \text{code} = 10$

$[3]_{10} = [11]_2 \rightarrow \text{code} = 11$



Codage uniforme : représentation binaire

Ecriture binaire (base 2):

$[\ldots a_3 a_2 a_1 a_0]_2$ représente $a_0 2^0 + a_1 2^1 + a_2 2^2 + a_3 2^3 + \ldots$

Exemple: $[110]_2 = [1 \times 4 + 1 \times 2 + 0 \times 1]_{10} = [6]_{10}$.

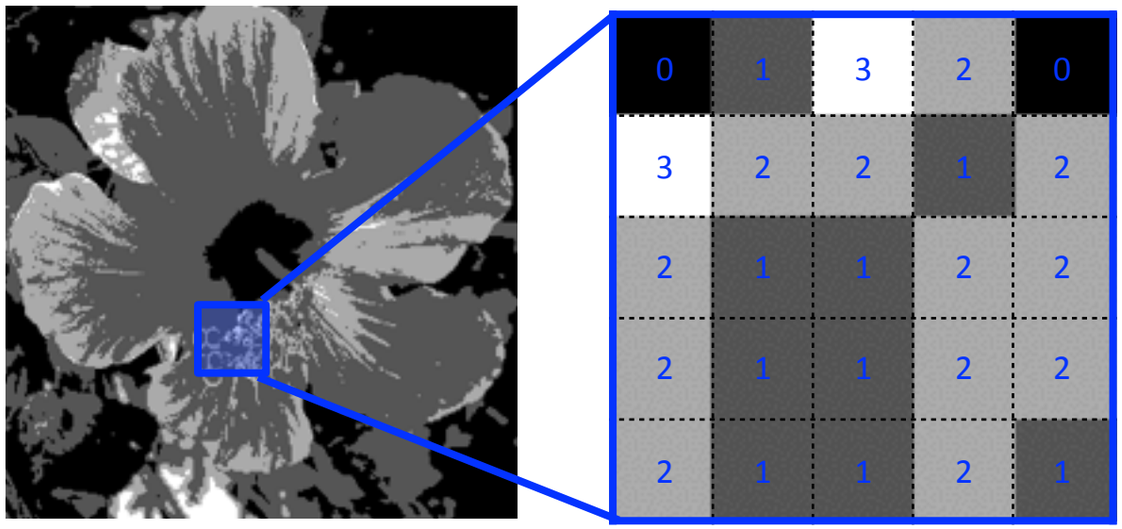
Codage des nombres $\{0, 1, 2, 3\}$:

$[0]_{10} = [0]_2 \quad \rightarrow \text{code} = 00$

$[1]_{10} = [1]_2 \quad \rightarrow \text{code} = 01$

$[2]_{10} = [10]_2 \rightarrow \text{code} = 10$

$[3]_{10} = [11]_2 \rightarrow \text{code} = 11$



Codage uniforme : représentation binaire

Ecriture binaire (base 2):

$[\ldots a_3 a_2 a_1 a_0]_2$ représente $a_0 2^0 + a_1 2^1 + a_2 2^2 + a_3 2^3 + \ldots$

Exemple: $[110]_2 = [1 \times 4 + 1 \times 2 + 0 \times 1]_{10} = [6]_{10}$.

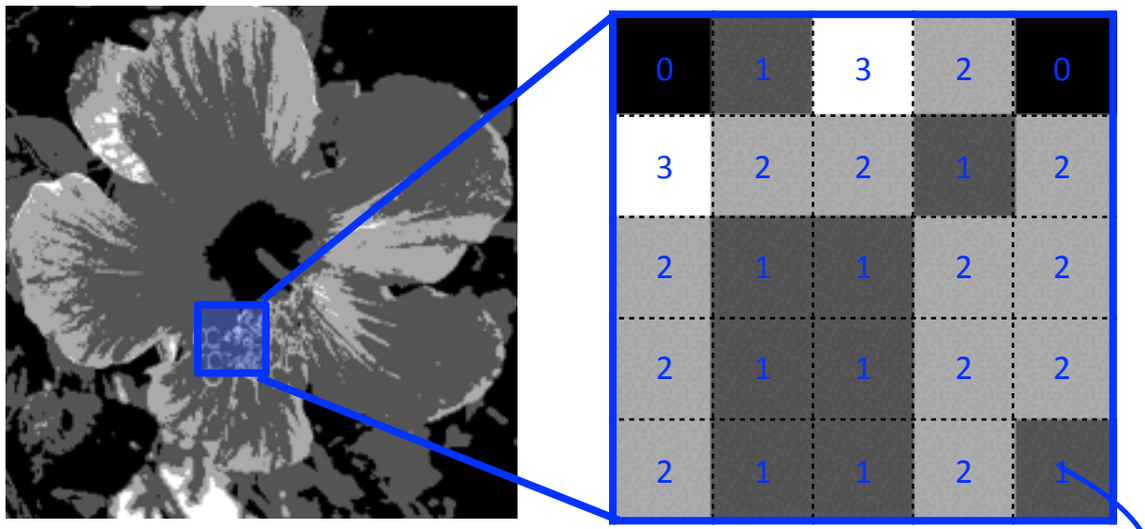
Codage des nombres $\{0, 1, 2, 3\}$:

$[0]_{10} = [0]_2 \quad \rightarrow \text{code} = 00$

$[1]_{10} = [1]_2 \quad \rightarrow \text{code} = 01$

$[2]_{10} = [10]_2 \rightarrow \text{code} = 10$

$[3]_{10} = [11]_2 \rightarrow \text{code} = 11$



Codage uniforme : représentation binaire

Ecriture binaire (base 2):

$[\ldots a_3 a_2 a_1 a_0]_2$ représente $a_0 2^0 + a_1 2^1 + a_2 2^2 + a_3 2^3 + \ldots$

Exemple: $[110]_2 = [1 \times 4 + 1 \times 2 + 0 \times 1]_{10} = [6]_{10}$.

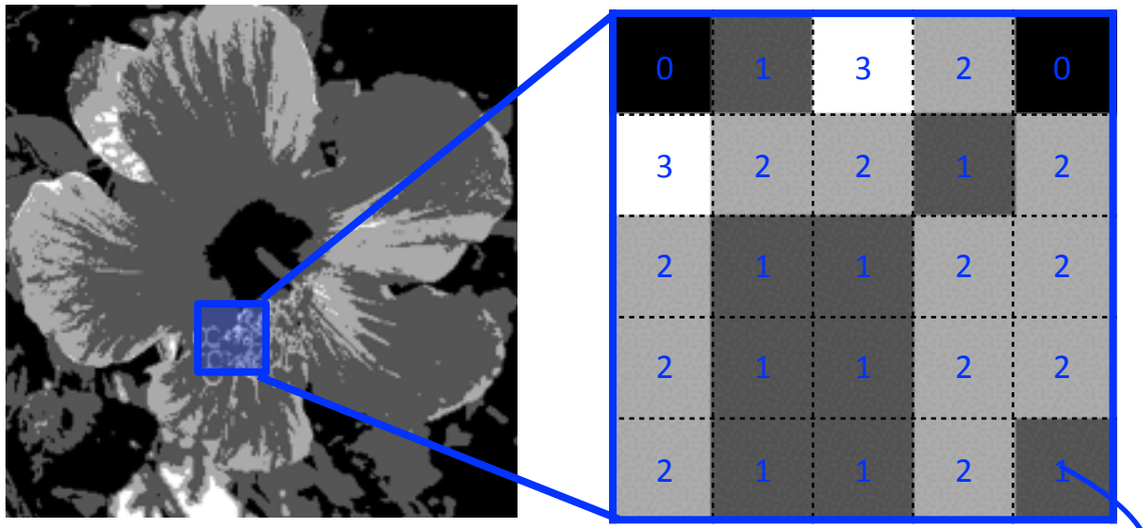
Codage des nombres $\{0, 1, 2, 3\}$:

$[0]_{10} = [0]_2 \quad \rightarrow \text{code} = 00$

$[1]_{10} = [1]_2 \quad \rightarrow \text{code} = 01$

$[2]_{10} = [10]_2 \rightarrow \text{code} = 10$

$[3]_{10} = [11]_2 \rightarrow \text{code} = 11$



codage

0	1	3	2	0	3	2	2	1	2	2	1	1	2	2	2	1	1	2	2	1	1	2	1
00	01	11	10	00	11	10	10	01	10	10	01	01	10	10	10	01	01	10	10	01	01	10	01

Codage uniforme : représentation binaire

Ecriture binaire (base 2):

$[\ldots a_3 a_2 a_1 a_0]_2$ représente $a_0 2^0 + a_1 2^1 + a_2 2^2 + a_3 2^3 + \ldots$

Exemple: $[110]_2 = [1 \times 4 + 1 \times 2 + 0 \times 1]_{10} = [6]_{10}$.

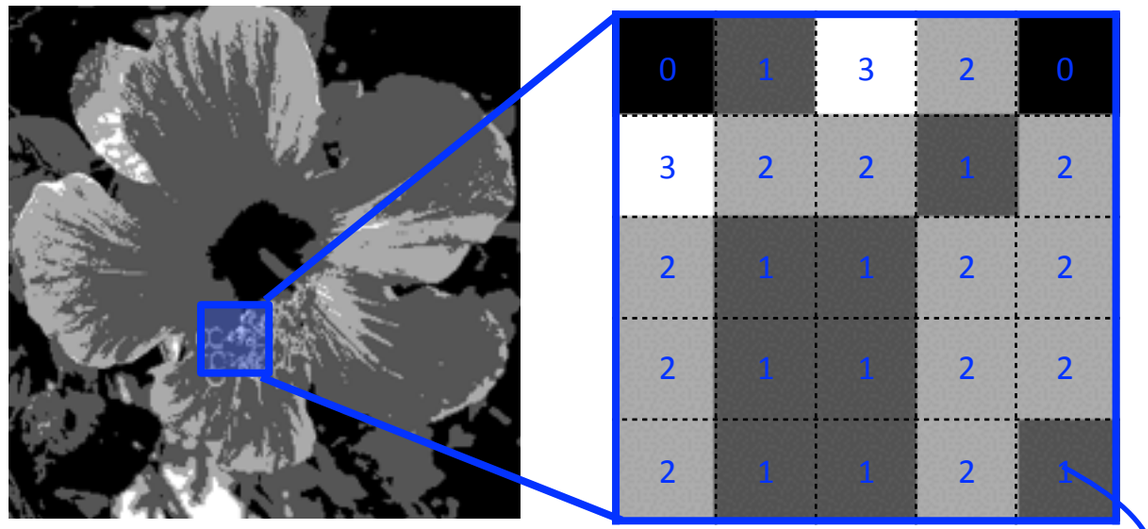
Codage des nombres $\{0, 1, 2, 3\}$:

$[0]_{10} = [0]_2 \quad \rightarrow \text{code} = 00$

$[1]_{10} = [1]_2 \quad \rightarrow \text{code} = 01$

$[2]_{10} = [10]_2 \rightarrow \text{code} = 10$

$[3]_{10} = [11]_2 \rightarrow \text{code} = 11$



codage

0	1	3	2	0	3	2	2	1	2	2	1	1	2	2	2	1	1	2	2	1	1	2	1
00	01	11	10	00	11	10	10	01	10	10	01	01	10	10	10	01	01	10	10	01	01	10	01

Code à envoyer: 000111100011101001101001011010100101101001011001

Codage uniforme : représentation binaire

Ecriture binaire (base 2):

$[\ldots a_3 a_2 a_1 a_0]_2$ représente $a_0 2^0 + a_1 2^1 + a_2 2^2 + a_3 2^3 + \ldots$

Exemple: $[110]_2 = [1 \times 4 + 1 \times 2 + 0 \times 1]_{10} = [6]_{10}$.

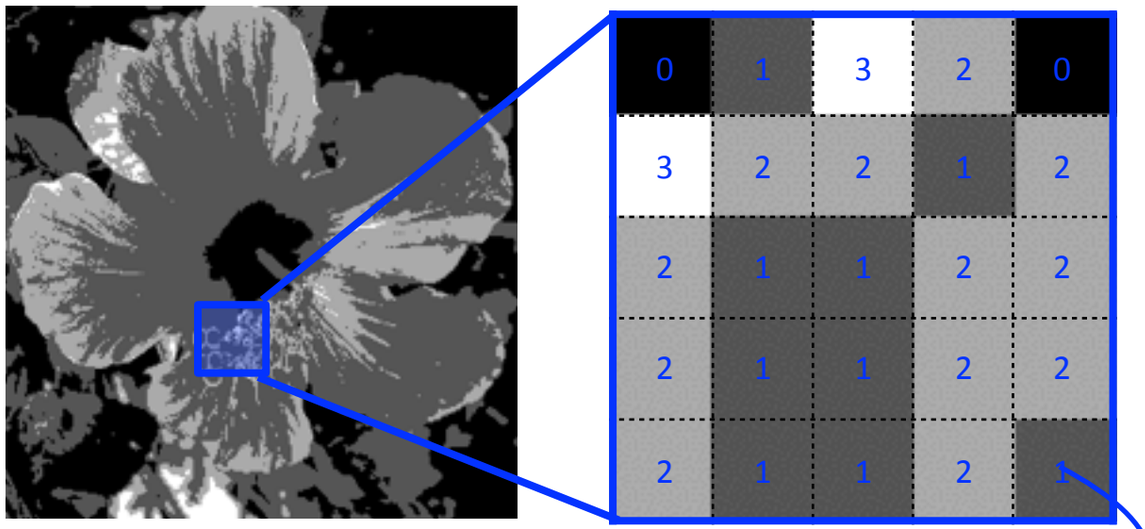
Codage des nombres $\{0, 1, 2, 3\}$:

$[0]_{10} = [0]_2 \quad \rightarrow \text{code} = 00$

$[1]_{10} = [1]_2 \quad \rightarrow \text{code} = 01$

$[2]_{10} = [10]_2 \rightarrow \text{code} = 10$

$[3]_{10} = [11]_2 \rightarrow \text{code} = 11$



codage

0	1	3	2	0	3	2	2	1	2	2	1	1	2	2	2	1	1	2	2	1	1	2	1
00	01	11	10	00	11	10	10	01	10	10	01	01	10	10	10	01	01	10	10	01	01	10	01

Code à envoyer: 000111100011101001101001011010100101101001011001

Chaque symbole 0 ou 1 correspond à 1 **bit**.

Codage uniforme : représentation binaire

Ecriture binaire (base 2):

$[\ldots a_3 a_2 a_1 a_0]_2$ représente $a_0 2^0 + a_1 2^1 + a_2 2^2 + a_3 2^3 + \ldots$

Exemple: $[110]_2 = [1 \times 4 + 1 \times 2 + 0 \times 1]_{10} = [6]_{10}$.

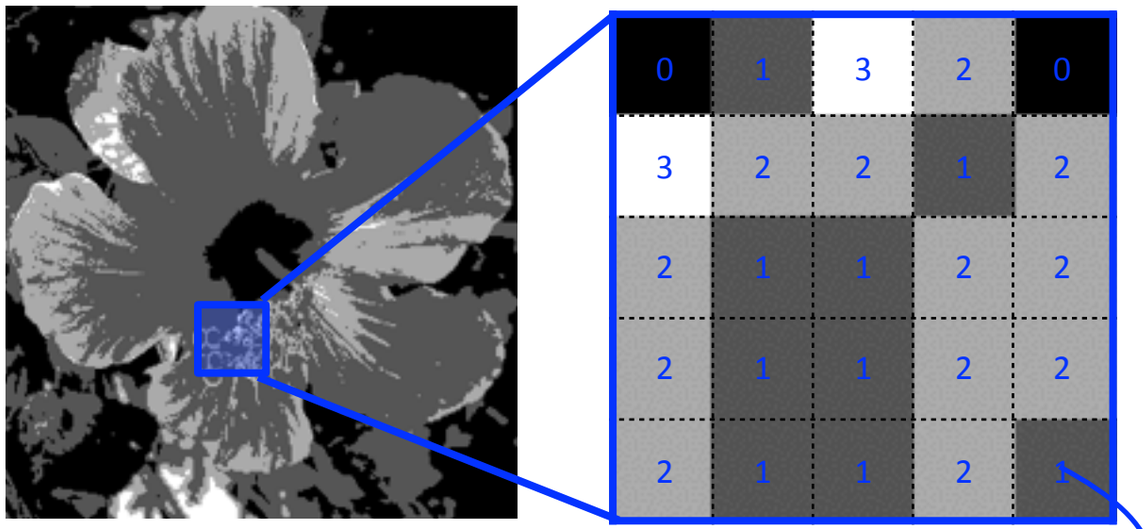
Codage des nombres $\{0, 1, 2, 3\}$:

$[0]_{10} = [0]_2 \quad \rightarrow \text{code} = 00$

$[1]_{10} = [1]_2 \quad \rightarrow \text{code} = 01$

$[2]_{10} = [10]_2 \rightarrow \text{code} = 10$

$[3]_{10} = [11]_2 \rightarrow \text{code} = 11$



codage

0	1	3	2	0	3	2	2	1	2	2	1	1	2	2	2	1	1	2	2	1	1	2	1
00	01	11	10	00	11	10	10	01	10	10	01	01	10	10	10	01	01	10	10	01	01	10	01

Code à envoyer: 000111100011101001101001011010100101101001011001

Chaque symbole 0 ou 1 correspond à 1 **bit**.

Décodage: découper en bloc de 2 bits.

Cas général et codage à longueur variable

CAS GÉNÉRAL

Logarithme en base 2: $\ell = \log_2(N)$ est l'unique réel tel que $2^\ell = N$.

Exemples: $\log_2(2) = 1$, $\log_2(4) = 2$, $\log_2(6) \approx 2.585\ldots$

Cas général et codage à longueur variable

CAS GÉNÉRAL

Logarithme en base 2: $\ell = \log_2(N)$ est l'unique réel tel que $2^\ell = N$.

Exemples: $\log_2(2) = 1,\ \log_2(4) = 2,\ \log_2(6) \approx 2.585\ldots$

N symboles $\{0, 1, \ldots, N-1\} \rightarrow$ besoin de $\lceil \log_2(N) \rceil$ bits/symbole.

Exemple: $N = 16 = 2^4 \rightarrow$ besoin de $\log_2(16) = 4$ bits/symbole.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111

Cas général et codage à longueur variable

CAS GÉNÉRAL

Logarithme en base 2: $\ell = \log_2(N)$ est l'unique réel tel que $2^\ell = N$.

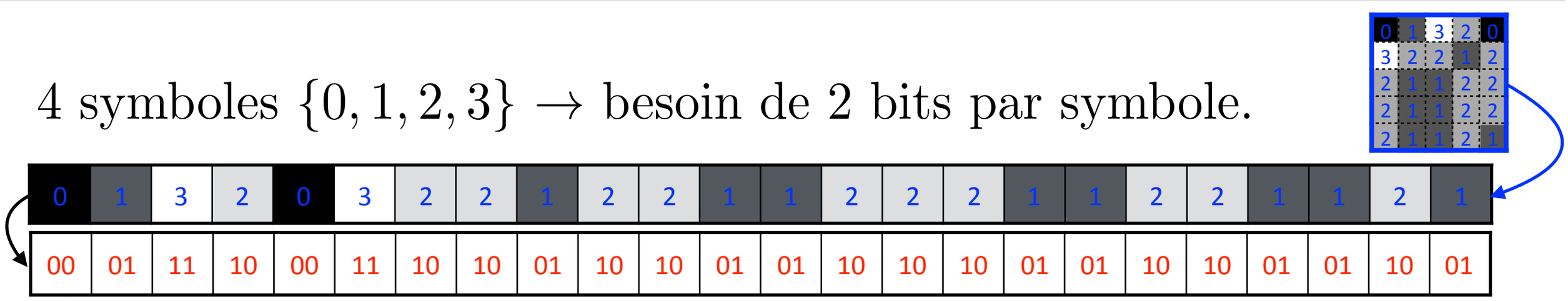
Exemples: $\log_2(2) = 1,\ \log_2(4) = 2,\ \log_2(6) \approx 2.585\ldots$

N symboles $\{0, 1, \ldots, N-1\} \rightarrow$ besoin de $\lceil \log_2(N) \rceil$ bits/symbole.

Exemple: $N = 16 = 2^4 \rightarrow$ besoin de $\log_2(16) = 4$ bits/symbole.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111

4 symboles $\{0, 1, 2, 3\} \rightarrow$ besoin de 2 bits par symbole.



Code à envoyer: 00011110001110100110100101101010010110100101011001 $\rightarrow$ 50 bits.

Cas général et codage à longueur variable

CAS GÉNÉRAL

Logarithme en base 2: $\ell = \log_2(N)$ est l'unique réel tel que $2^\ell = N$.

