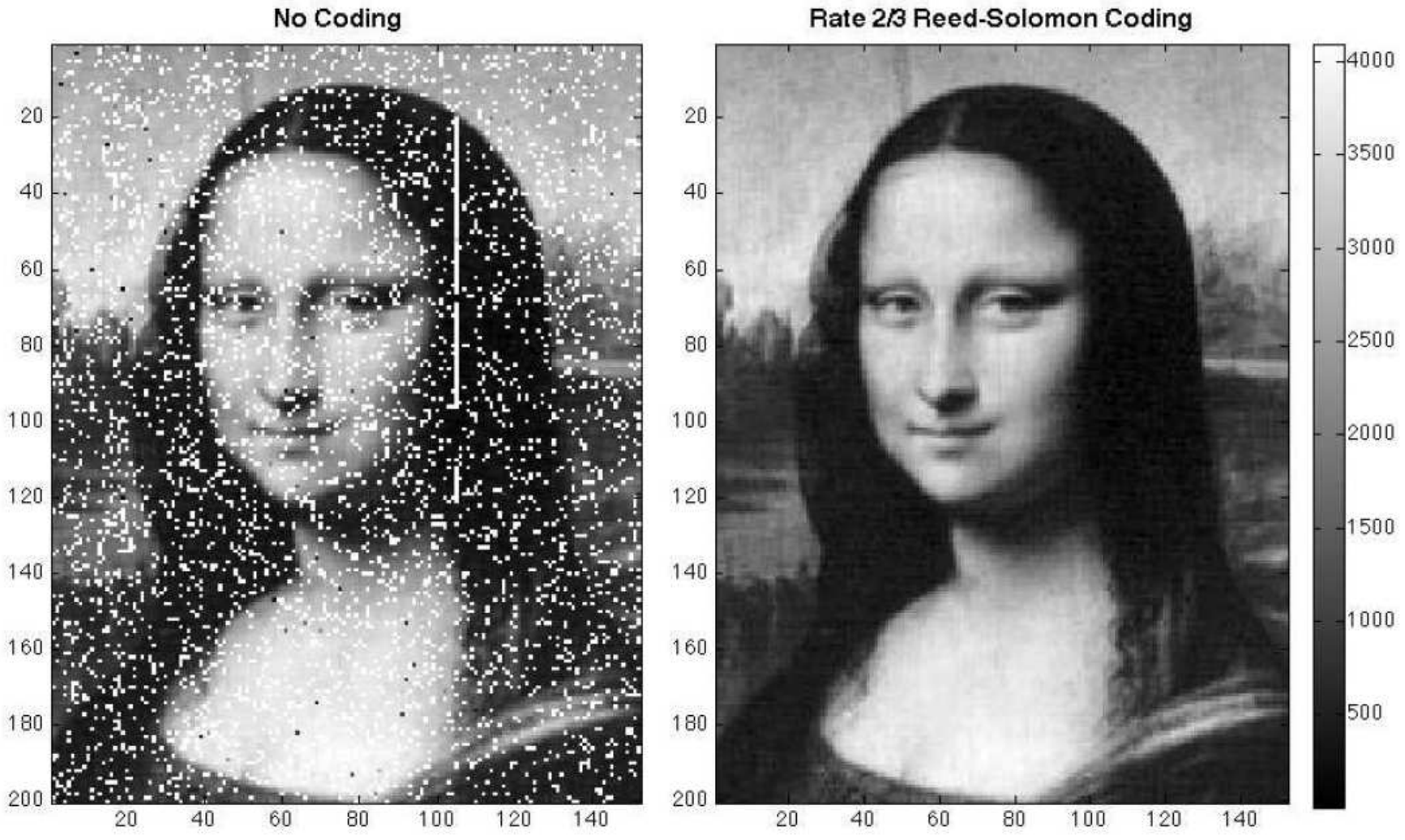
Encodage de l'information

vincent.huvelle@ac-grenoble.fr


No Coding
Rate 2/3 Reed-Solomon Coding
20
40
60
80
100
120
140
160
180
200
4000
3500
3000
2500
2000
1500
1000
500