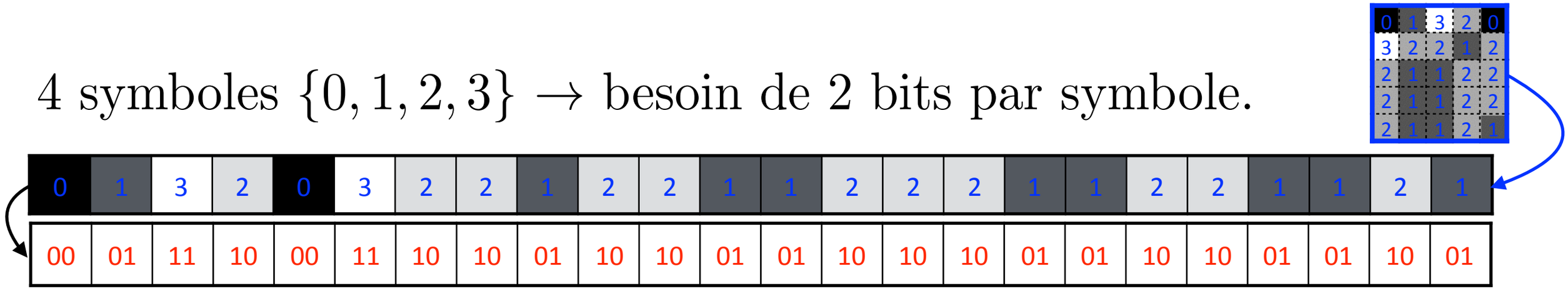
Exemples: $\log_2(2) = 1,\ \log_2(4) = 2,\ \log_2(6) \approx 2.585\ldots$

N symboles $\{0, 1, \ldots, N-1\} \rightarrow$ besoin de $\lceil \log_2(N) \rceil$ bits/symbole.

Exemple: $N = 16 = 2^4 \rightarrow$ besoin de $\log_2(16) = 4$ bits/symbole.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111

4 symboles $\{0, 1, 2, 3\} \rightarrow$ besoin de 2 bits par symbole.



Code à envoyer: 00011110001110100110100101101010010110100101011001 $\rightarrow$ 50 bits.

Peut-on faire mieux?

Cas général et codage à longueur variable

CAS GÉNÉRAL

Logarithme en base 2: $\ell = \log_2(N)$ est l'unique réel tel que $2^\ell = N$.

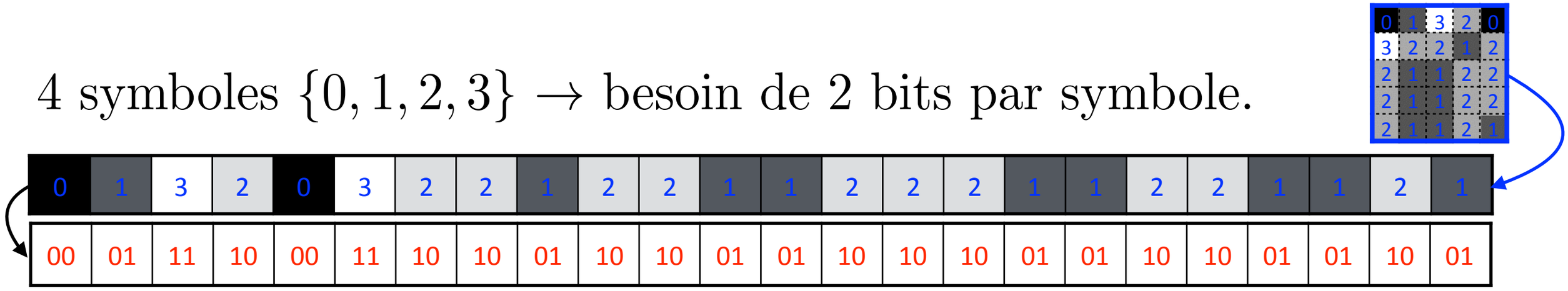
Exemples: $\log_2(2) = 1,\ \log_2(4) = 2,\ \log_2(6) \approx 2.585\ldots$

N symboles $\{0, 1, \ldots, N-1\} \rightarrow$ besoin de $\lceil \log_2(N) \rceil$ bits/symbole.

Exemple: $N = 16 = 2^4 \rightarrow$ besoin de $\log_2(16) = 4$ bits/symbole.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111

4 symboles $\{0, 1, 2, 3\} \rightarrow$ besoin de 2 bits par symbole.



Code à envoyer: 00011110001110100110100101101010010110100101011001 $\rightarrow$ 50 bits.

Peut-on faire mieux? $0 \rightarrow 001,\quad 1 \rightarrow 01,\quad 2 \rightarrow 1,\quad 3 \rightarrow 000$

Cas général et codage à longueur variable

CAS GÉNÉRAL

Logarithme en base 2: $\ell = \log_2(N)$ est l'unique réel tel que $2^\ell = N$.

Exemples: $\log_2(2) = 1,\ \log_2(4) = 2,\ \log_2(6) \approx 2.585\ldots$

N symboles $\{0, 1, \ldots, N-1\} \rightarrow$ besoin de $\lceil \log_2(N) \rceil$ bits/symbole.

Exemple: $N = 16 = 2^4 \rightarrow$ besoin de $\log_2(16) = 4$ bits/symbole.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111

4 symboles $\{0, 1, 2, 3\} \rightarrow$ besoin de 2 bits par symbole.

0	1	3	2	0	3	2	2	1	2	2	1	1	2	2	2	1	1	2	2	1	1	2	1
00	01	11	10	00	11	10	10	01	10	10	01	01	10	10	10	01	01	10	10	01	01	10	01

Code à envoyer: 00011110001110100110100101101010010110100 1011001 $\rightarrow$ 50 bits.

Peut-on faire mieux? $0 \rightarrow 001,\quad 1 \rightarrow 01,\quad 2 \rightarrow 1,\quad 3 \rightarrow 000$

001	01	000	1	001	000	1	1	01	1	1	01	01	1	1	1	01	01	1	1	01	01	1	01

Code à envoyer: 00101000100100011011101011110101110101101 $\rightarrow$ 41 bits.

Décodage et arbres binaires

Code reçu : 00101000100100011011101011110101110101101

Décodage et arbres binaires

Code reçu : 00101000100100011011101011110101110101101

Décodage non-ambiguë : aucun mot du code n'est le début d'un autre.

Code préfixe.

Décodage et arbres binaires

Code reçu : 00101000100100011011101011110101110101101

Décodage non-ambiguë : aucun mot du code n'est le début d'un autre.

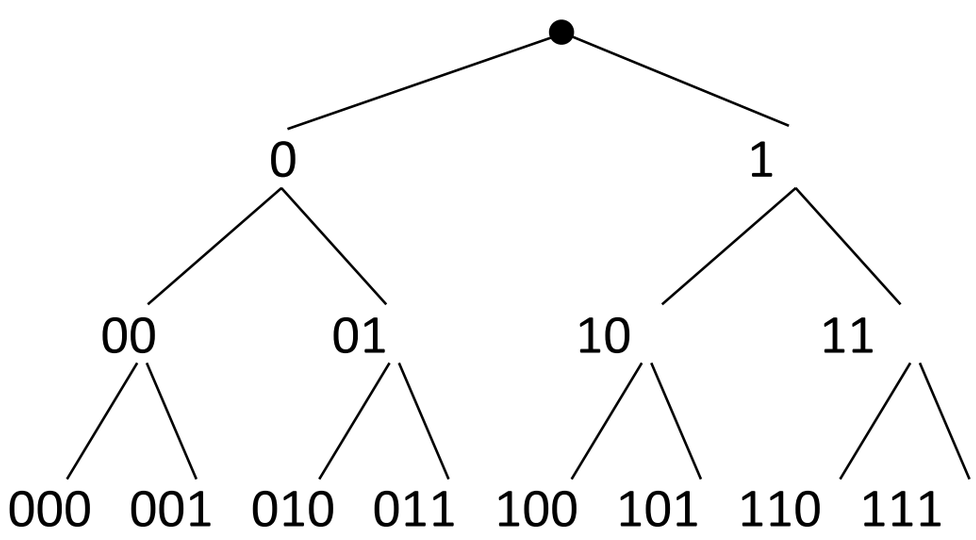
Code préfixe. → Se représente sous forme d'un arbre binaire.

Décodage et arbres binaires

Code reçu : 00101000100100011011101011110101110101101

Décodage non-ambiguë : aucun mot du code n'est le début d'un autre.

Code préfixe. → Se représente sous forme d'un arbre binaire.

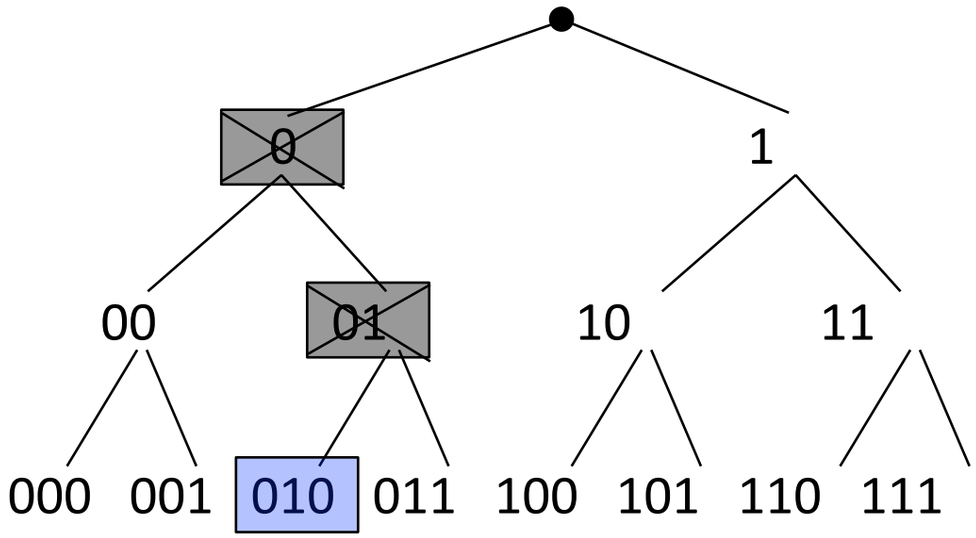


Décodage et arbres binaires

Code reçu : 00101000100100011011101011110101110101101

Décodage non-ambiguë : aucun mot du code n'est le début d'un autre.

Code préfixe. → Se représente sous forme d'un arbre binaire.



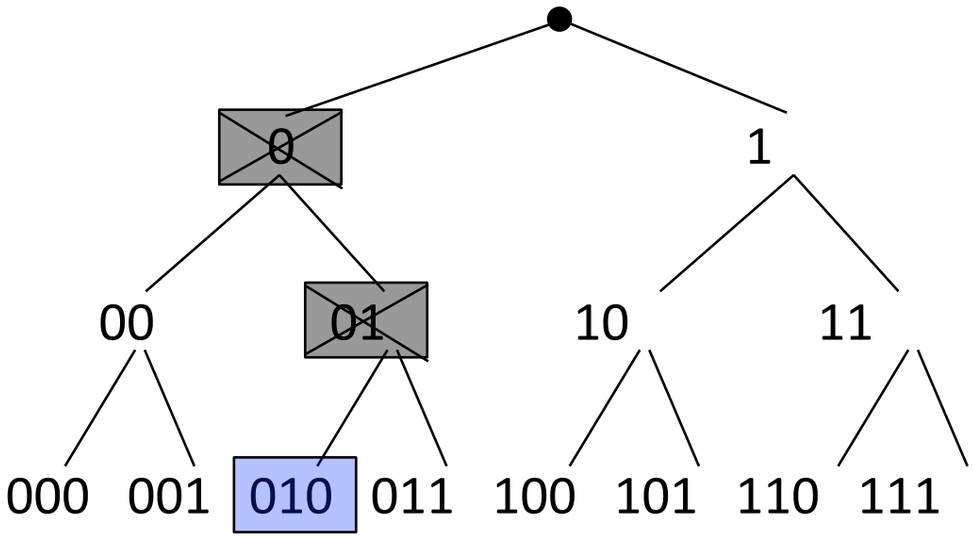
Décodage et arbres binaires

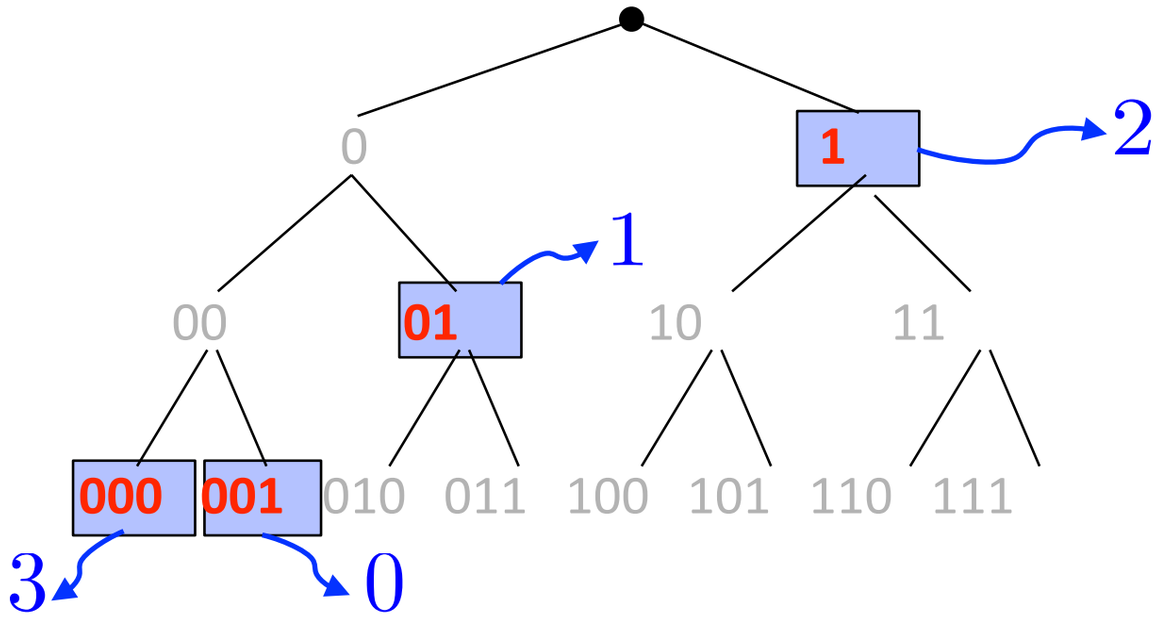
Code reçu : 00101000100100011011101011110101110101101

Décodage non-ambiguë : aucun mot du code n'est le début d'un autre.

Code préfixe. → Se représente sous forme d'un arbre binaire.

$0 \to 001, \quad 1 \to 01, \quad 2 \to 1, \quad 3 \to 000$





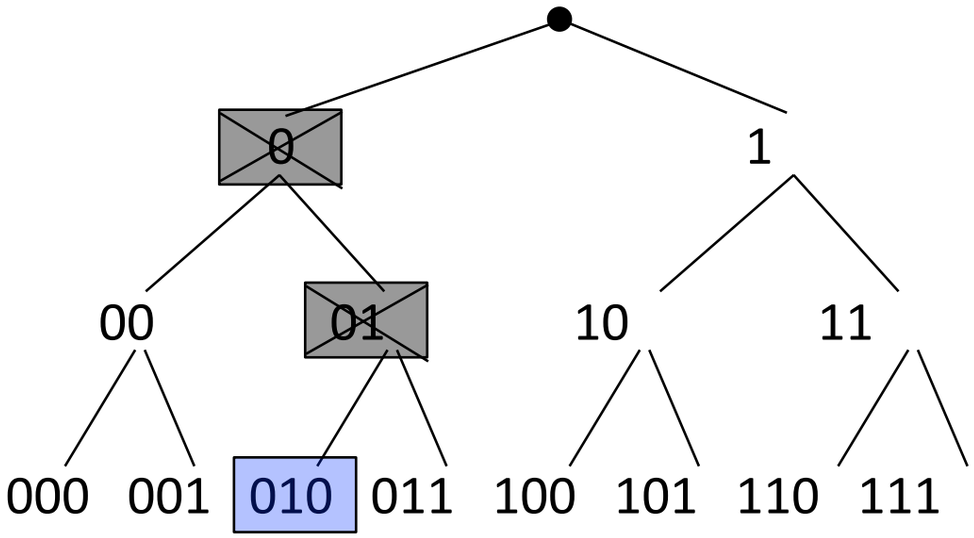
Décodage et arbres binaires

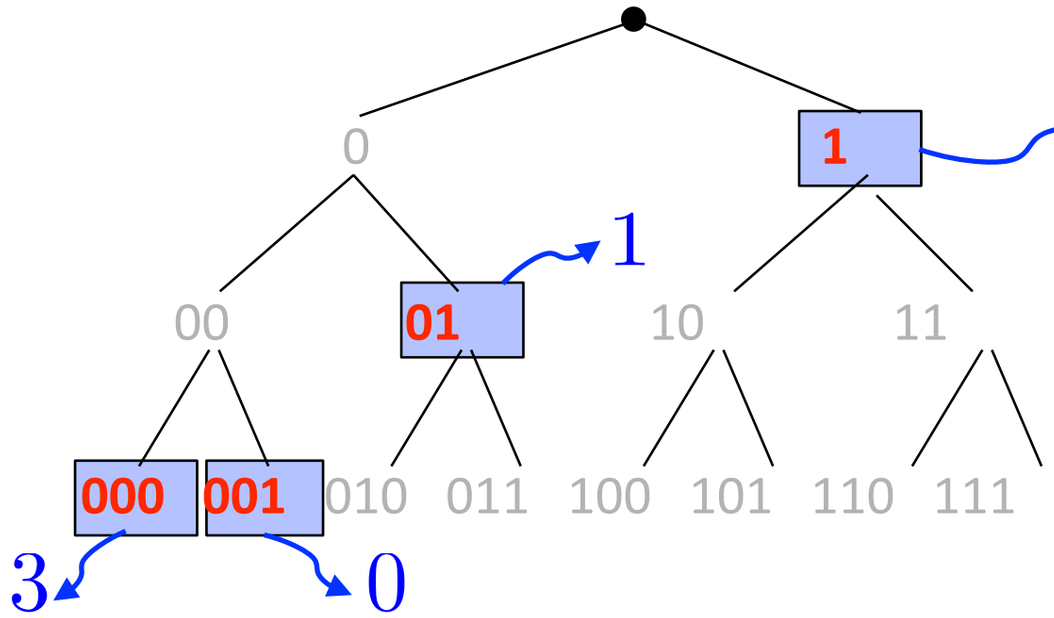
Code reçu : 00101000100100011011101011110101110101101

Décodage non-ambiguë : aucun mot du code n'est le début d'un autre.

Code préfixe. → Se représente sous forme d'un arbre binaire.

$0 \to 001, \quad 1 \to 01, \quad 2 \to 1, \quad 3 \to 000$





Décodage: parcours de l'arbre.

001010001001000110111010111101011101011 01

→ décode 0

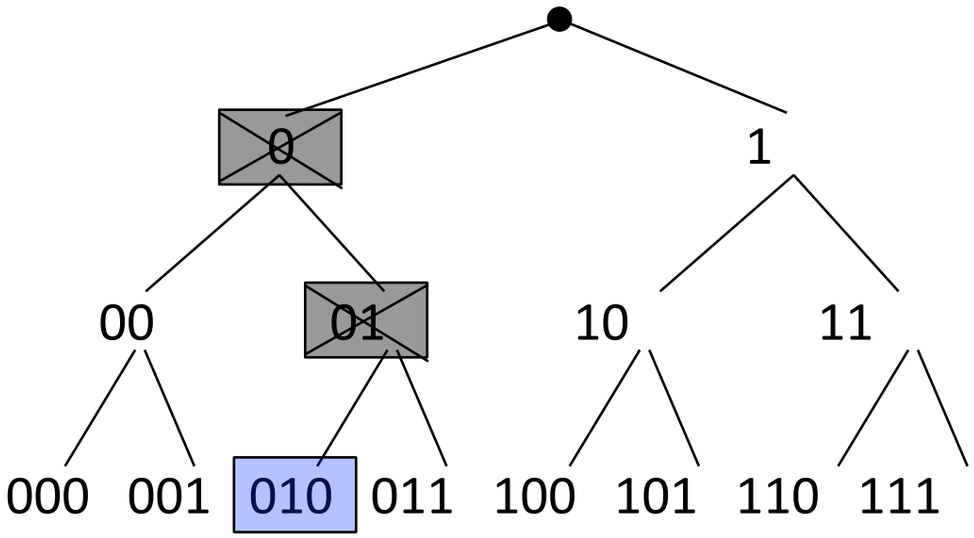
Décodage et arbres binaires

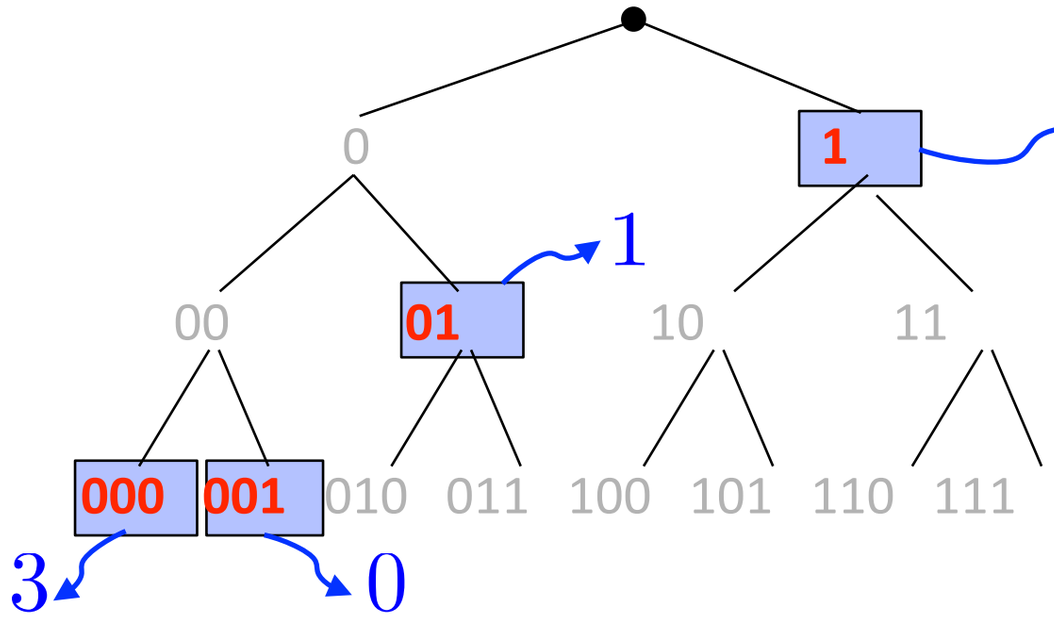
Code reçu : 00101000100100011011101011110101110101101

Décodage non-ambiguë : aucun mot du code n'est le début d'un autre.

Code préfixe. → Se représente sous forme d'un arbre binaire.

$0 \to 001, \quad 1 \to 01, \quad 2 \to 1, \quad 3 \to 000$





Décodage: parcours de l'arbre.

001010001001000110111010111101011101011 01 → décode 0

0 01000100100011011101011110101110101101 → décode 1

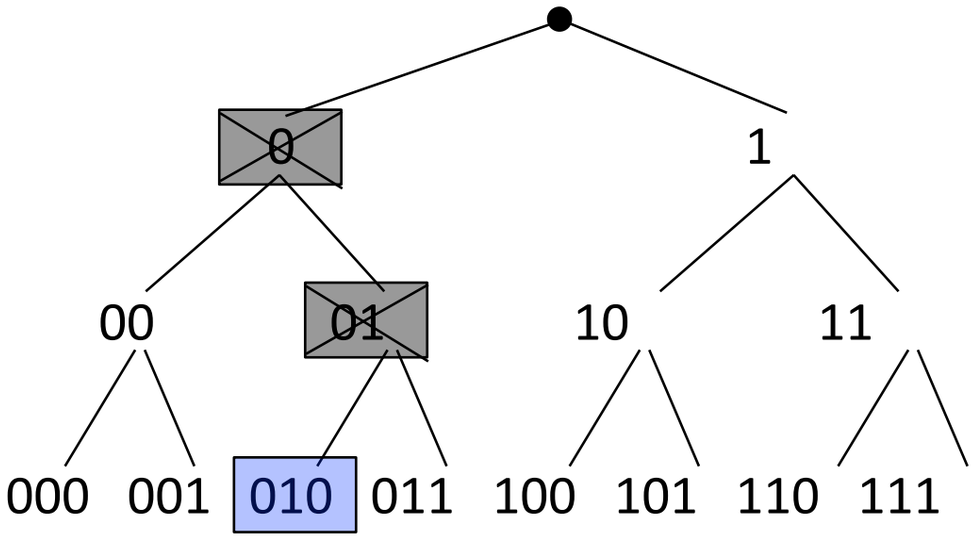
Décodage et arbres binaires

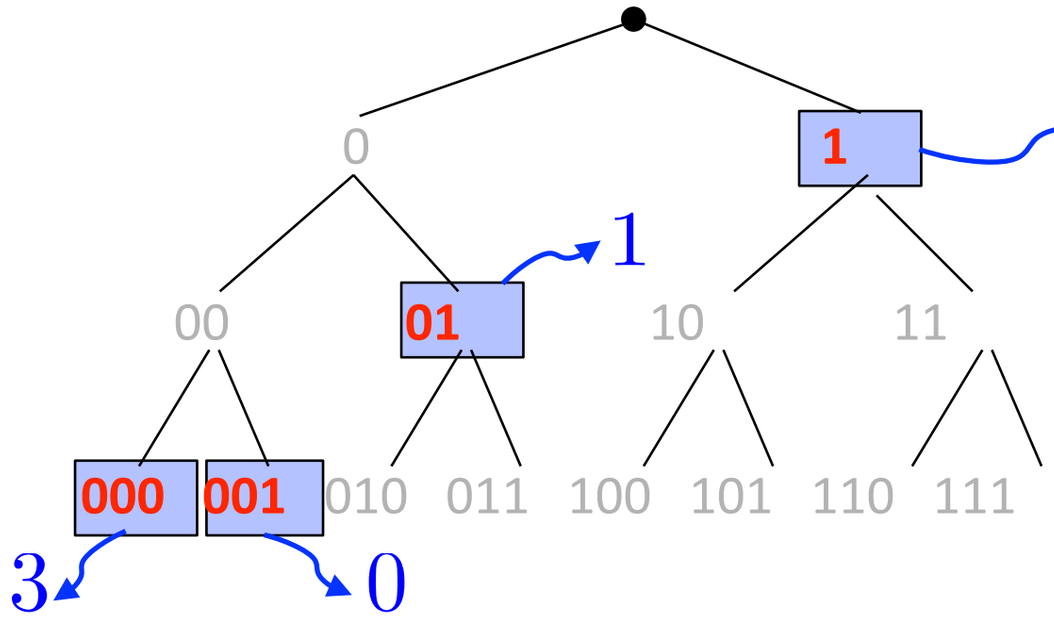
Code reçu : 00101000100100011011101011110101110101101

Décodage non-ambiguë : aucun mot du code n'est le début d'un autre.

Code préfixe. → Se représente sous forme d'un arbre binaire.

$0 \to 001, \quad 1 \to 01, \quad 2 \to 1, \quad 3 \to 000$





Décodage: parcours de l'arbre.

001010001001000110111010111101011101011 01 → décode 0

0 01000100100011011101011110101110101101 → décode 1

0 1 000100100011011101011110101110101101 → décode 3

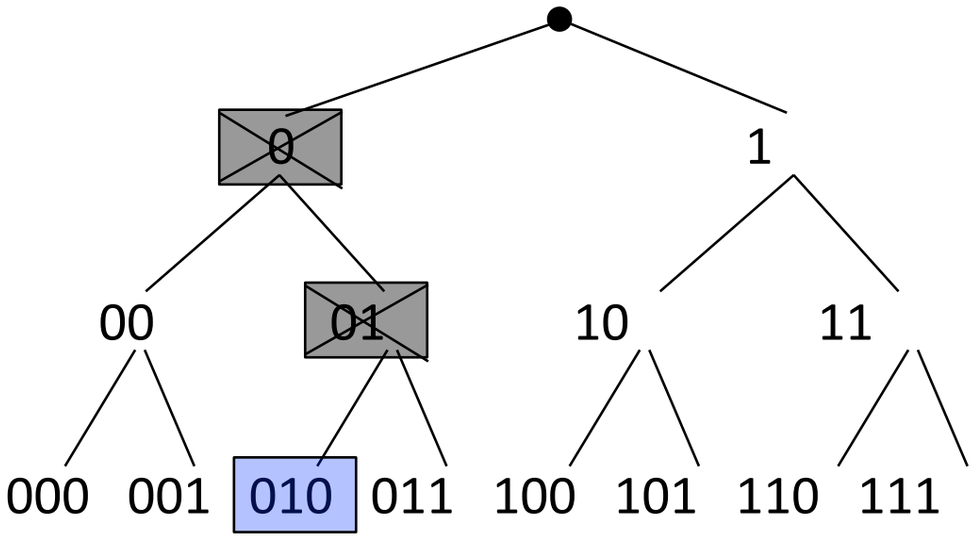
Décodage et arbres binaires

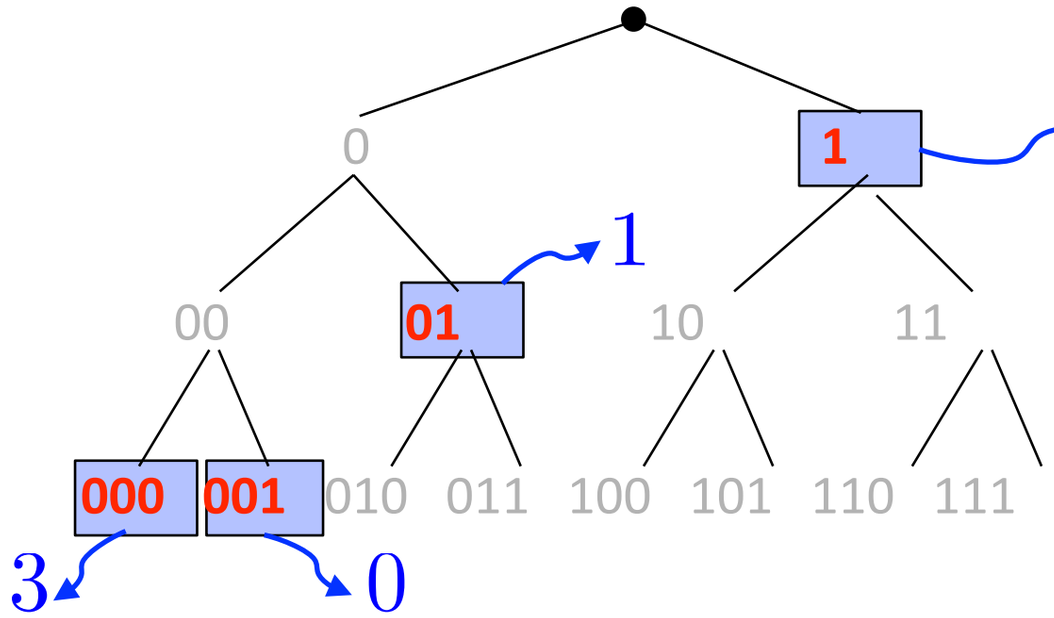
Code reçu : 00101000100100011011101011110101110101101

Décodage non-ambiguë : aucun mot du code n'est le début d'un autre.

Code préfixe. → Se représente sous forme d'un arbre binaire.

$0 \to 001, \quad 1 \to 01, \quad 2 \to 1, \quad 3 \to 000$





Décodage: parcours de l'arbre.

001 0100010010001101110101111010111010110 1 → décode 0

0 01 000100100011011101011110101110101101 → décode 1

0 1 000 100100011011101011110101110101101 → décode 3

0 1 3 1001000110111010111101011101011 01 → décode 2 ...

Entropie

Comment quantifier la quantité d'information d'une suite de symboles ?

Entropie

Comment quantifier la quantité d'information d'une suite de symboles ?

p_i = fréquence d'apparition du symbole i.

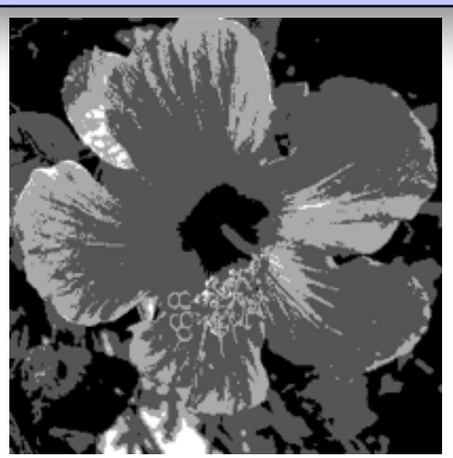
$$p_0 + p_1 + p_2 + p_3 = 1$$

Entropie

Comment quantifier la quantité d'information d'une suite de symboles ?

$p_i =$ fréquence d'apparition du symbole i.

$p_0 + p_1 + p_2 + p_3 = 1$



$256 \times 256 = 2^{16}$ pixels:

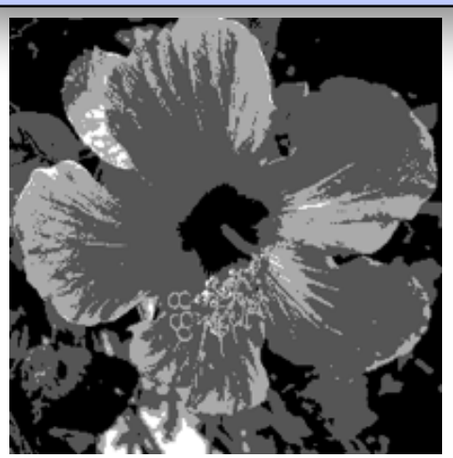
17432 pixels noirs $\rightarrow p_0 = \frac{17432}{2^{16}} \approx 0.27.$

Entropie

Comment quantifier la quantité d'information d'une suite de symboles ?

p_i = fréquence d'apparition du symbole i.

$p_0 + p_1 + p_2 + p_3 = 1$



$256 \times 256 = 2^{16}$ pixels:

17432 pixels noirs $\rightarrow p_0 = \frac{17432}{2^{16}} \approx 0.27.$

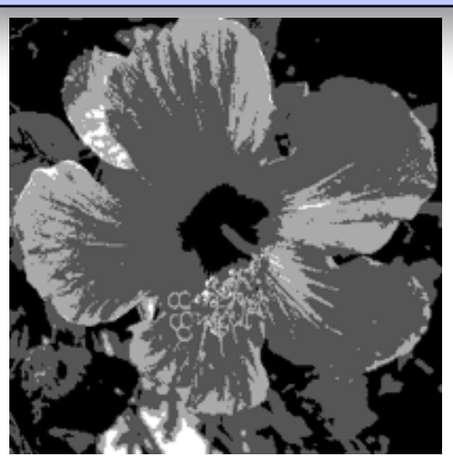
923 pixels blanc $\rightarrow p_3 = \frac{923}{2^{16}} \approx 0.01.$

Entropie

Comment quantifier la quantité d'information d'une suite de symboles ?

p_i = fréquence d'apparition du symbole i.

$$p_0 + p_1 + p_2 + p_3 = 1$$



$256 \times 256 = 2^{16}$ pixels:

17432 pixels noirs $\rightarrow p_0 = \frac{17432}{2^{16}} \approx 0.27.$

923 pixels blanc $\rightarrow p_3 = \frac{923}{2^{16}} \approx 0.01.$

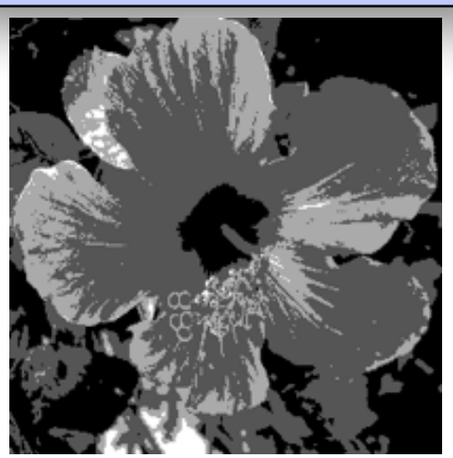
$p_1 \approx 0.53, \quad p_2 \approx 0.19$

Entropie

Comment quantifier la quantité d'information d'une suite de symboles ?

p_i = fréquence d'apparition du symbole i.

$$p_0 + p_1 + p_2 + p_3 = 1$$

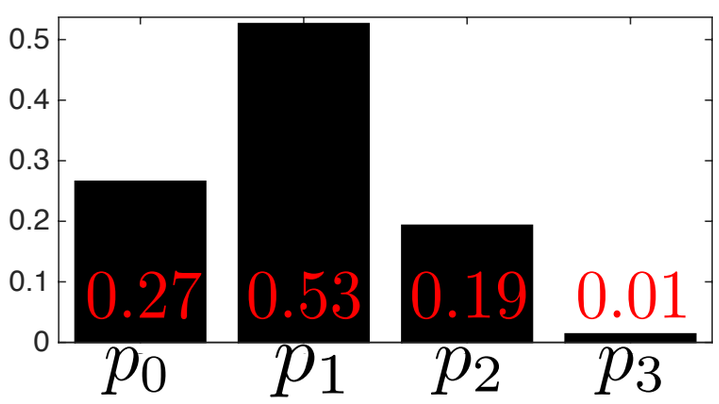


$256 \times 256 = 2^{16}$ pixels:

17432 pixels noirs $\rightarrow p_0 = \frac{17432}{2^{16}} \approx 0.27.$

923 pixels blanc $\rightarrow p_3 = \frac{923}{2^{16}} \approx 0.01.$

$p_1 \approx 0.53, \quad p_2 \approx 0.19$

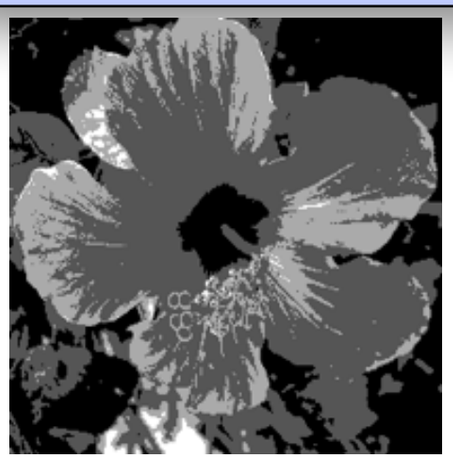


Entropie

Comment quantifier la quantité d'information d'une suite de symboles ?

p_i = fréquence d'apparition du symbole i.

$$p_0 + p_1 + p_2 + p_3 = 1$$

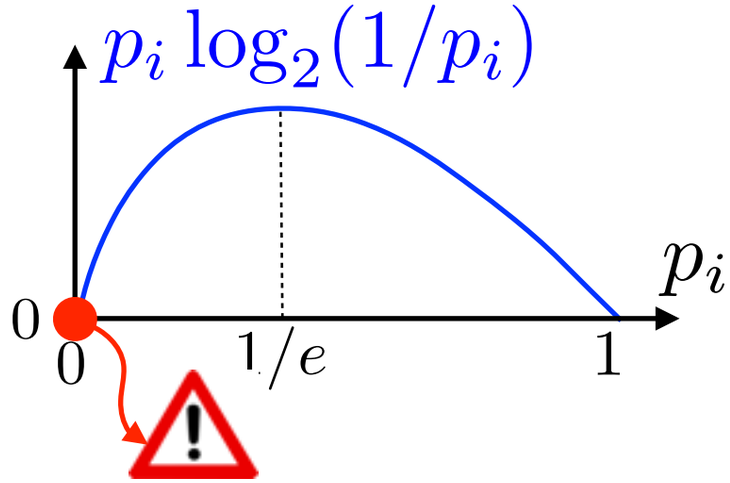


$256 \times 256 = 2^{16}$ pixels:

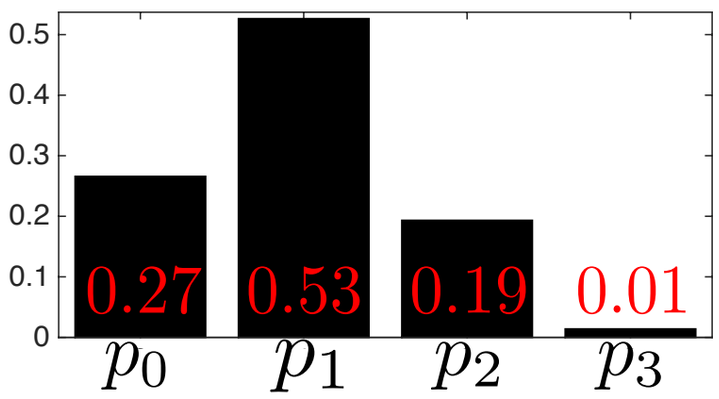
17432 pixels noirs $\rightarrow p_0 = \frac{17432}{2^{16}} \approx 0.27.$