– Talking Drum

LE BIT

L'unité [1] la plus petite est *le bit* [2] *(bit=binary digit)* (0 ou 1)

- Lié à la technologie matérielle (composants, stockage)
- Favorise une omniprésence de la numération en base 2

L'unité usuelle : *l'octet* (= 8 bits)

- Permet de représenter $2^8 = 256$ valeurs différentes.
- L'ordre de grandeur : un caractère (lettre, symbole)

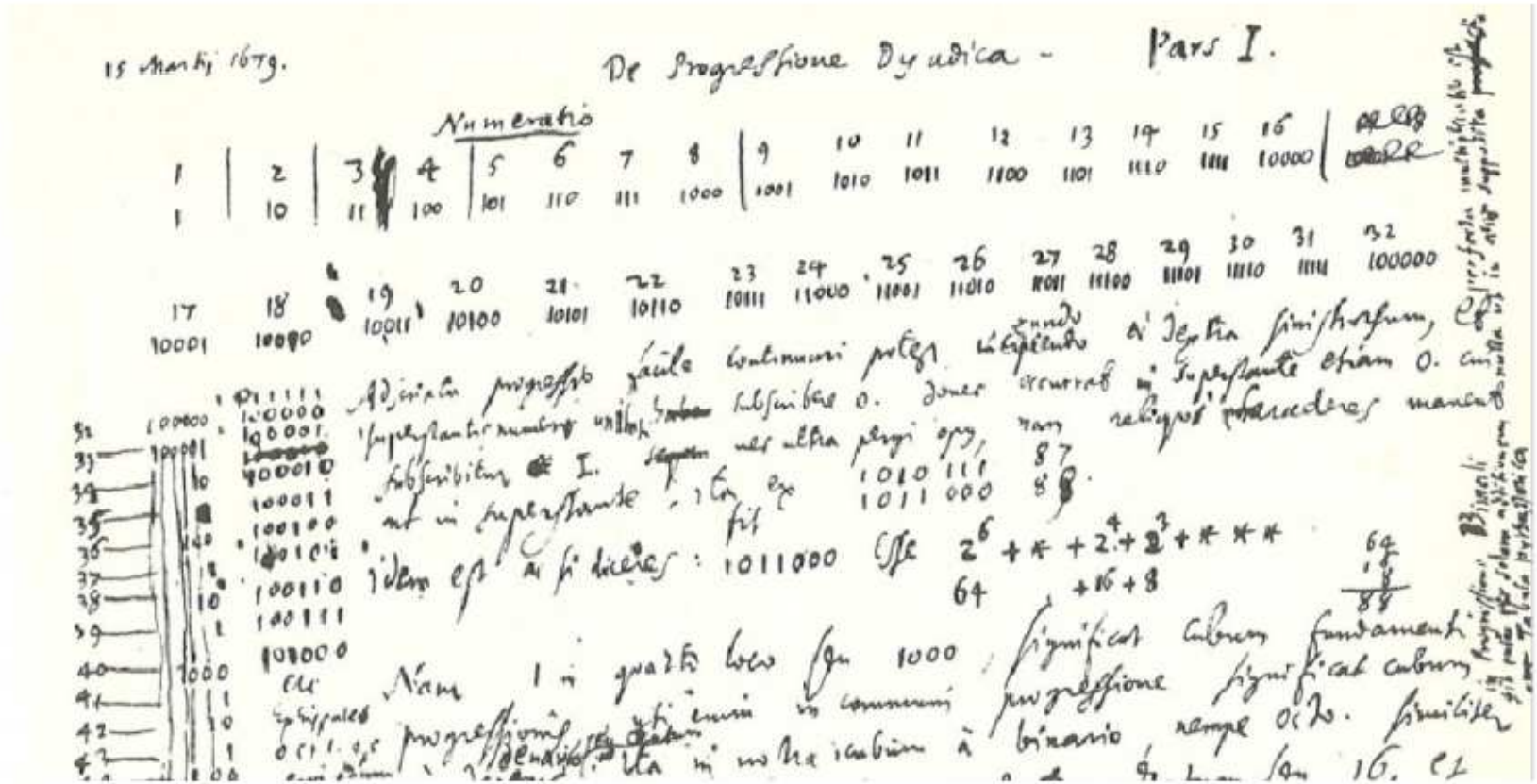
Les autres unités courantes sont des multiples du bit ou de l'octet (donc des puissances de 2).