923 pixels blanc $\rightarrow p_3 = \frac{923}{2^{16}} \approx 0.01.$

$p_1 \approx 0.53, \quad p_2 \approx 0.19$



$$\textit{Entropie :} \quad H(p) \stackrel{\text{def.}}{=} \sum_i p_i \log_2 \left(\frac{1}{p_i}\right)$$

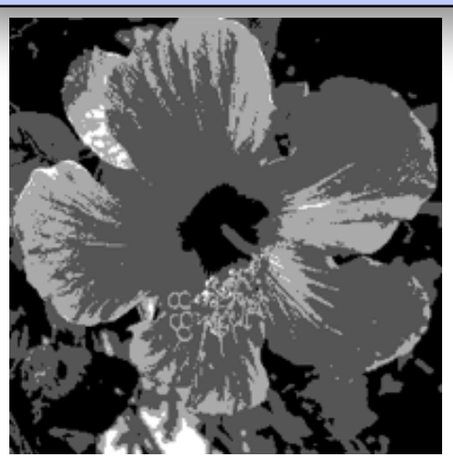


Entropie

Comment quantifier la quantité d'information d'une suite de symboles ?

p_i = fréquence d'apparition du symbole i.

$$p_0 + p_1 + p_2 + p_3 = 1$$

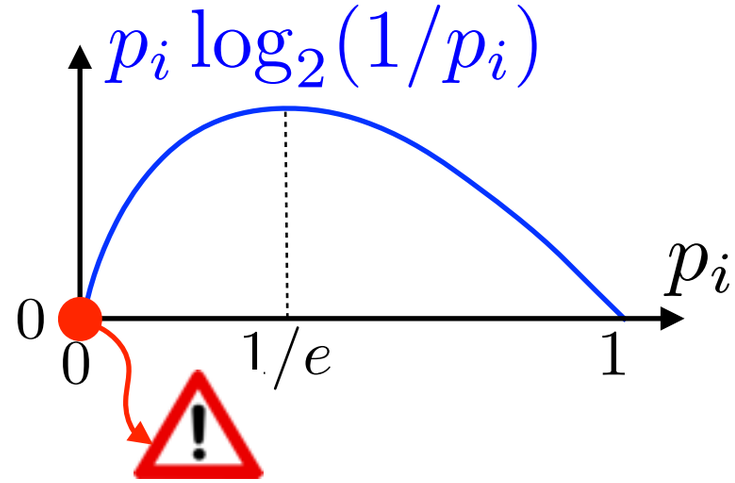


$256 \times 256 = 2^{16}$ pixels:

17432 pixels noirs $\to p_0 = \frac{17432}{2^{16}} \approx 0.27.$

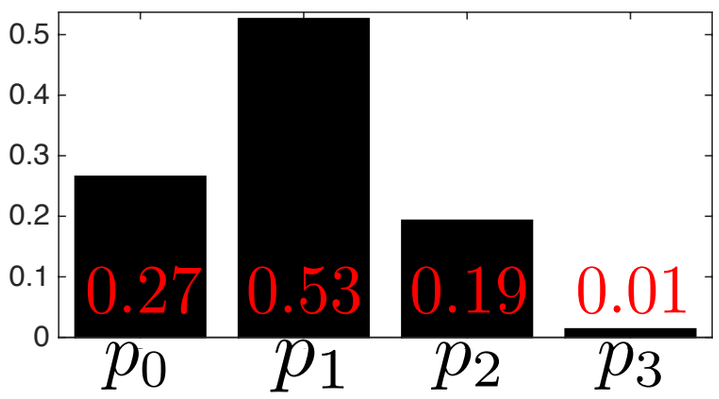
923 pixels blanc $\to p_3 = \frac{923}{2^{16}} \approx 0.01.$

$p_1 \approx 0.53, \quad p_2 \approx 0.19$



$$\textit{Entropie :} \quad H(p) \stackrel{\text{def.}}{=} \sum_i p_i \log_2\left(\frac{1}{p_i}\right)$$

Propriété: $0 \leqslant H(p) \leqslant \log_2(N).$

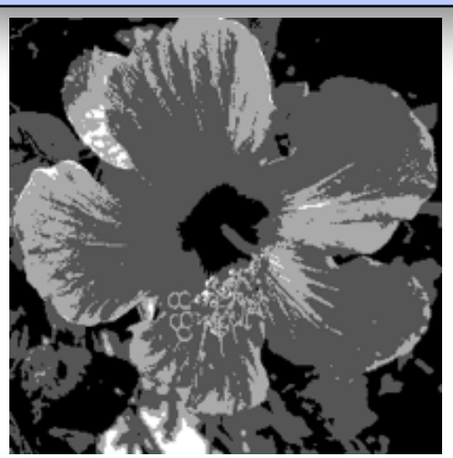


Entropie

Comment quantifier la quantité d'information d'une suite de symboles ?

p_i = fréquence d'apparition du symbole i.

$$p_0 + p_1 + p_2 + p_3 = 1$$

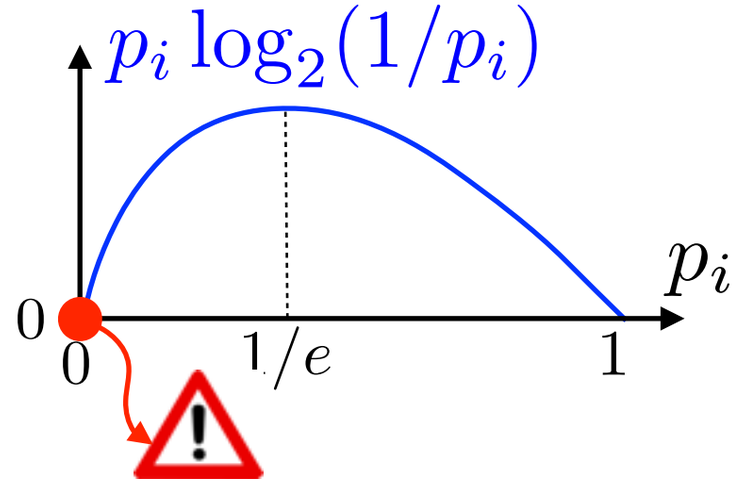


$256 \times 256 = 2^{16}$ pixels:

17432 pixels noirs $\to p_0 = \frac{17432}{2^{16}} \approx 0.27.$

923 pixels blanc $\to p_3 = \frac{923}{2^{16}} \approx 0.01.$

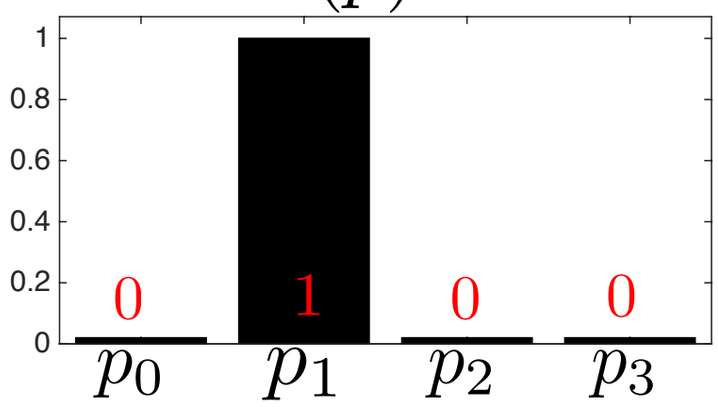
$p_1 \approx 0.53, \quad p_2 \approx 0.19$

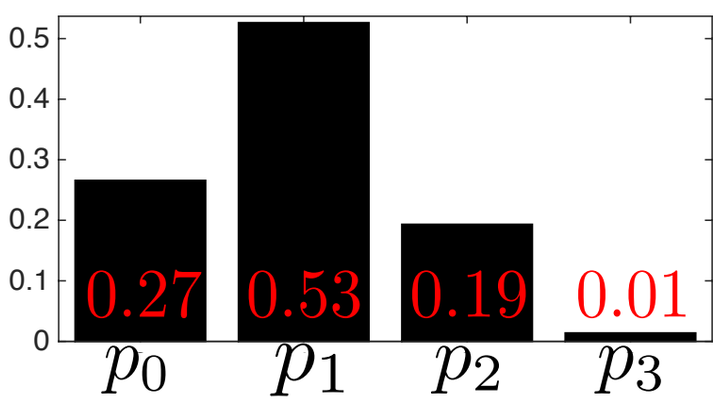


$$\textit{Entropie :} \quad H(p) \stackrel{\text{def.}}{=} \sum_i p_i \log_2\left(\frac{1}{p_i}\right)$$

Propriété: $0 \leqslant H(p) \leqslant \log_2(N).$

$H(p) = 0$



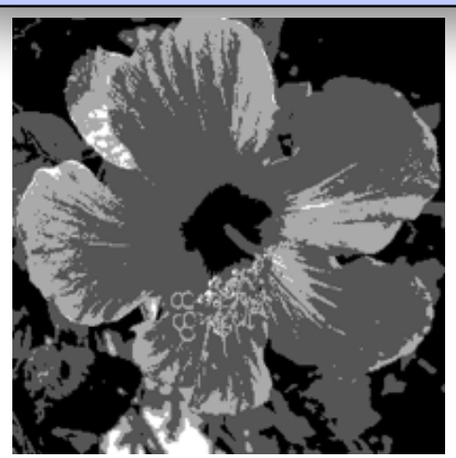


Entropie

Comment quantifier la quantité d'information d'une suite de symboles ?

p_i = fréquence d'apparition du symbole i.

$$p_0 + p_1 + p_2 + p_3 = 1$$

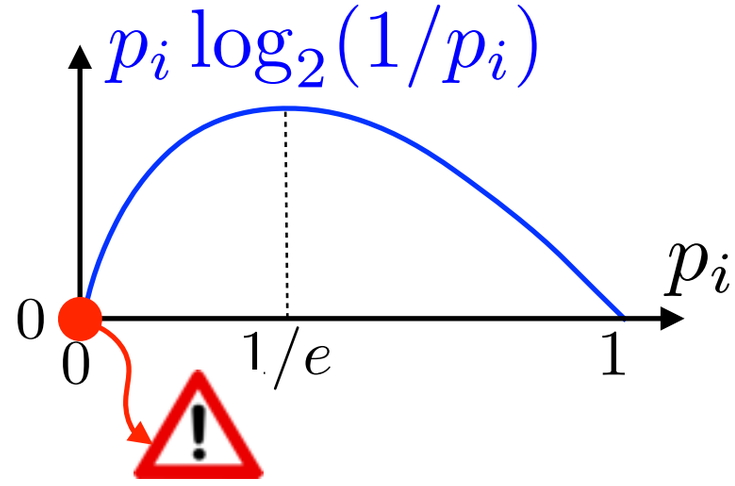


$256 \times 256 = 2^{16}$ pixels:

17432 pixels noirs $\to p_0 = \frac{17432}{2^{16}} \approx 0.27.$

923 pixels blanc $\to p_3 = \frac{923}{2^{16}} \approx 0.01.$

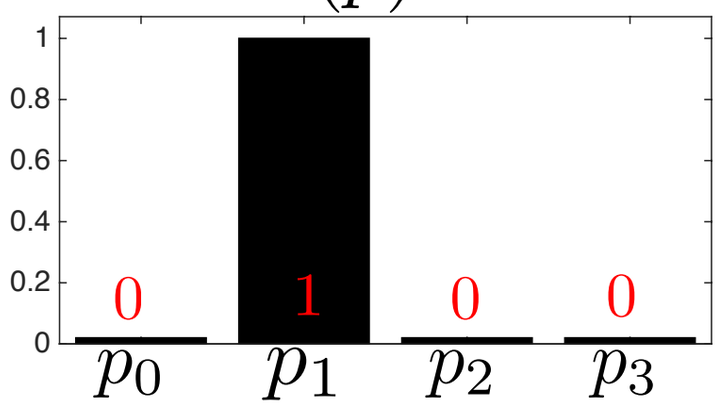
$p_1 \approx 0.53, \quad p_2 \approx 0.19$

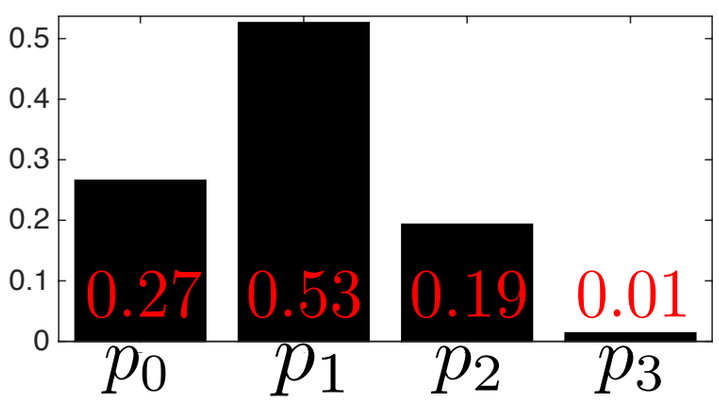


$$\textit{Entropie :} \quad H(p) \stackrel{\text{def.}}{=} \sum_i p_i \log_2\left(\frac{1}{p_i}\right)$$

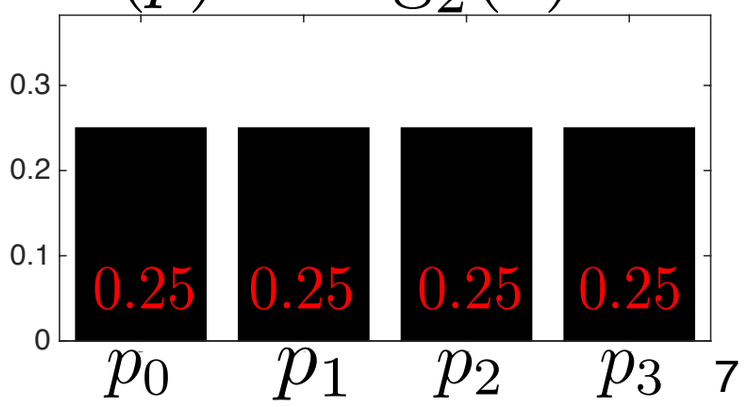
Propriété: $0 \leqslant H(p) \leqslant \log_2(N).$

$H(p) = 0$





$H(p) = \log_2(4) = 2$

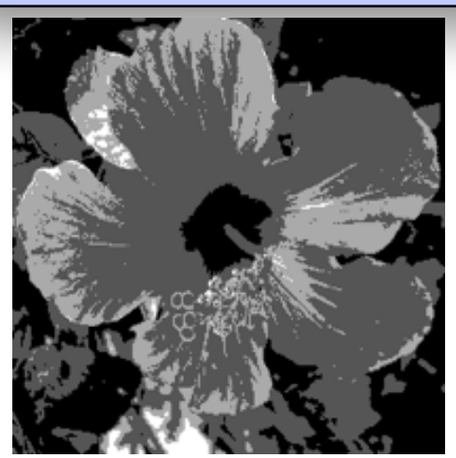


Entropie

Comment quantifier la quantité d'information d'une suite de symboles ?

p_i = fréquence d'apparition du symbole i.

$$p_0 + p_1 + p_2 + p_3 = 1$$

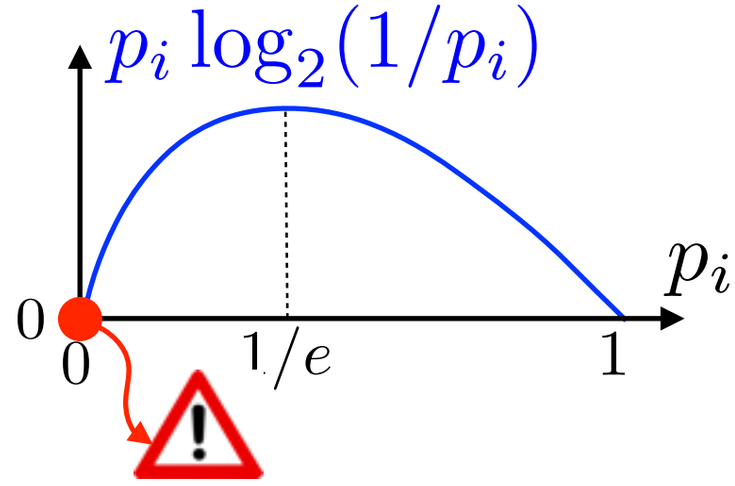


$256 \times 256 = 2^{16}$ pixels:

17432 pixels noirs $\to p_0 = \frac{17432}{2^{16}} \approx 0.27.$

923 pixels blanc $\to p_3 = \frac{923}{2^{16}} \approx 0.01.$

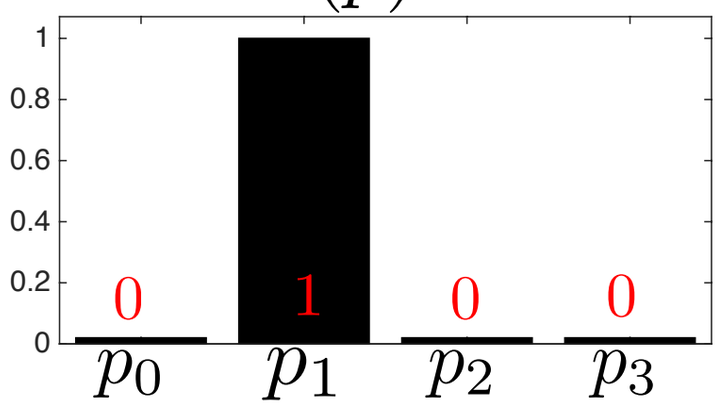
$p_1 \approx 0.53, \quad p_2 \approx 0.19$



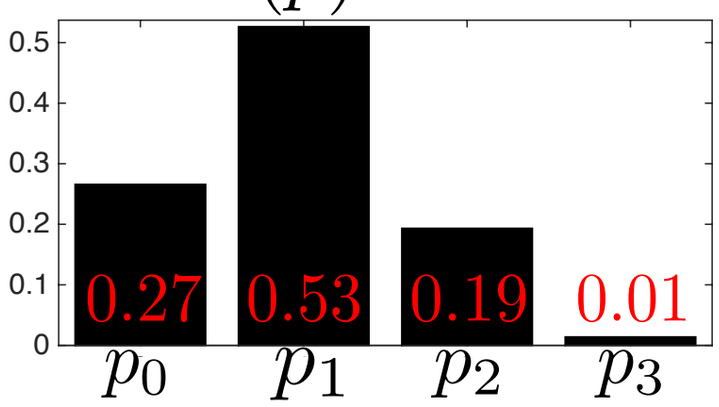
$$\textit{Entropie :} \quad H(p) \stackrel{\text{def.}}{=} \sum_i p_i \log_2\left(\frac{1}{p_i}\right)$$

Propriété: $0 \leqslant H(p) \leqslant \log_2(N).$

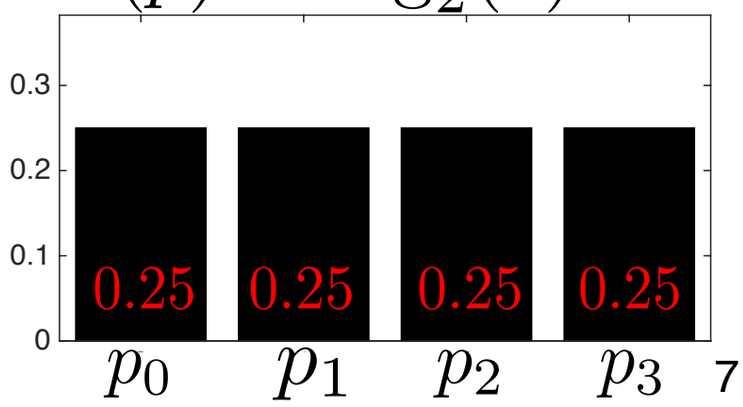
$H(p) = 0$



$H(p) \approx 1.54$



$H(p) = \log_2(4) = 2$



Borne de Shannon

Théorème de Shannon:

si les symboles sont tirés au **hasard** selon la loi $(p_i)_i$

le nombre **moyen** de bits d'un code **préfixe** est supérieur à $H(p)$.

Borne de Shannon

Théorème de Shannon:
si les symboles sont tirés au **hasard** selon la loi $(p_i)_i$
le nombre **moyen** de bits d'un code **préfixe** est supérieur à $H(p)$.

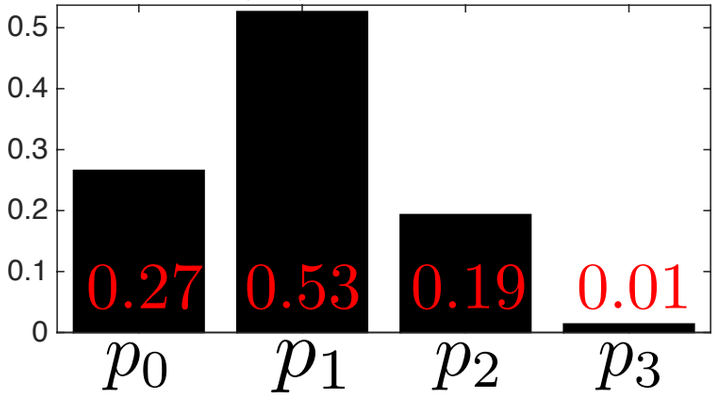
$H(p)$ faible $\Leftrightarrow$ peu d'information $\Leftrightarrow$ facile à coder.

Borne de Shannon

Théorème de Shannon:
si les symboles sont tirés au **hasard** selon la loi $(p_i)_i$
le nombre **moyen** de bits d'un code **préfixe** est supérieur à $H(p)$.

$H(p)$ faible $\Leftrightarrow$ peu d'information $\Leftrightarrow$ facile à coder.

$H(p) \approx 1.54$



Borne de Shannon

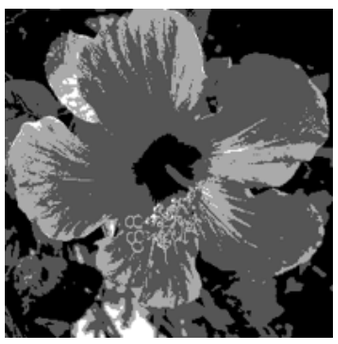
Théorème de Shannon:

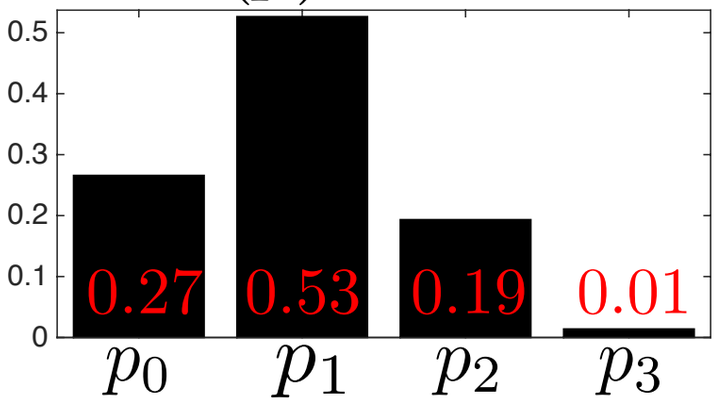
si les symboles sont tirés au **hasard** selon la loi $(p_i)_i$

le nombre **moyen** de bits d'un code **préfixe** est supérieur à $H(p)$.

$H(p)$ faible $\Leftrightarrow$ peu d'information $\Leftrightarrow$ facile à coder.

$H(p) \approx 1.54$





Codage préfixe:

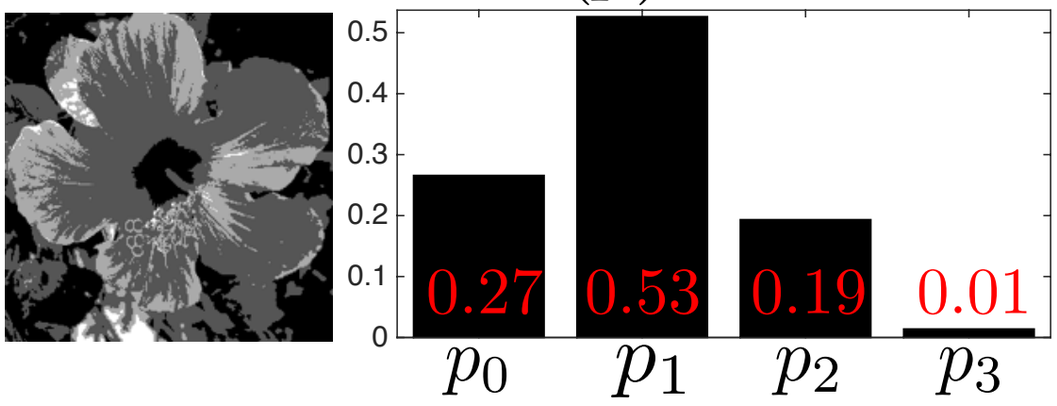
$0 \to c_0, \quad 1 \to c_1, \quad 2 \to c_2, \quad 3 \to c_3$

Borne de Shannon

Théorème de Shannon:
si les symboles sont tirés au **hasard** selon la loi $(p_i)_i$
le nombre **moyen** de bits d'un code **préfixe** est supérieur à $H(p)$.

$H(p)$ faible $\Leftrightarrow$ peu d'information $\Leftrightarrow$ facile à coder.

$H(p) \approx 1.54$



Codage préfixe:
$0 \to c_0, \quad 1 \to c_1, \quad 2 \to c_2, \quad 3 \to c_3$

$$\text{Long. moy.} = p_0 \times \text{Long}(c_0) + p_1 \times \text{Long}(c_1) + p_2 \times \text{Long}(c_2) + p_3 \times \text{Long}(c_3)$$

Borne de Shannon

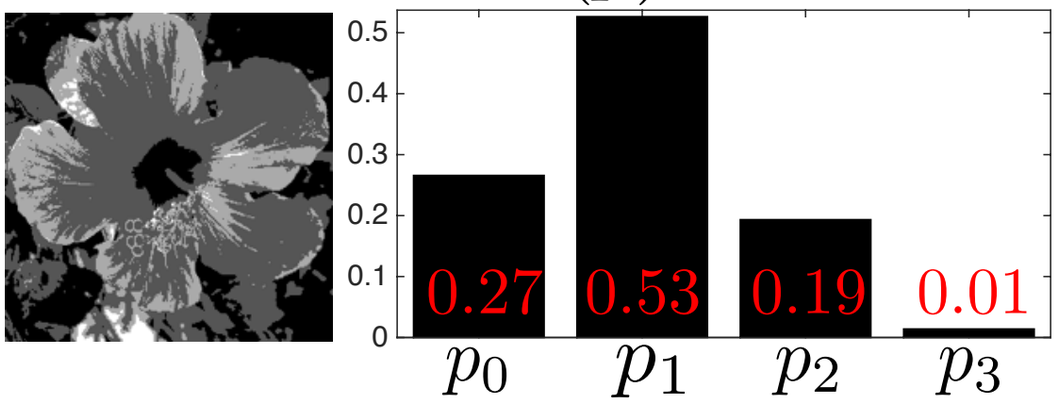
Théorème de Shannon:

si les symboles sont tirés au **hasard** selon la loi $(p_i)_i$

le nombre **moyen** de bits d'un code **préfixe** est supérieur à $H(p)$.

$H(p)$ faible $\Leftrightarrow$ peu d'information $\Leftrightarrow$ facile à coder.

$H(p) \approx 1.54$



Codage préfixe:

$0 \to c_0, \quad 1 \to c_1, \quad 2 \to c_2, \quad 3 \to c_3$

Exemple:

$0 \to 10, \quad 1 \to 0, \quad 2 \to 110, \quad 3 \to 111$

$$\text{Long. moy.} = p_0 \times \text{Long}(c_0) + p_1 \times \text{Long}(c_1) + p_2 \times \text{Long}(c_2) + p_3 \times \text{Long}(c_3)$$

Borne de Shannon

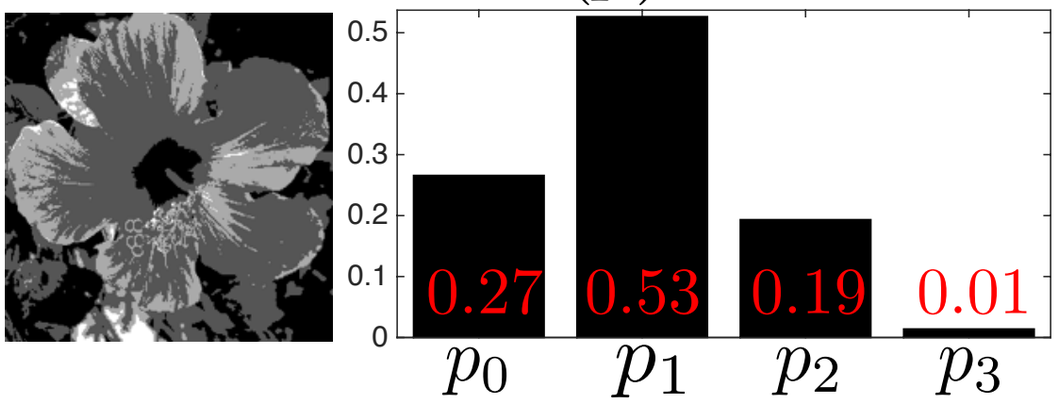
Théorème de Shannon:

si les symboles sont tirés au **hasard** selon la loi $(p_i)_i$

le nombre **moyen** de bits d'un code **préfixe** est supérieur à $H(p)$.

$H(p)$ faible $\Leftrightarrow$ peu d'information $\Leftrightarrow$ facile à coder.

$H(p) \approx 1.54$



Codage préfixe:

$0 \to c_0, \quad 1 \to c_1, \quad 2 \to c_2, \quad 3 \to c_3$

Exemple:

$0 \to 10, \quad 1 \to 0, \quad 2 \to 110, \quad 3 \to 111$

$$\text{Long. moy.} = p_0 \times \text{Long}(c_0) + p_1 \times \text{Long}(c_1) + p_2 \times \text{Long}(c_2) + p_3 \times \text{Long}(c_3)$$

$$\text{Long. moy.} = 0.27 \times 2 + 0.53 \times 1 + 0.19 \times 3 + 0.01 \times 3 = 1.67 \text{ bits/symbole.}$$

Borne de Shannon

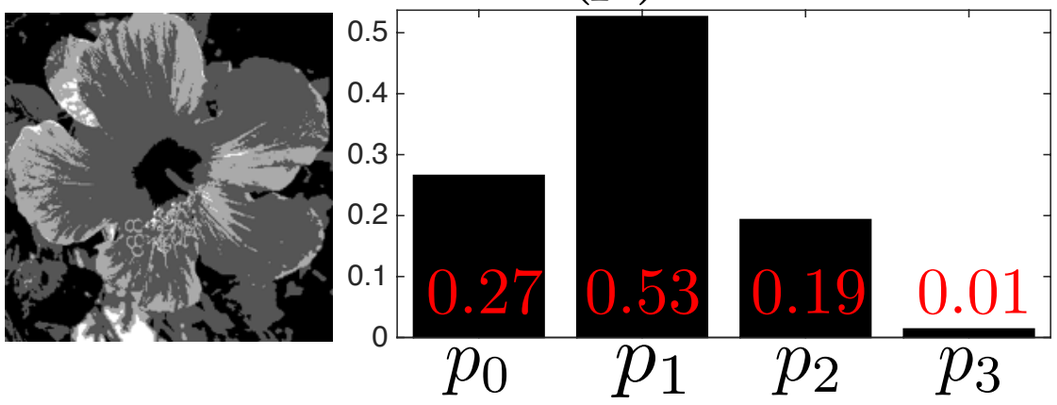
Théorème de Shannon:

si les symboles sont tirés au **hasard** selon la loi $(p_i)_i$

le nombre **moyen** de bits d'un code **préfixe** est supérieur à $H(p)$.

$H(p)$ faible $\Leftrightarrow$ peu d'information $\Leftrightarrow$ facile à coder.

$H(p) \approx 1.54$



Codage préfixe:

$0 \to c_0, \quad 1 \to c_1, \quad 2 \to c_2, \quad 3 \to c_3$

Exemple:

$0 \to 10, \quad 1 \to 0, \quad 2 \to 110, \quad 3 \to 111$

$$\text{Long. moy.} = p_0 \times \text{Long}(c_0) + p_1 \times \text{Long}(c_1) + p_2 \times \text{Long}(c_2) + p_3 \times \text{Long}(c_3)$$

$$\text{Long. moy.} = 0.27 \times 2 + 0.53 \times 1 + 0.19 \times 3 + 0.01 \times 3 = 1.67 \text{ bits/symbole.}$$

Codage uniforme
2 bits

Borne de Shannon

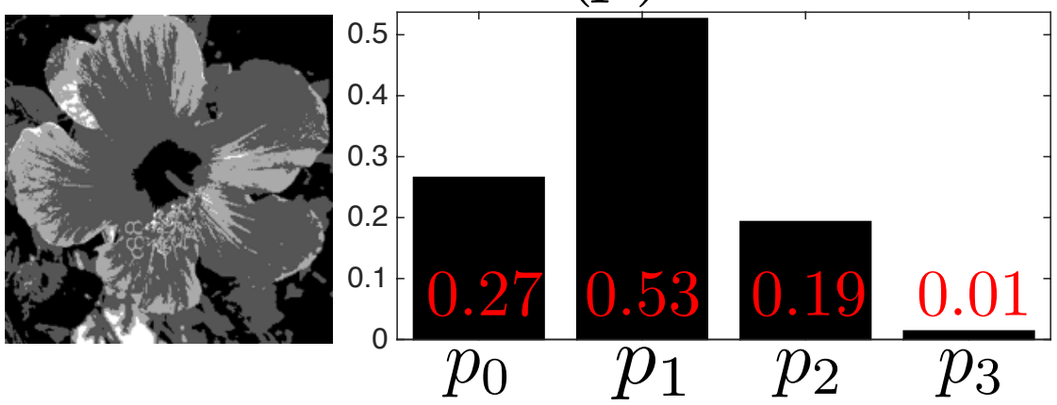
Théorème de Shannon:

si les symboles sont tirés au **hasard** selon la loi $(p_i)_i$

le nombre **moyen** de bits d'un code **préfixe** est supérieur à $H(p)$.

$H(p)$ faible $\Leftrightarrow$ peu d'information $\Leftrightarrow$ facile à coder.

$H(p) \approx 1.54$



Codage préfixe:

$0 \to c_0, \quad 1 \to c_1, \quad 2 \to c_2, \quad 3 \to c_3$

Exemple:

$0 \to 10, \quad 1 \to 0, \quad 2 \to 110, \quad 3 \to 111$

$$\text{Long. moy.} = p_0 \times \text{Long}(c_0) + p_1 \times \text{Long}(c_1) + p_2 \times \text{Long}(c_2) + p_3 \times \text{Long}(c_3)$$

$$\text{Long. moy.} = 0.27 \times 2 + 0.53 \times 1 + 0.19 \times 3 + 0.01 \times 3 = 1.67 \text{ bits/symbole.}$$

Codage uniforme 2 bits **>** Codage variable 1.67 bits

Borne de Shannon

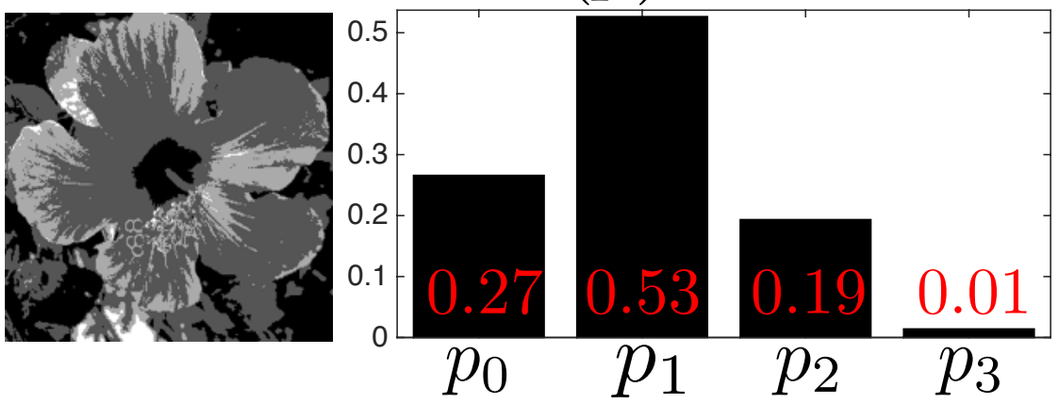
Théorème de Shannon:

si les symboles sont tirés au **hasard** selon la loi $(p_i)_i$

le nombre **moyen** de bits d'un code **préfixe** est supérieur à $H(p)$.

$H(p)$ faible $\Leftrightarrow$ peu d'information $\Leftrightarrow$ facile à coder.

$H(p) \approx 1.54$



Codage préfixe:

$0 \to c_0, \quad 1 \to c_1, \quad 2 \to c_2, \quad 3 \to c_3$

Exemple:

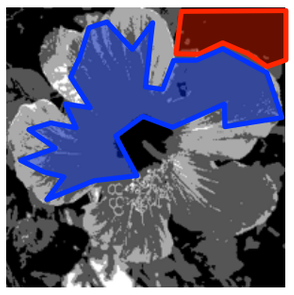
$0 \to 10, \quad 1 \to 0, \quad 2 \to 110, \quad 3 \to 111$

$$\text{Long. moy.} = p_0 \times \text{Long}(c_0) + p_1 \times \text{Long}(c_1) + p_2 \times \text{Long}(c_2) + p_3 \times \text{Long}(c_3)$$

$$\text{Long. moy.} = 0.27 \times 2 + 0.53 \times 1 + 0.19 \times 3 + 0.01 \times 3 = 1.67 \text{ bits/symbole.}$$

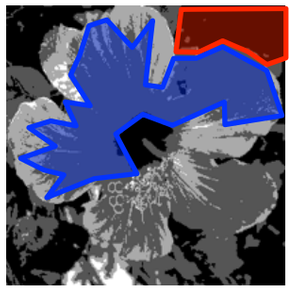
Codage uniforme 2 bits **>** Codage variable 1.67 bits **>** Entropie 1.54 bits

Comment faire mieux?



→ Les pixels ne sont pas indépendants les uns des autres !
Retransformation des symboles ⇔ diminuer l'entropie.

Comment faire mieux?



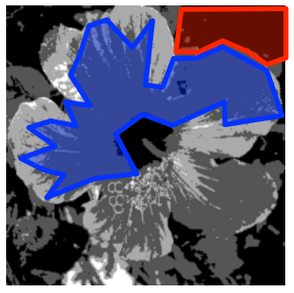
0	1	3	2	0
3	2	2	1	2
2	1	1	2	2
2	1	1	2	2
2	1	1	2	1

→ Les pixels ne sont pas indépendants les uns des autres !

Retransformation des symboles ⇔ diminuer l'entropie.

0	1	3	2	0	3	2	2	1	2	2	1	1	2	2	2	1	1	2	2	1	1	2	1

Comment faire mieux?



0	1	3	2	0
3	2	2	1	2
2	1	1	2	2
2	1	1	2	2
2	1	1	2	1

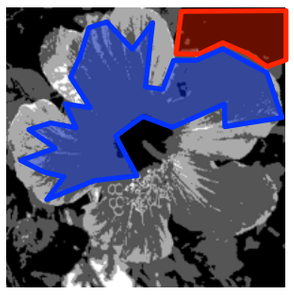
→ Les pixels ne sont pas indépendants les uns des autres !

Retransformation des symboles ⇔ diminuer l'entropie.

0	1	3	2	0	3	2	2	1	2	2	1	1	2	2	2	1	1	2	2	1	1	2	1

0																						

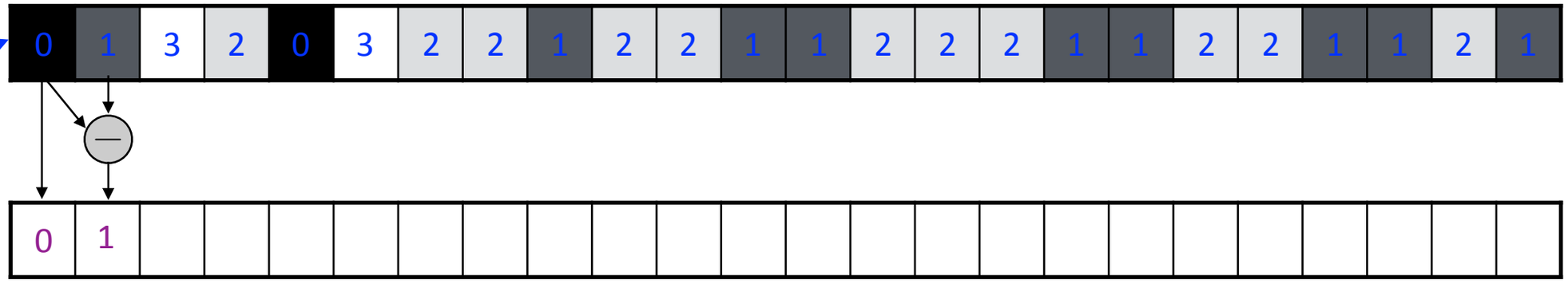
Comment faire mieux?