1. Unité bien définie servant d'unité fondamental
2. Unité de mesure de l'information selon SHANNON (A la sortie du MIT, il travaille sur la sécurité du système X (ligne téléphonique directe entre Winston Churchill et Franklin Roosevelt), énoncé dans une monographie : « Une théorie mathématique de la communication »

AUTRES UNITÉS

Unité	Valeur	Ordre de grandeur
kilo-octet	1 Ko = 2^{10} o = 1024 o	un mél
mega-octet	1 Mo = 2^{20} o = 1024 Ko	une disquette
giga-octet	1 Go = 2^{30} o = 1024 Mo	un DVD
tera-octet	1 To = 2^{40} o = 1024 Go	un (gros) disque dur

L'idée d'utiliser du binaire est plus ancienne que sur les ordinateurs :



Codage des nombres entiers

Tous les entiers positifs sont représentés directement en binaire :

$$(b_n \ldots b_0)_2 = \sum_{i=0}^{n} b_i 2^i \text{avec } b_i \in \{0, 1\}$$

- $(10101)_2$ représente 21
- Cf méthode des divisions successives.

Pour les entiers relatifs, l'idée naturel est de rajouter un bit (le premier) pour le signe.

- +82 codé sur un octet : « 0 1010010 »
- -82 codé sur un octet avec la méthode du bit de signe : « 1 1010010 »

Mais cette représentation n'est pas adaptée aux opérations arithmétiques de base...

Une méthode un peu plus efficiente consiste pour coder les entiers négatifs :

- Prendre le complément à 2 du codage de n et d'ajouter 1 [3].

Un exemple : -82

- +82 est codé sur un octet par : $(01010010)_2$
- complément à 2 de +82 sur 1 octet : $(10101101)_2$
- on ajoute 1 d'où -82 codé sur 1 octet : $(10101110)_2$

Remarques :

- Sur un octet, le dernier nombre positif est $2^7 - 1$ donc codé avec un premier bit égal à 0.
- En revanche, les nombres négatifs ont un premier bit égal à 1.
- $(-1)_{10} = (11111111)_2$.

3. On peut démontrer, en effet, que si l'on ajoute un nombre a et le complément à un d'un nombre b, on trouve a-b-1.

Représentation des nombres réels

On représente certains nombres en base 10 :

$$d = \sum_{i=-n}^{n} 10^i \times d_i$$

- $12,34_{10} = 1 \times 10^1 + 2 \times 10^0 + 3 \times 10^{-1} + 4 \times 10^{-2}$
- Tous les nombres n'ont pas une représentation en base 10. (ex : $1/3$)
- On ne peut représenter exactement que les nombres qui s'écrivent sous la forme $\frac{X}{10^k}$

De même, on représente en base 2, certains nombres :

$$b = \sum_{i=-n}^{n} 2^i \times b_i$$

Cette représentation s'appelle `représentation en virgule fixe`.

- $101.11_2 = 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 + 1 \times 2^{-1} + 1 \times 2^{-2} = 5\frac{3}{4}$
- De même, on ne représente exactement que les nombres qui s'écrivent sous la forme $\frac{X}{2^k}$

Ecrire la partie décimale d'un nombre en base 2 :

- $0,3 = 0,01001[1001]$

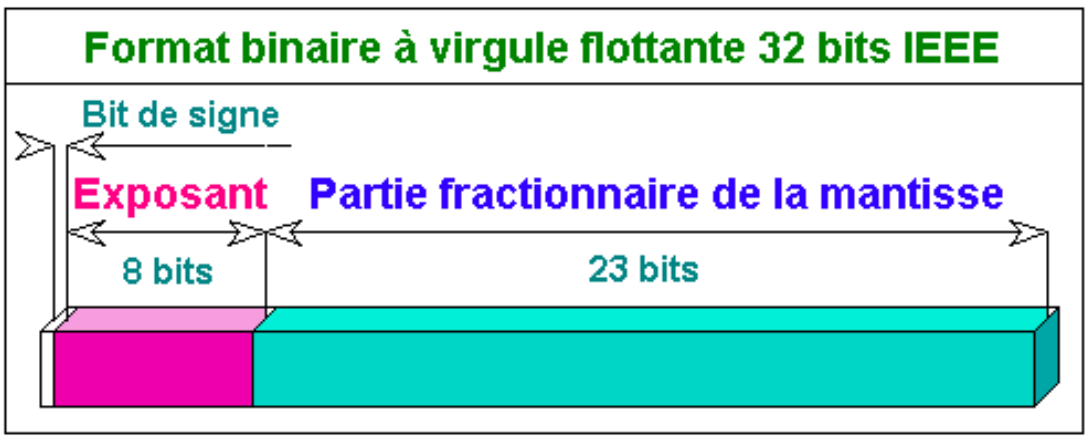
$$\begin{array}{rcl} 0,3 \times 2 & = & 0,6 \\ 0,6 \times 2 & = & 1,2 \\ \hline 0,2 \times 2 & = & 0,4 \\ 0,4 \times 2 & = & 0,8 \\ 0,8 \times 2 & = & 1,6 \\ 0,6 \times 2 & = & 1,2 \\ \hline \end{array}$$

- $0,375 = 0,011$

$$\begin{array}{rcl} 0,375 \times 2 & = & 0,75 \\ 0,75 \times 2 & = & 1,5 \\ 0,5 \times 2 & = & 1,0 \\ 0 \times 2 & = & 0 \\ \hline \end{array}$$

Représentation en virgule flottante des nombres réels.

Le principal problème de la représentation en virgule fixe est lié à sa représentation en mémoire.
La représentation en virgule flottante c'est la même idée que l'écriture scientifique (on décale la virgule d'où le nom de virgule flottante).



Utilisé pour le type "float" (simple précision)

UN EXEMPLE : 153,25

On suppose code sur :

- 1 bit pour le signe,
- 7 bits pour l'exposant,
- 10 bits pour la mantisse.

$$\begin{aligned}(153,25)_{10} &= (10011001.01)_2 \\ &= (1.001100101 \times 2^7)\end{aligned}$$

- Biais=$2^{n-1}-1$. Ici $B = 2^{7-1}-1 = 63$
- E(exposant)=D(décalage)+Biais= $7+63 = 70 = (1000110)_2$

$(153,25)_{10} = 0$	1000110	00110010100000000

Imprécisions en calcul flottant

- $3 \times 0,1 \neq 0,3$
- $0,1^2 \neq 0,01$
- $sin(\pi) \neq 0$
- $tan(\pi) \neq +\infty$
- $(a + b) + c \neq a + (+b + c)$
- $(a + b)c \neq ac + bc$

Encodage de l'information

L'encodage de l'information consiste à utiliser des codes pour représenter l'information afin de résoudre 3 types de problèmes :

- Assurer l'*intégrité* de l'information (détection et correction d'erreurs)
- minimiser le *taille* de l'information (compression)
- garantir la *sécurité* de l'information (encryptage/chiffrement).

Les informations sont stockées sous forme binaire dans un ordinateur.
Lors de la transmission ou du stockage, une information peut subir des modifications.
Pour corriger les erreurs portant sur quelques bits (mais qui peuvent engendrer des problèmes plus graves), on utilise des *codes détecteurs et correcteurs d'erreurs*.

Aux m bits de données, on ajoute donc k bits de contrôle. Ce sont donc $n = m + k$ bits qui vont être transmis ou stockés en mémoire. On parle de codes redondants.