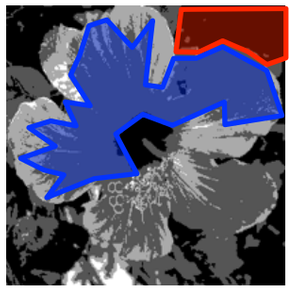
0	1	3	2	0
3	2	2	1	2
2	1	1	2	2
2	1	1	2	2
2	1	1	2	1

→ Les pixels ne sont pas indépendants les uns des autres !

Retransformation des symboles ⇔ diminuer l'entropie.



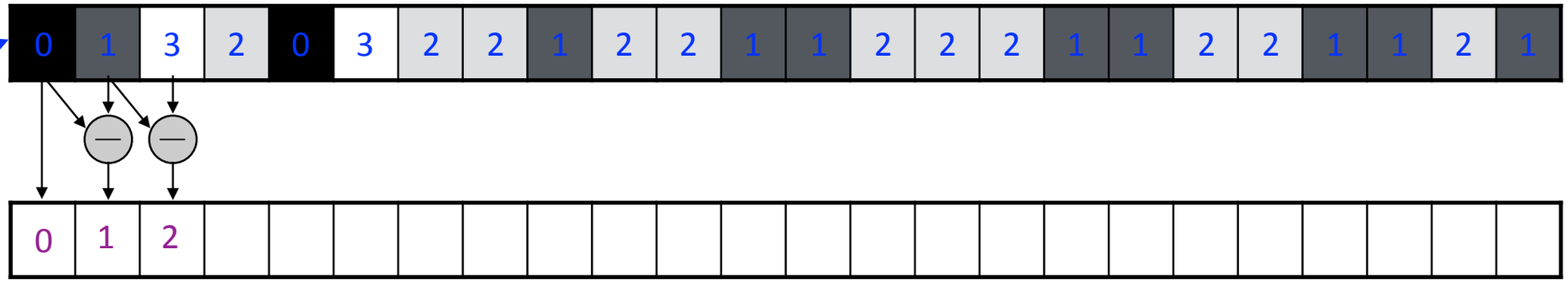
Comment faire mieux?



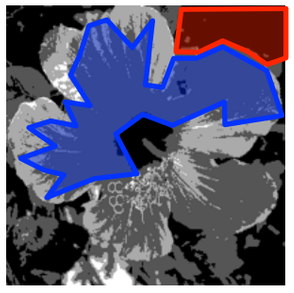
0	1	3	2	0
3	2	2	1	2
2	1	1	2	2
2	1	1	2	2
2	1	1	2	1

→ Les pixels ne sont pas indépendants les uns des autres !

Retransformation des symboles ⇔ diminuer l'entropie.



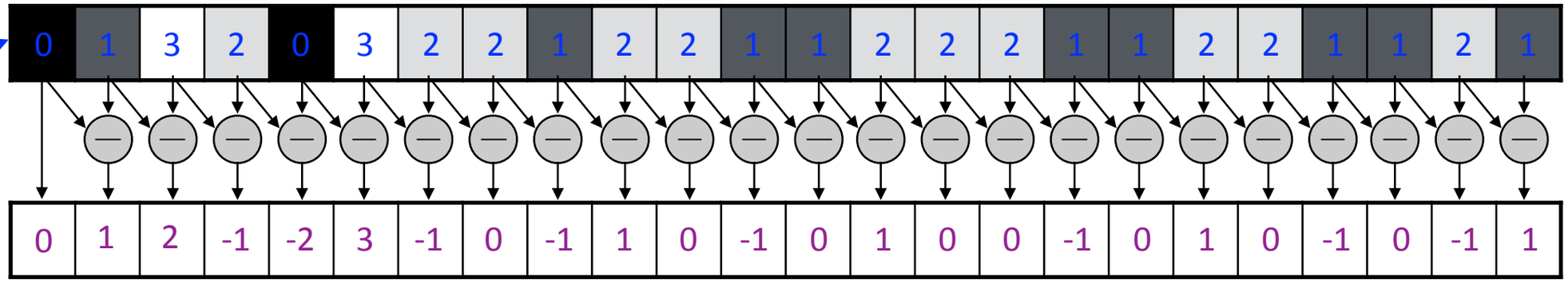
Comment faire mieux?



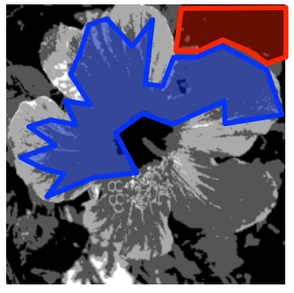
0	1	3	2	0
3	2	2	1	2
2	1	1	2	2
2	1	1	2	2
2	1	1	2	1

→ Les pixels ne sont pas indépendants les uns des autres !

Retransformation des symboles ⇔ diminuer l'entropie.

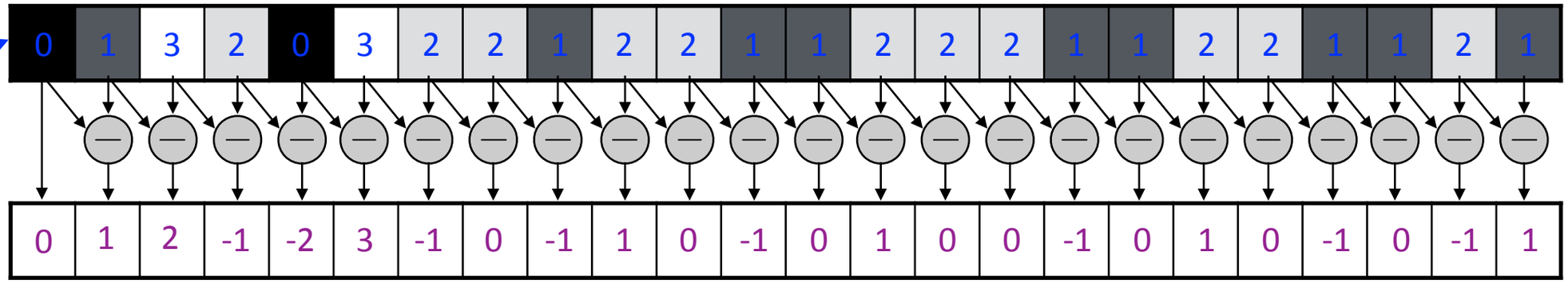


Comment faire mieux?



0	1	3	2	0
3	2	2	1	2
2	1	1	2	2
2	1	1	2	2
2	1	1	2	1

→ Les pixels ne sont pas indépendants les uns des autres !
Retransformation des symboles ⇔ diminuer l'entropie.



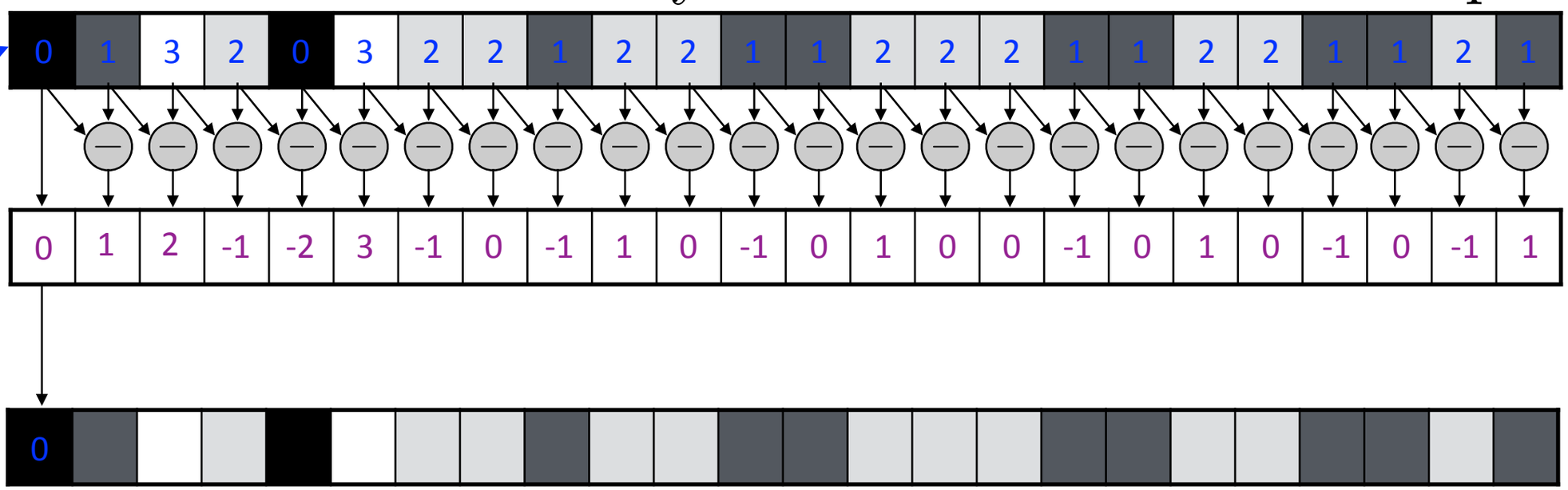
bijectivité

Comment faire mieux?

0	1	3	2	0
3	2	2	1	2
2	1	1	2	2
2	1	1	2	2
2	1	1	2	1

bijectivité

→ Les pixels ne sont pas indépendants les uns des autres !
Retransformation des symboles ⇔ diminuer l'entropie.

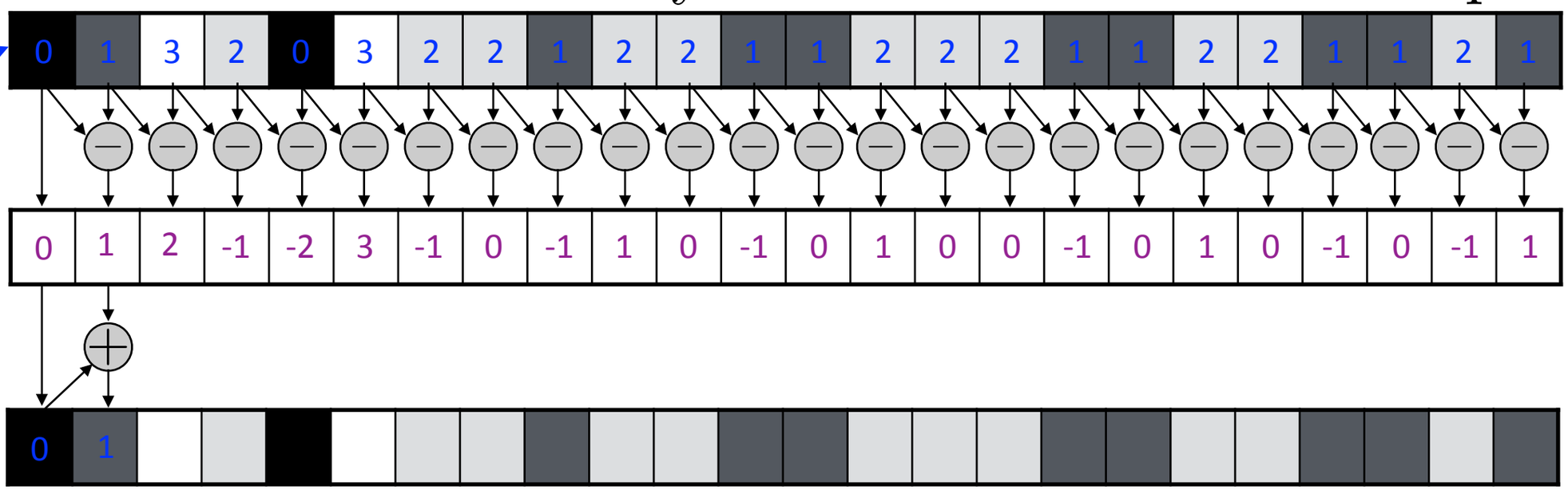


Comment faire mieux?

0	1	3	2	0
3	2	2	1	2
2	1	1	2	2
2	1	1	2	2
2	1	1	2	1

bijectivité

→ Les pixels ne sont pas indépendants les uns des autres !
Retransformation des symboles ⇔ diminuer l'entropie.

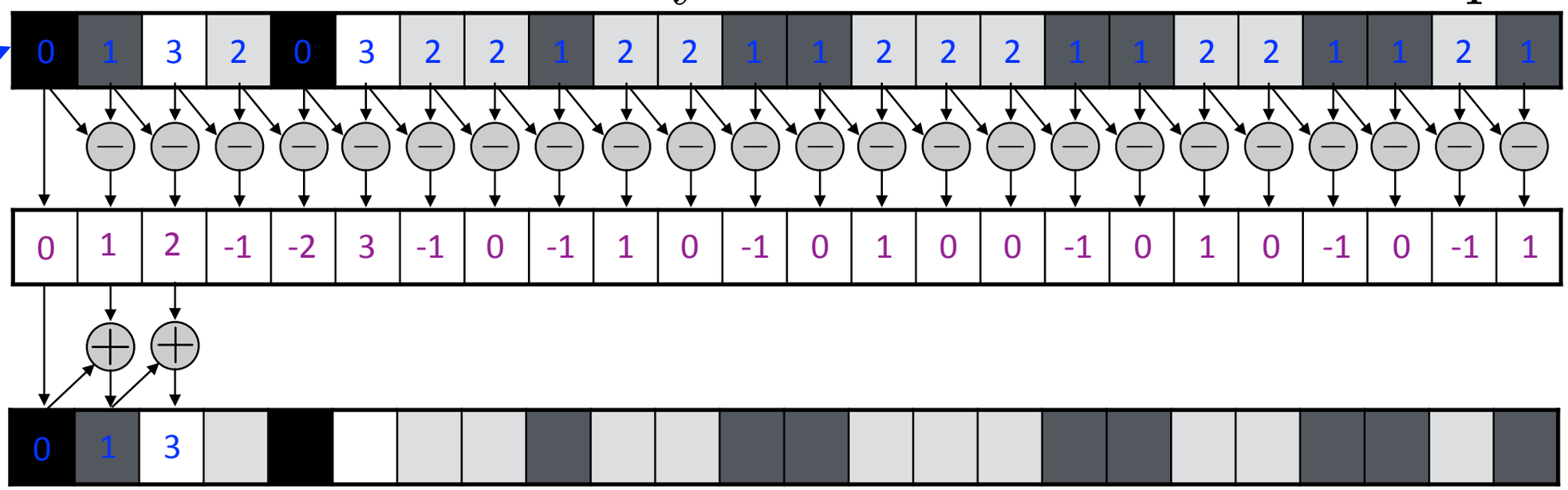


Comment faire mieux?

0	1	3	2	0
3	2	2	1	2
2	1	1	2	2
2	1	1	2	2
2	1	1	2	1

bijectivité

→ Les pixels ne sont pas indépendants les uns des autres !
Retransformation des symboles ⇔ diminuer l'entropie.

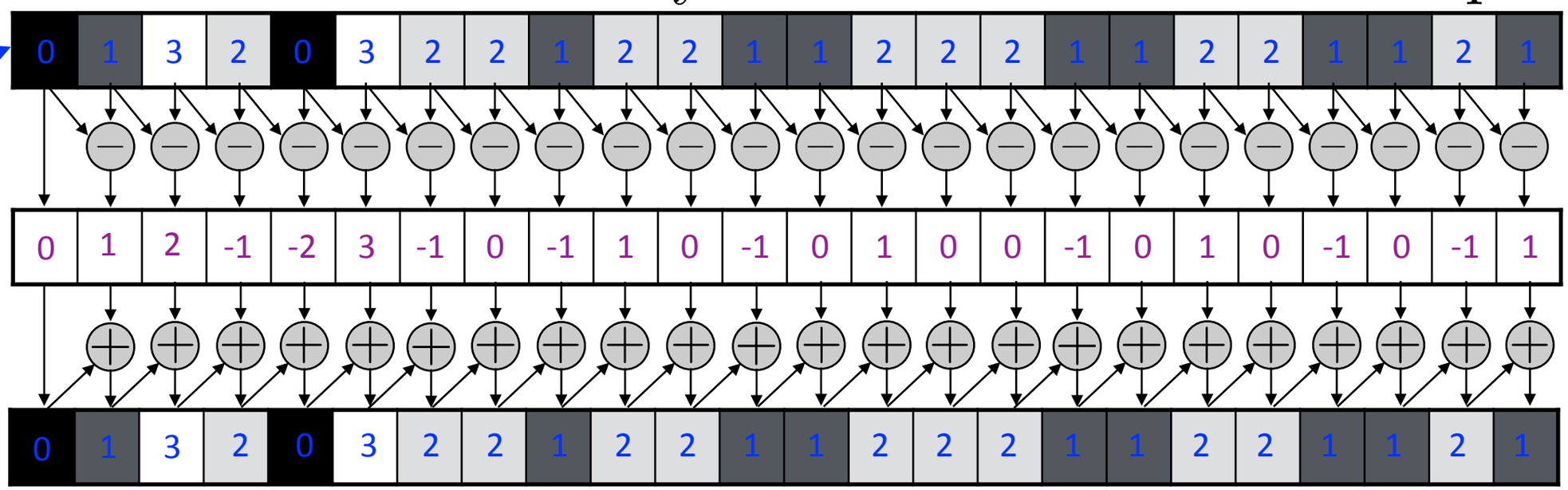


Comment faire mieux?

0	1	3	2	0
3	2	2	1	2
2	1	1	2	2
2	1	1	2	2
2	1	1	2	1

bijectivité

→ Les pixels ne sont pas indépendants les uns des autres !
Retransformation des symboles ⇔ diminuer l'entropie.

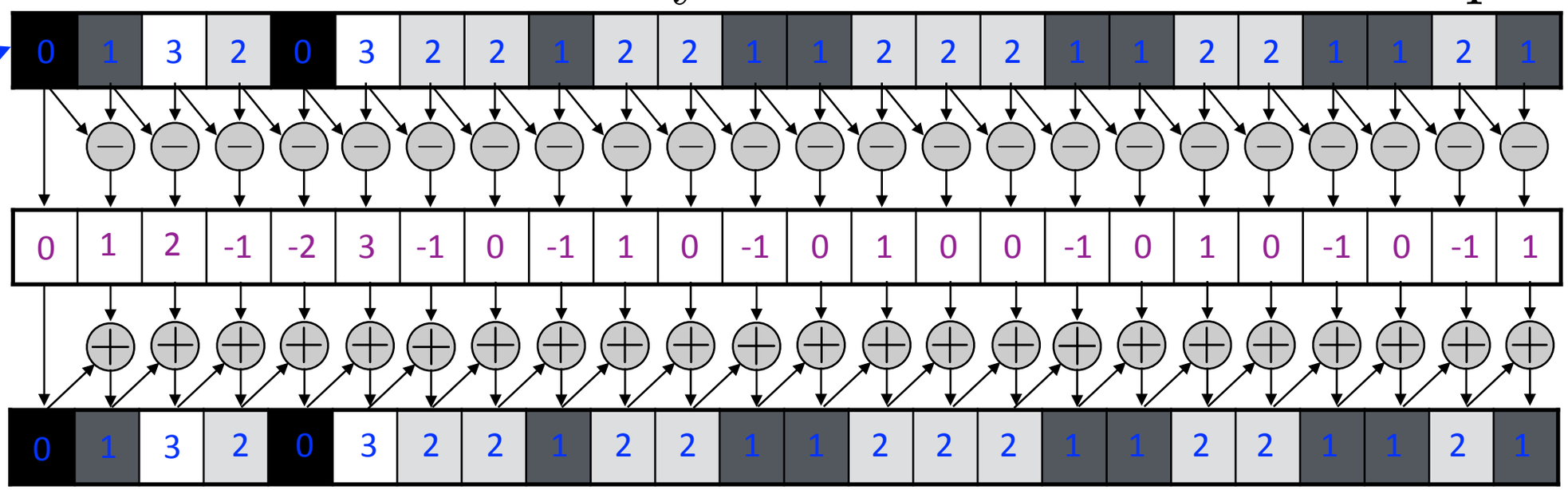


Comment faire mieux?

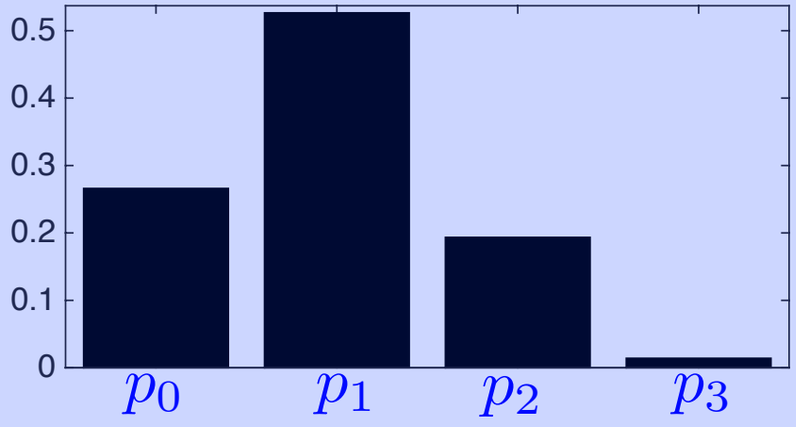
0	1	3	2	0
3	2	2	1	2
2	1	1	2	2
2	1	1	2	2
2	1	1	2	1

bijectivité

→ Les pixels ne sont pas indépendants les uns des autres !
Retransformation des symboles ⇔ diminuer l'entropie.



$H(p) = 1.54$

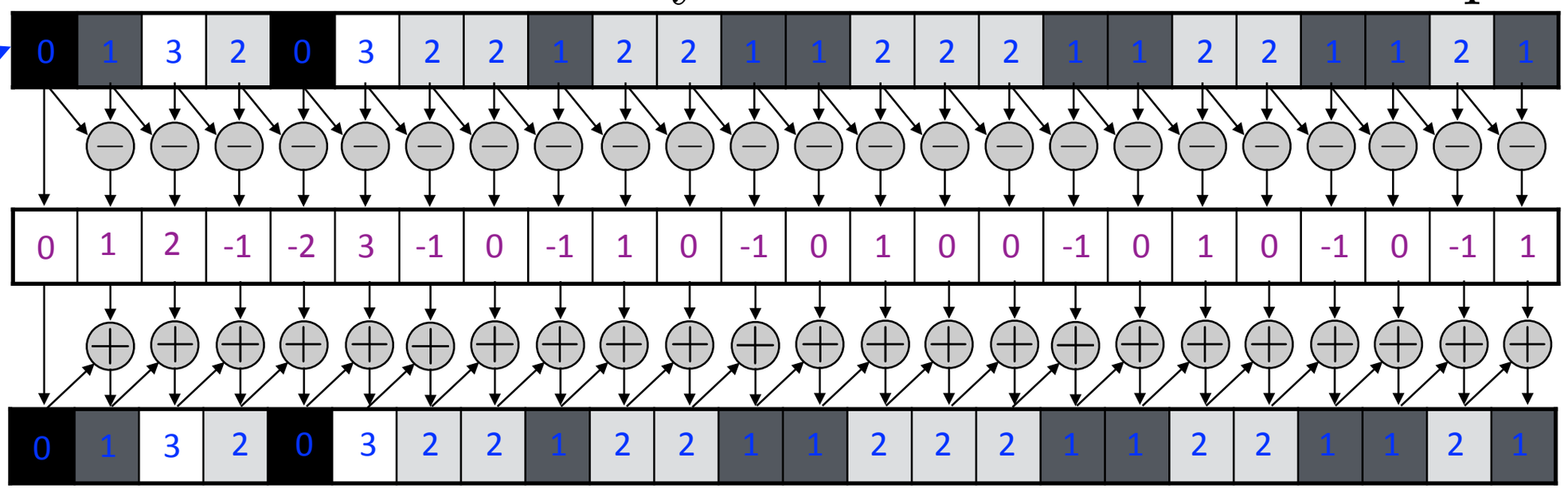


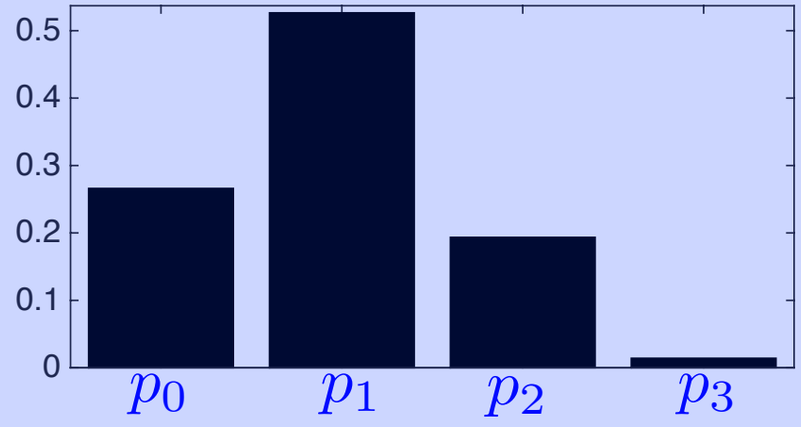
Comment faire mieux?

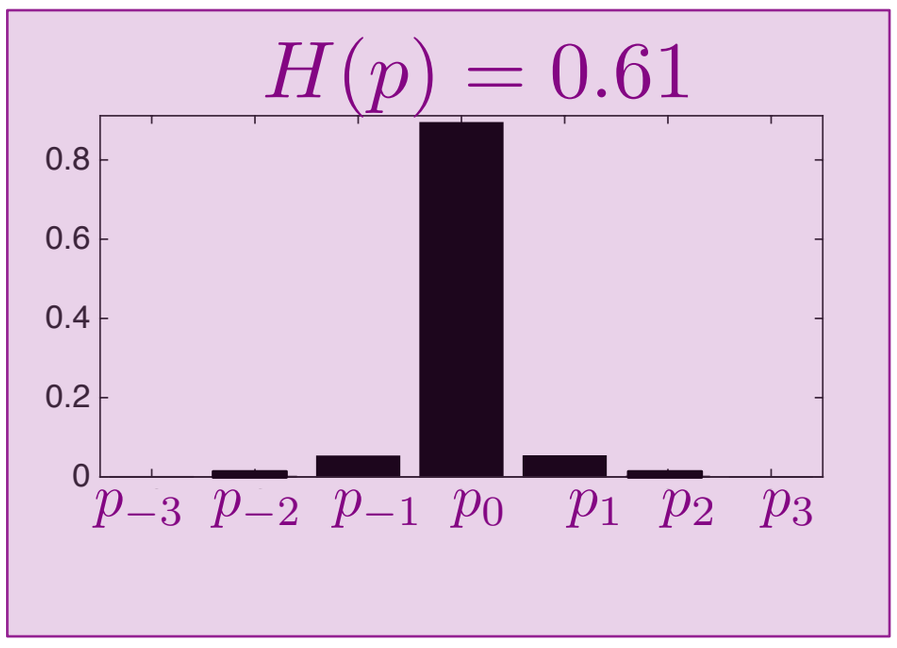
0	1	3	2	0
3	2	2	1	2
2	1	1	2	2
2	1	1	2	2
2	1	1	2	1

bijectivité

→ Les pixels ne sont pas indépendants les uns des autres !
Retransformation des symboles ⇔ diminuer l'entropie.





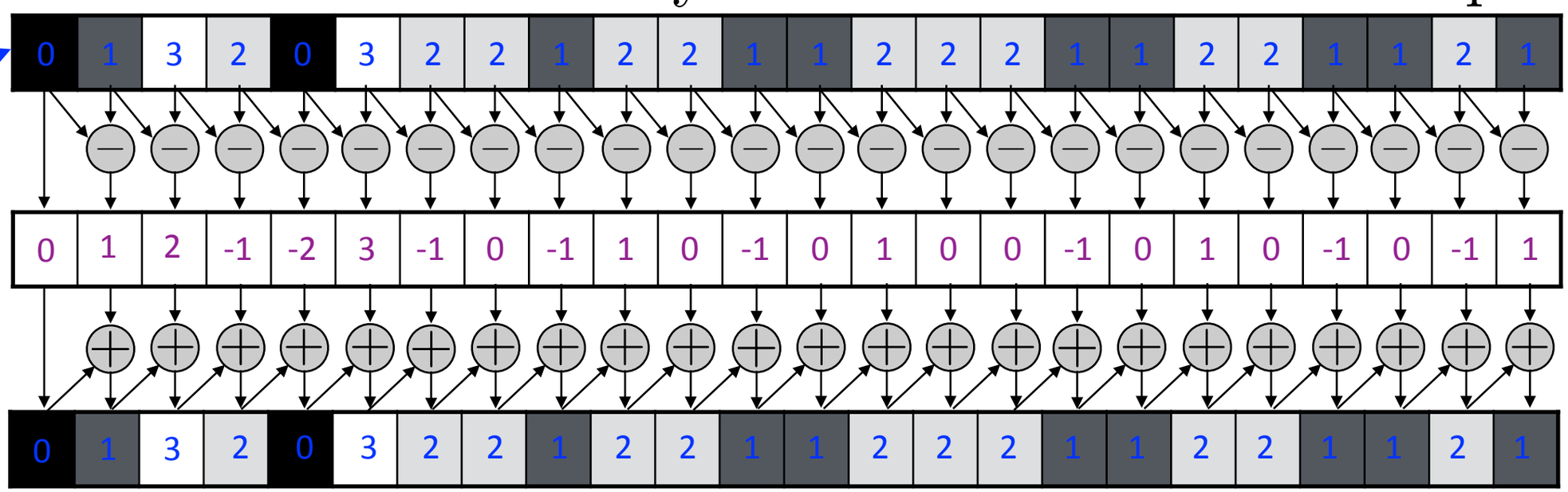


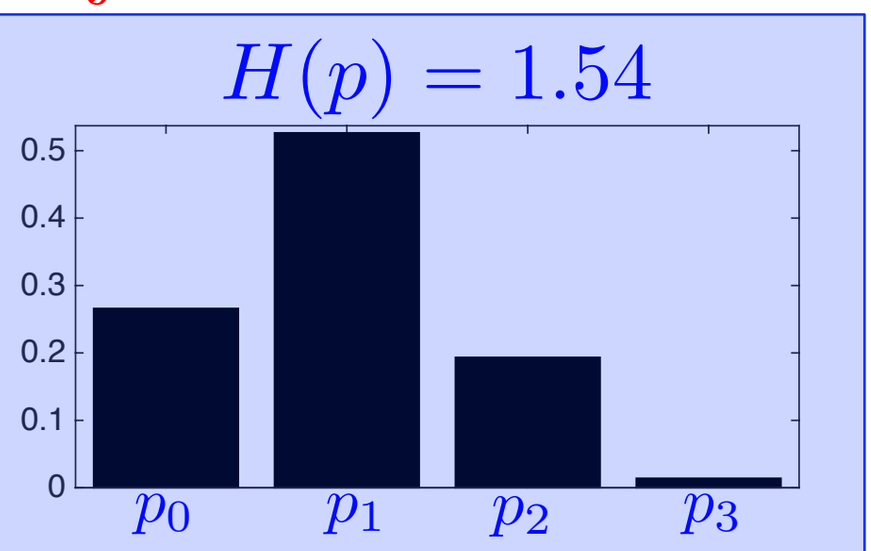
Comment faire mieux?

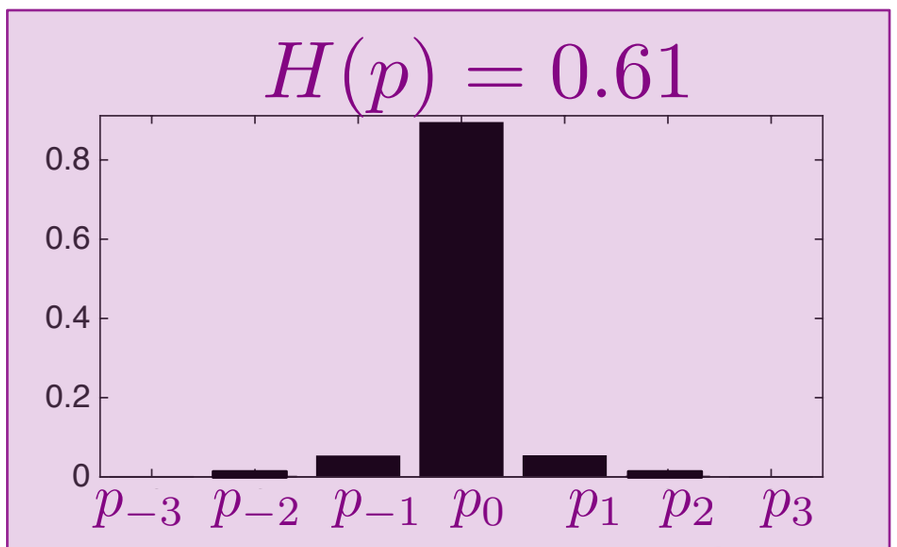
0	1	3	2	0
3	2	2	1	2
2	1	1	2	2
2	1	1	2	2
2	1	1	2	1

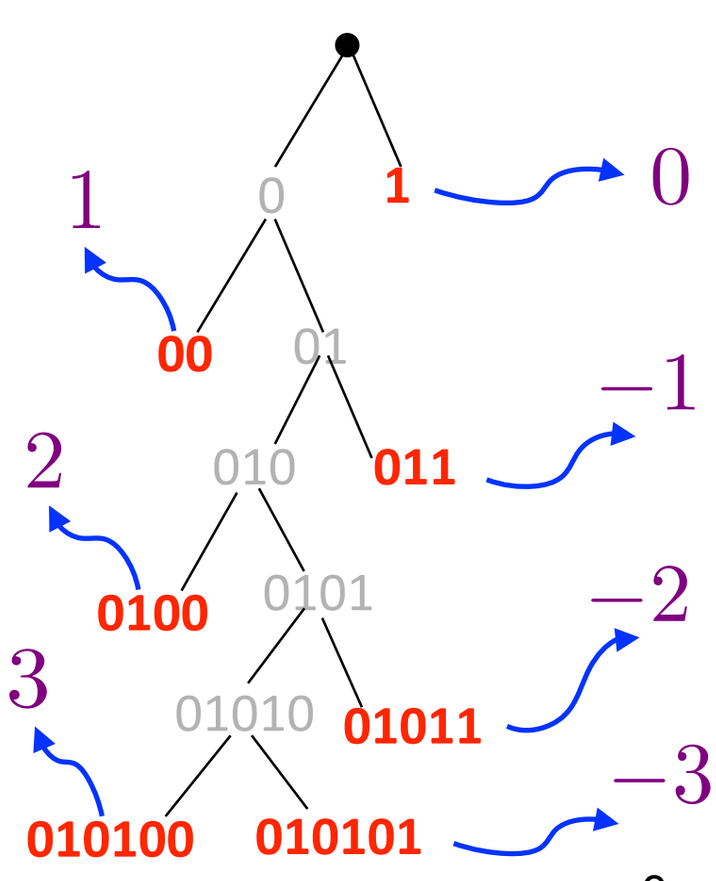
bijectivité

→ Les pixels ne sont pas indépendants les uns des autres !
Retransformation des symboles ⇔ diminuer l'entropie.







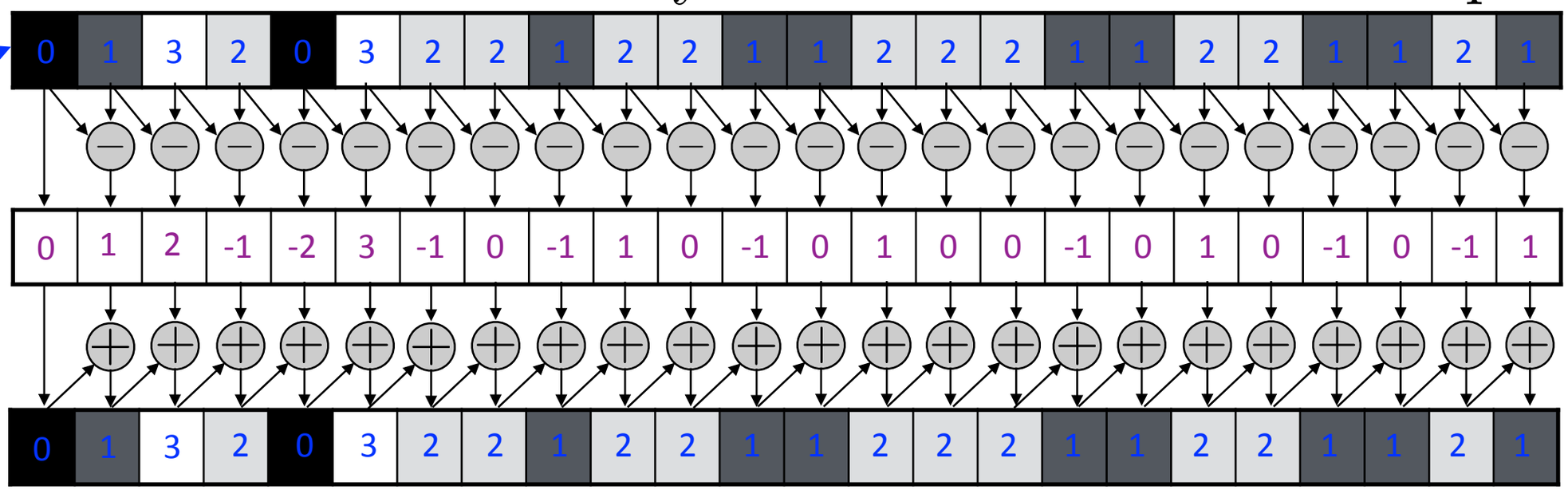


Comment faire mieux?

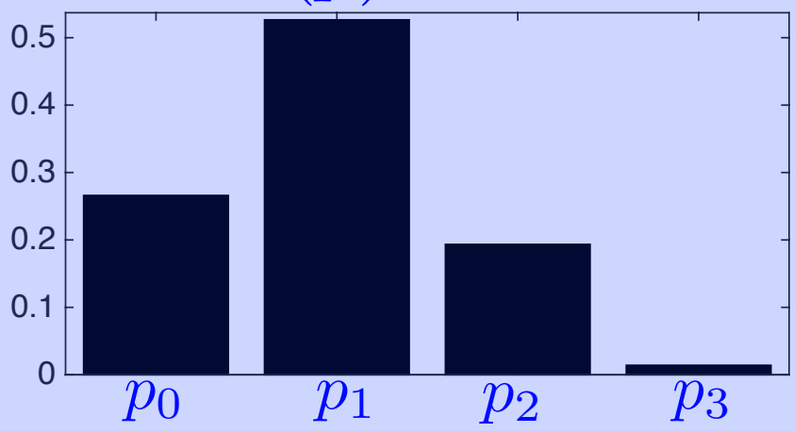
0	1	3	2	0
3	2	2	1	2
2	1	1	2	2
2	1	1	2	2
2	1	1	2	1

bijectivité

→ Les pixels ne sont pas indépendants les uns des autres !
Retransformation des symboles ⇔ diminuer l'entropie.

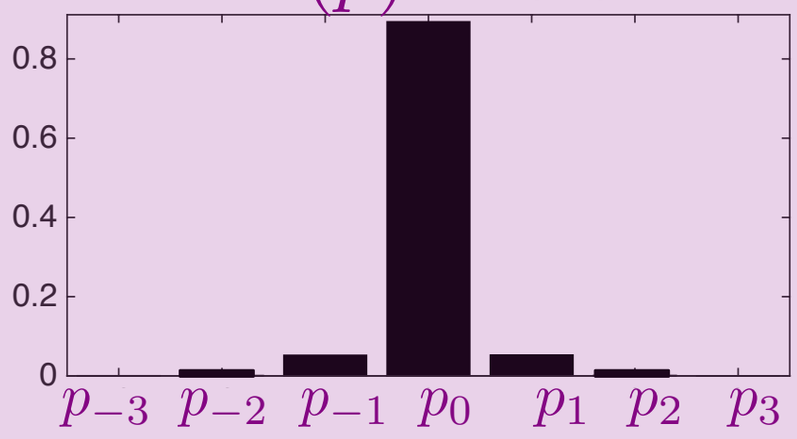


$H(p) = 1.54$



Long. moy. = 1.67 bits.

$H(p) = 0.61$



Long. moy. = 1.16 bits.