CODES AUTOVÉRIFICATEURS

- Un code autovérificateur ne permet que la détection des erreurs (si une erreur est détectée alors il faut retransmettre l'information).
- Le *contrôle de parité* est le code autovérificateur le plus simple.
  Il se compose de $m + 1$ bits : les m bits d'informations auquels on ajoute un (m+1)-ème bit, dit de *parité*.
  Sa valeur est telle que le nombre de bits à 1, calculé sur les $m + 1$ bits, est pair.
- Ce contrôle ne va détecter qu'un nombre impair d'erreurs (dans le cas d'un nombre pair, les effets s'annulent).

Exemple :

Soit à coder en ASCII(7 bits) + 1 bit de parité le caractère %.

La table des codes ASCII nous indique : 0100101.

Il y a 3 bits 1, le bit de parité sera 1.

Soit les 8 bits à transmettre : 01001011.

Un code autovérificateur permet de détecter rapidement une erreur :

- Numéro de sécurité sociale : un nombre de 13 chiffres suivi d'une clé à 2 chiffres tel que le nombre total soit un multiple de 97.
- La formule de Luhn pour les cartes bancaires : on double les chiffres de rang pair (le premier est de rang 0) modulo 9 qu'on additionne à ceux de rang impair.
  Le résultat doit être divisible par 10.
- Les code-barres.

Codes autocorrecteurs

- Double parité

1	1	1	1	0	0	1	1
0	1	1	1	0	0	1	0
0	1	0	0	1	1	0	1
0	1	1	1	0	0	0	1
1	0	1	1	1	1	0	

La double parité permet donc de corriger une erreur ou même, dans certains cas un nombre impair d'erreurs.

CODES AUTOCORRECTEURS

Code de Hamming : ce code est basé sur les tests de parités.

- m bits d'informations + k bits de contrôle de parités = n bits.
- Les k bits de parités doivent détecter les n erreurs (+ absence d'erreurs), d'où :

$$2^k \geqslant n + 1$$

Exemple : Si le nombre de bits d'information est 4 (m=4).

$$\begin{aligned} 2^k &\geqslant n+1 \\ 2^k &\geqslant m+k+1 \\ 2^k &\geqslant k+5 \end{aligned}$$

On teste : $2^2 < 2+7$; $2^3 \geqslant 3+5$.

- On construit un code de Hamming sur 7 bits
- les 3 bits de contrôles k_1, k_2, k_3 sont placés sur les puissances de 2 :

7	6	5	4	3	2	1
m_4	m_3	m_2	k_3	m_1	k_2	k_1

Qui contrôle qui ?

7	(0111)	= 4 + 2 + 1	→ 7	est contrôlé par k3, k2, k1
6	(0110)	= 4 + 2	→ 6	est contrôlé par k3, k2
5	(0101)	= 4 + 1	→ 5	est contrôlé par k3, k1
4	(0100)		est le bit contrôle k3	
3	(0011)	= 2 + 1	→ 3	est contrôlé par k2, k1
2	(0010)		est le bit contrôle k2	
1	(0001)		est le bit contrôle k1	

- k_1 contrôle les bits 1, 3, 5, 7
- k_2 contrôle les bits 2, 3, 6, 7
- k_3 contrôle les bits 4, 5, 6, 7

Pour chacun des bits de contrôle, on compare la valeur reçue :

- Si elles sont identiques, on assigne la valeur 0 à la variable binaire A_i associée au bit de contrôle k_i
- sinon, on lui assigne la valeur 1

Code de Hamming : réception d'un message

- Supposons qu'on reçoive le message suivant : 1011100 (n=7, k=3, m=4).

numéro	**7**	**6**	**5**	**4**	**3**	**2**	**1**
type	m4	m3	m2	k3	m1	k2	k1
valeur	1	0	1	1	1	0	0

- $k_1 = 0$ (bit 1,3,5,7) est donc faux => $A_1 = 1$
- $k_2 = 0$ (bit 2,3,6,7) est donc exact => $A_2 = 0$
- $k_3 = 0$ (bit 4,5,6,7) est donc faux => $A_3 = 1$
- L'adresse binaire de l'erreur est $A_3A_2A_1 = 101 = 5$.
  Le bit 5, qui vaut 1, est faux.

Le message initial corrigé est : 1001100 et, si l'on ôte les bits de parité, on obtient les données initiales : 1001.

Calcul simplifié du code de Hamming

Transmission d'un message : Coder 1010 1011 001 avec une parité paire : m=11, donc k=4. Le message à transmettre contient n=15 bits.

numéro	**15**	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**
type	m11	m10	m9	m8	m7	m6	m5	k4	m4	m3	m2	k3	m1	k2	k1
valeur	1	0	1	0	1	0	1	?	1	0	0	?	1	?	?

- Dans le message à transmettre, on a des bits 1 dans les positions suivantes : 15, 13, 11, 9, 7, 3.
- on utilise la Nim-Addition

$$
\begin{array}{rcccc}
15 & = & 1 & 1 & 1 & 1 \\
13 & = & 1 & 1 & 0 & 1 \\
11 & = & 1 & 0 & 1 & 1 \\
9 & = & 1 & 0 & 0 & 1 \\
7 & = & 0 & 1 & 1 & 1 \\
3 & = & 0 & 0 & 1 & 1 \\
\hline
 & & 0 & 1 & 0 & 0 \\
 & & k_4 & k_3 & k_2 & k_1
\end{array}
$$

Le message codé est donc 1010 1010 1001 100.

Réception d'un message On a reçu le message suivant : 1010 0010 1001 100. La parité utilisée est paire. On a des bits 1 dans les positions :

$$
\begin{array}{rcccccl}
15 & = & 1 & 1 & 1 & 1 & \\
13 & = & 1 & 1 & 0 & 1 & \\
9 & = & 1 & 0 & 0 & 1 & \\
7 & = & 0 & 1 & 1 & 1 & \\
4 & = & 0 & 1 & 0 & 0 & \\
3 & = & 0 & 0 & 1 & 1 & \text{Nim-Addition} \\
\hline
 & & 1 & 0 & 1 & 1 & \\
 & & A_4 & A_3 & A_2 & A_1 & \text{erreur à la position } 1011_2 = 11_{10}
\end{array}
$$

Après la correction du bit en position 11, on a le message suivant : 1010 1010 1001 100

Codage des caractères

Plusieurs normes existent pour représenter les caractères en binaire :

- Norme ASCII : caractères codés sur 7 bits (128 valeurs)
- Norme UTF-8 : caractères codés sur 1 à 4 octets (plusieurs milliers de symboles)

Attention à utiliser le même codage pour écrire et pour lire : Ãa m ?Ã c nerve ces chaÃ R nes de caractÃ"res qui dÃ c raillent !

Table des caractères ASCII :

b_7 →					0	0	0	0	1	1	1	1
b_6 →					0	0	1	1	0	0	1	1
b_5 →					0	1	0	1	0	1	0	1
Bits	b_4 ↓	b_3 ↓	b_2 ↓	b_1 ↓	Row ↓ / Column →	0	1	2	3	4	5	6	7
	0	0	0	0	0	NUL	DLE	SP	0	@	P	`	p
	0	0	0	1	1	SOH	DC1	!	1	A	Q	a	q
	0	0	1	0	2	STX	DC2	"	2	B	R	b	r
	0	0	1	1	3	ETX	DC3	#	3	C	S	c	s
	0	1	0	0	4	EOT	DC4	$	4	D	T	d	t
	0	1	0	1	5	ENQ	NAK	%	5	E	U	e	u
	0	1	1	0	6	ACK	SYN	&	6	F	V	f	v
	0	1	1	1	7	BEL	ETB	'	7	G	W	g	w
	1	0	0	0	8	BS	CAN	(	8	H	X	h	x
	1	0	0	1	9	HT	EM	)	9	I	Y	i	y
	1	0	1	0	10	LF	SUB	*	:	J	Z	j	z
	1	0	1	1	11	VT	ESC	+	;	K	[	k	{
	1	1	0	0	12	FF	FC	,	<	L	\	l	|
	1	1	0	1	13	CR	GS	-	=	M	]	m	}
	1	1	1	0	14	SO	RS	.	>	N	^	n	~
	1	1	1	1	15	SI	US	/	?	O	_	o	DEL