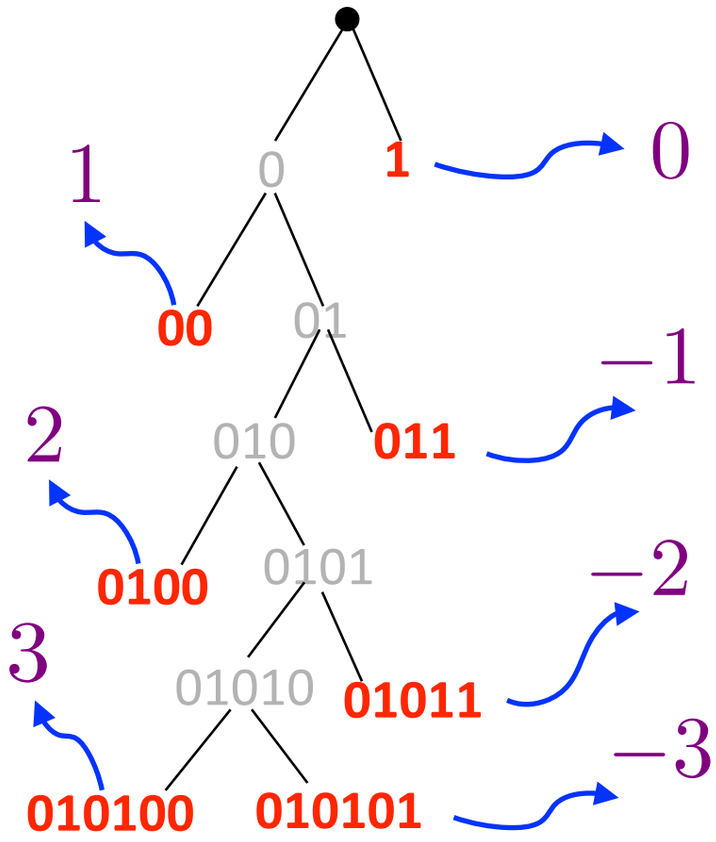


Image 256 × 256 pixels:

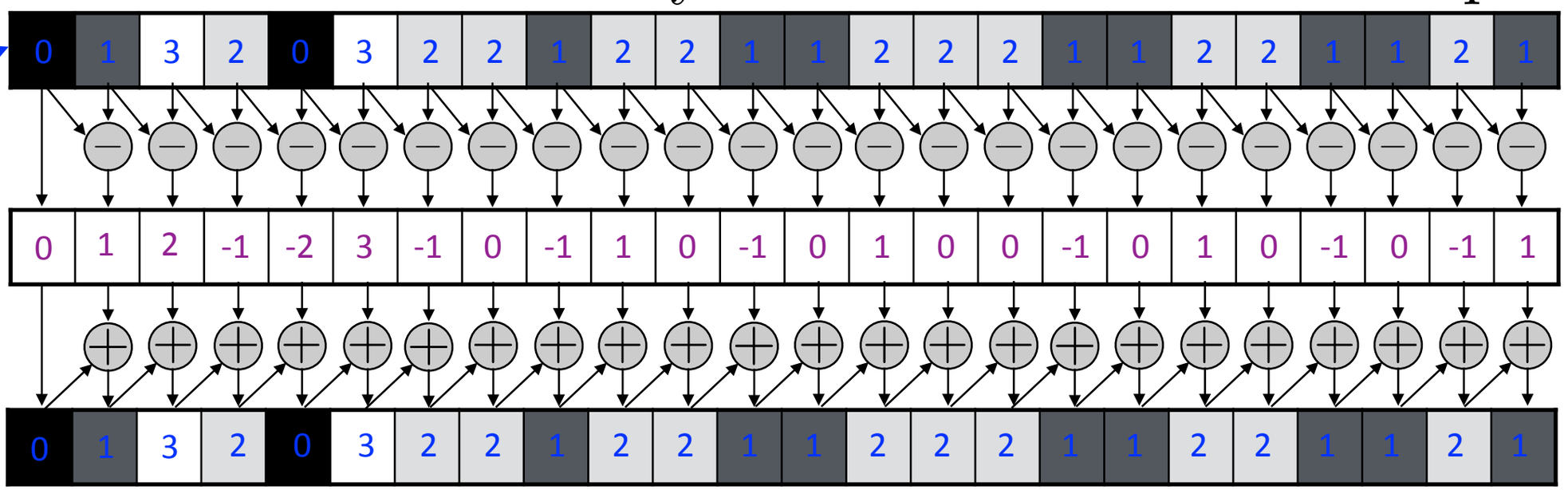
16.3 ko

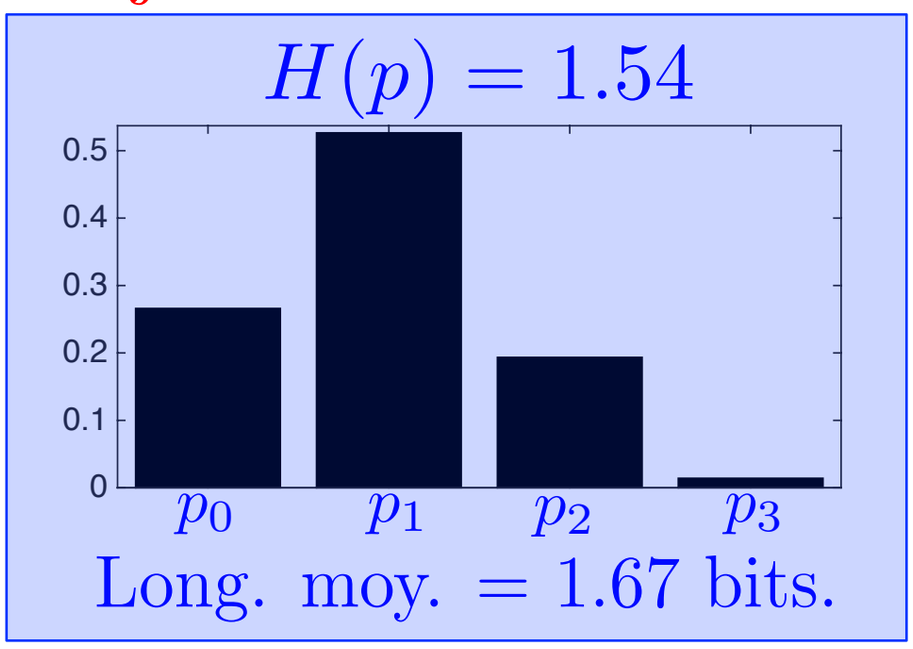
Comment faire mieux?

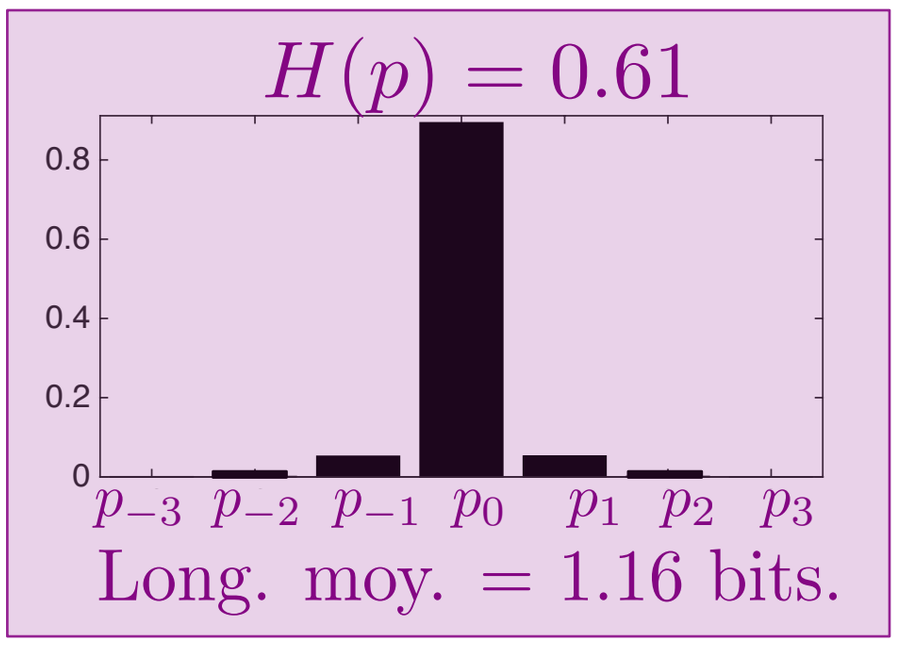
0	1	3	2	0
3	2	2	1	2
2	1	1	2	2
2	1	1	2	2
2	1	1	2	1

bijectivité

→ Les pixels ne sont pas indépendants les uns des autres !
Retransformation des symboles ⇔ diminuer l'entropie.







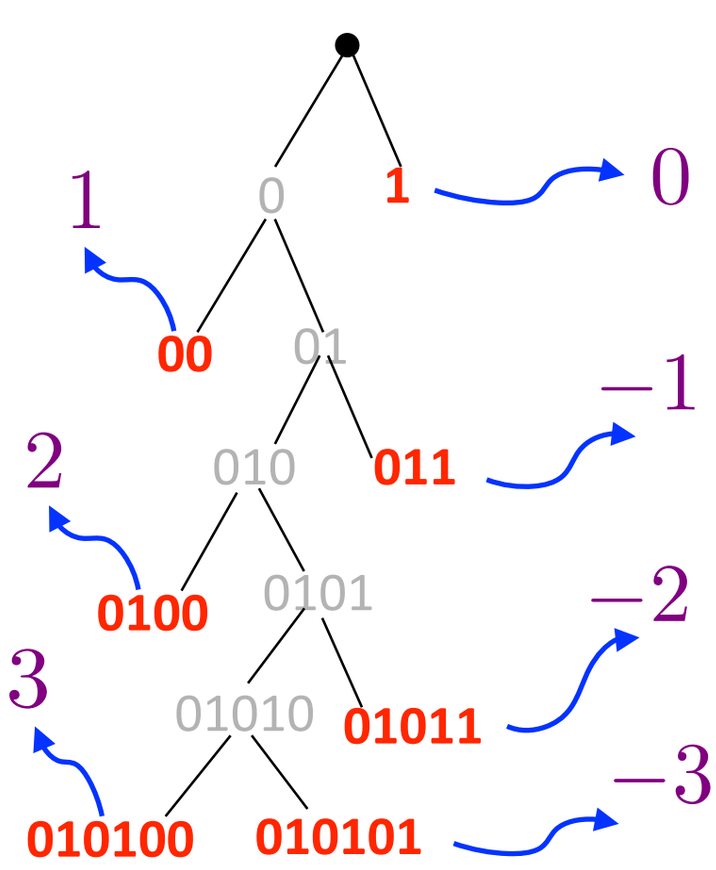


Image 256×256 pixels:

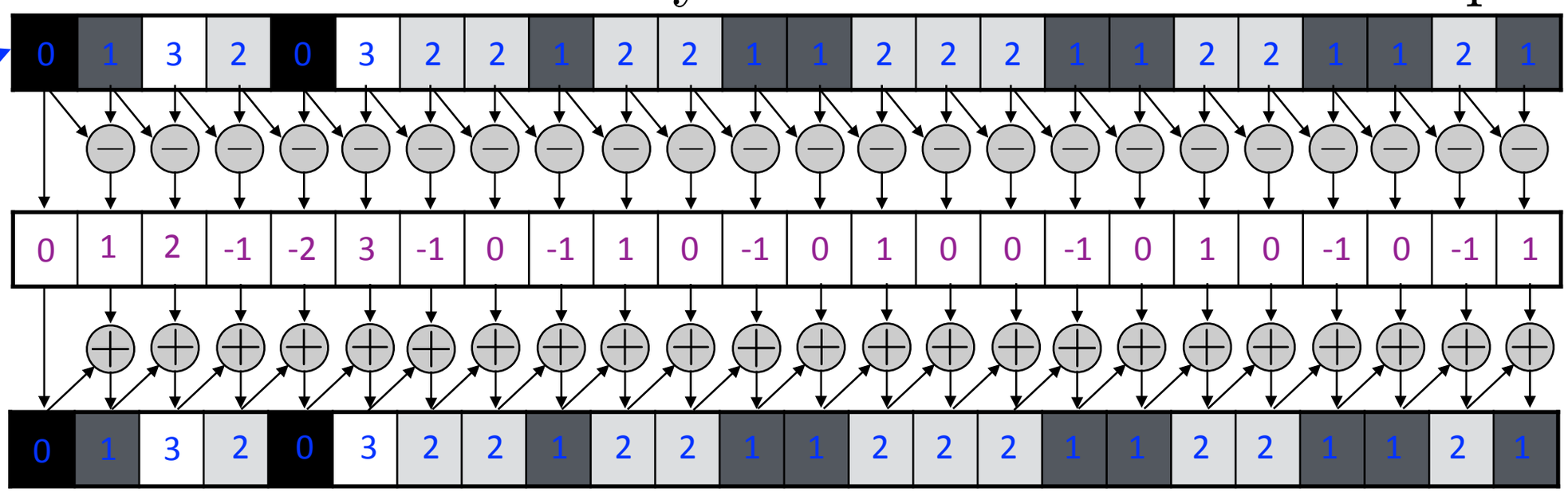
16.3 ko → 13.7 ko

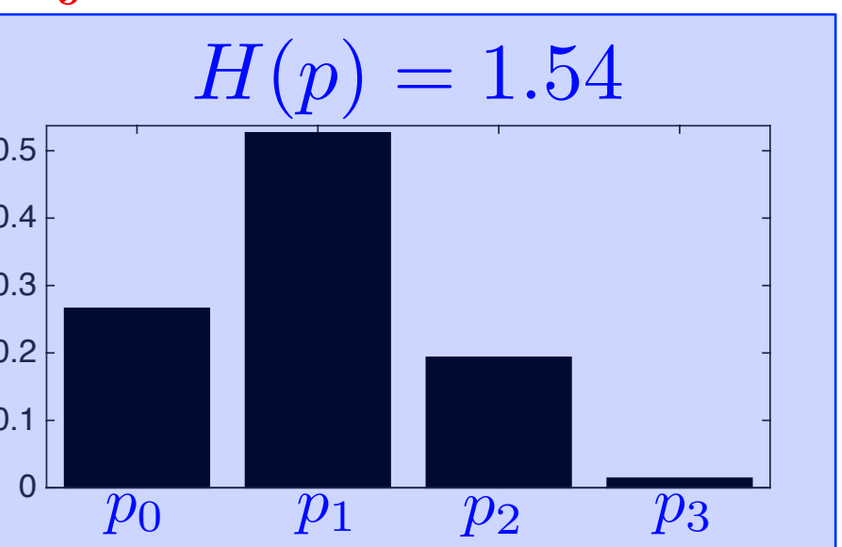
Comment faire mieux?

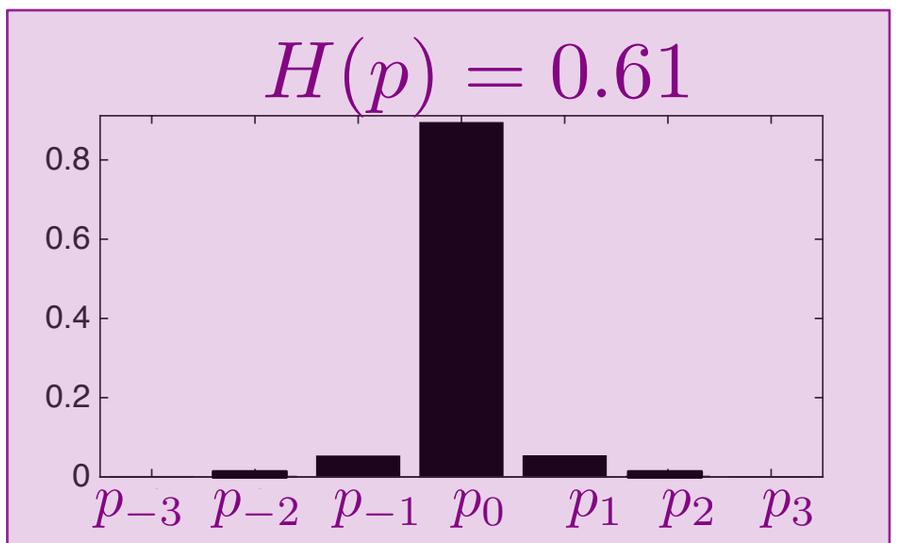
0	1	3	2	0
3	2	2	1	2
2	1	1	2	2
2	1	1	2	2
2	1	1	2	1

bijectivité

→ Les pixels ne sont pas indépendants les uns des autres !
Retransformation des symboles ⇔ diminuer l'entropie.







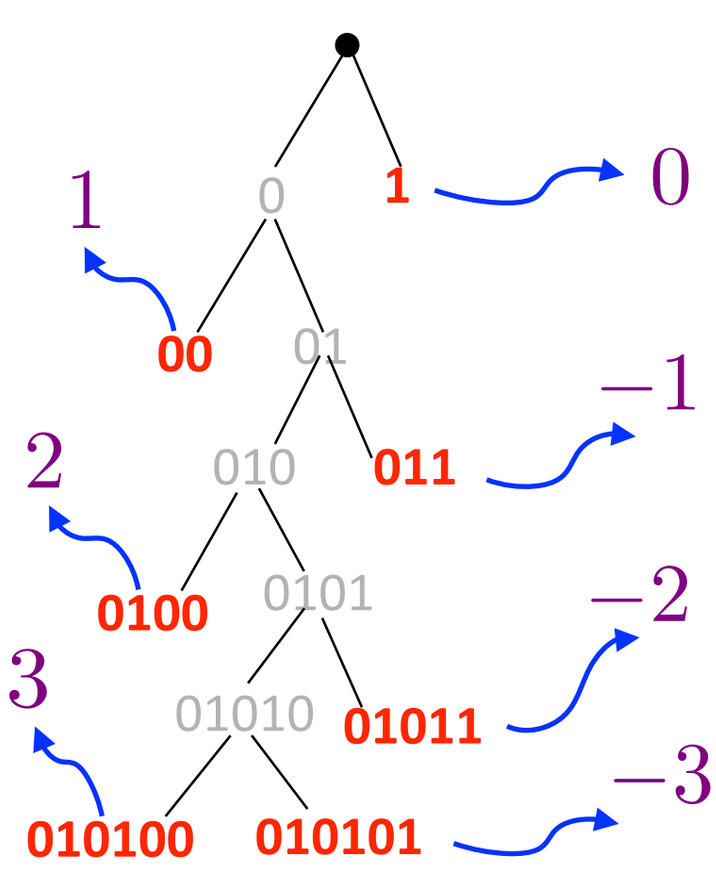
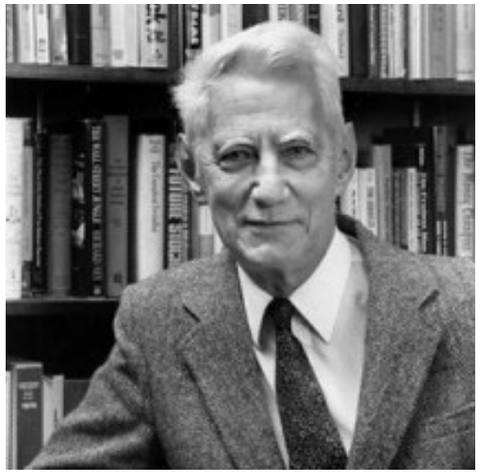


Image 256 × 256 pixels:

16.3 ko → 13.7 ko → 9.5 ko

Conclusion

Claude Shannon, père fondateur de la théorie de l'information.

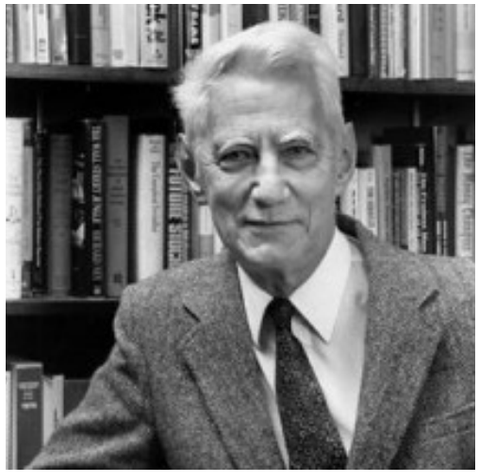


Conclusion

Claude Shannon, père fondateur de la théorie de l'information.

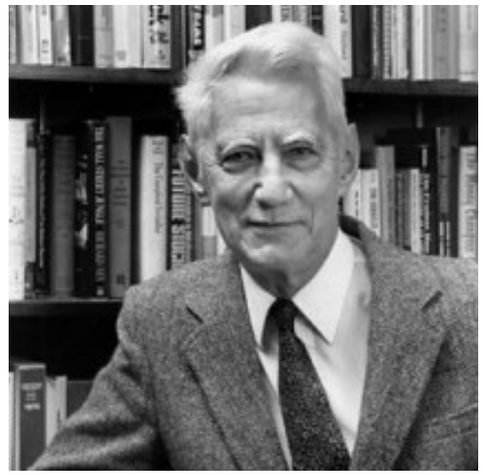
Compression : utilisation de code variables.

symbole fréquent $\rightarrow$ code court



Conclusion

Claude Shannon, père fondateur de la théorie de l'information.



Compression : utilisation de code variables.

symbole fréquent $\rightarrow$ code court

Non abordé: comment calculer ces codes ?

$\rightarrow$ arbre de Huffman (1952).

$\rightarrow$ codage arithmétique (1976).

Merci
Questions ?