Compression de données

« **La compression de données** est l'opération informatique consistant à transformer une suite de bits A en une suite de bits B plus courte pouvant restituer les mêmes informations, ou des informations voisines, en utilisant un algorithme de **décompression**. » *Source Wikipédia*

On distingue deux types de compression :

- **sans perte**, utilisé pour les archives, les fichiers exécutables ou les textes.
- **avec perte**, utilisé pour les images, le son et la vidéo.

Les formats de données tels que Zip, RAR, gzip, MP3 et JPEG utilisent des algorithmes de compression de données.

– jpeg 18 Ko

– jpeg 271 Ko

– png 361 Ko

Le codage de Huffman

Le codage de Huffman est un codage sans perte, préfixe à longueur variable.
Ce type de codage est utilisé dans tous les formats de compression usuels (jpeg, png, mp3...).
Le code ASCII n'est pas performant car tous les caractères sont codés avec 7 bits.

- Consiste à remplacer les caractères les plus fréquents par des codes courts et les caractères les moins fréquents par des codes longs (Le Morse repose déjà sur ce principe).
- Utilise la notion de code préfixe, c'est-à-dire qu'il est inutile de rajouter des caractères entre les mots (contrairement au Morse (blanc)). C'est la construction de l'arbre où tous les noeuds internes ont exactement 2 successeurs qui justifie cela. Aucun des mots ne commence par un des mots.

Algorithme de compression :

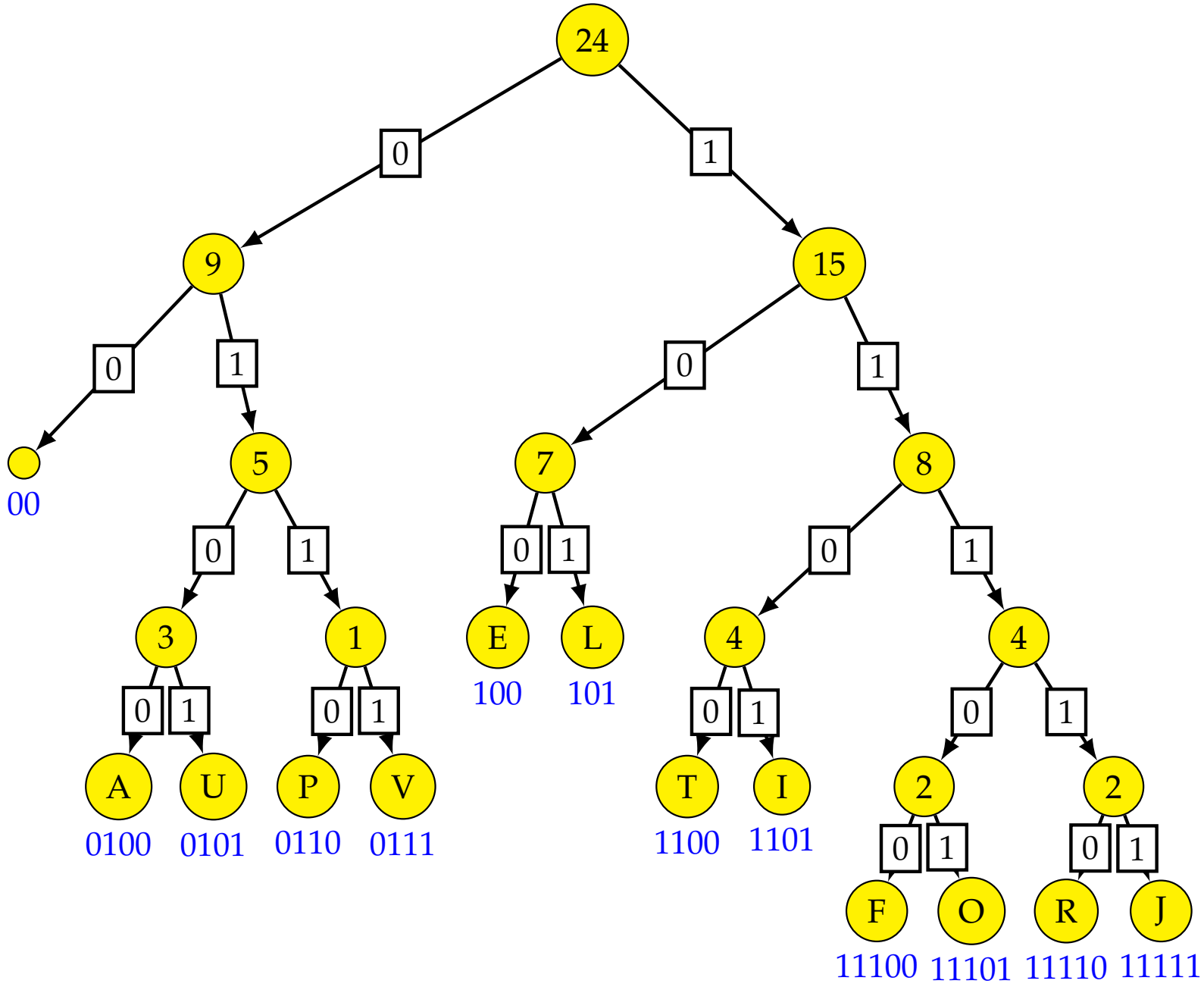
- On cherche les fréquences des caractères,
- on trie les caractères par ordre croissant de fréquence,
- on construit un arbre pour donner le code binaire de chaque caractère.

LA JOLIE PETITE FLEUR VA

- Fréquence des lettres :

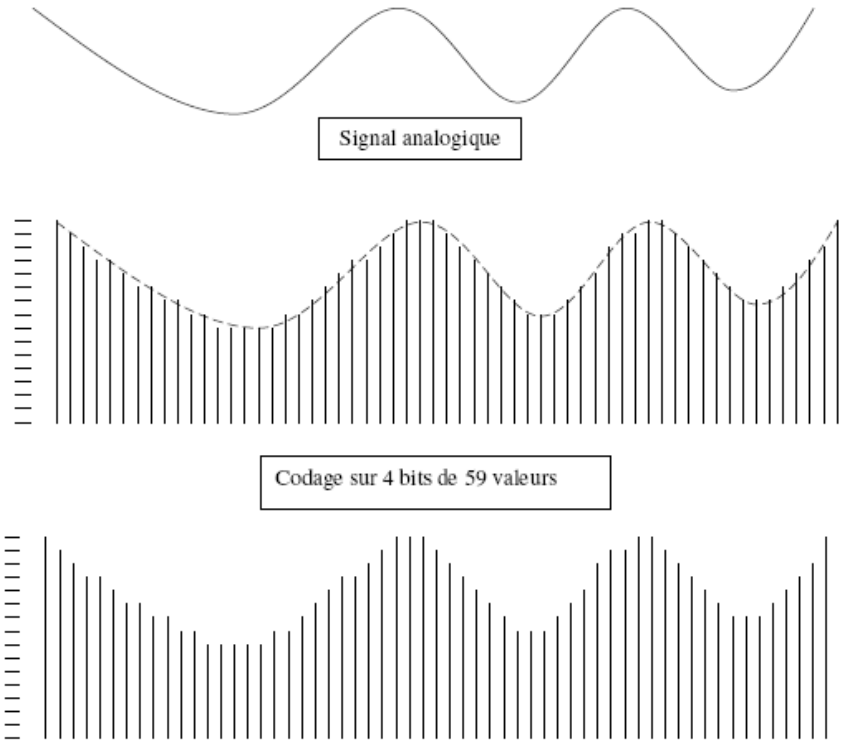
E	_	L	A	T	I	P	R	U	F	O	V	J
4	4	3	2	2	2	1	1	1	1	1	1	1

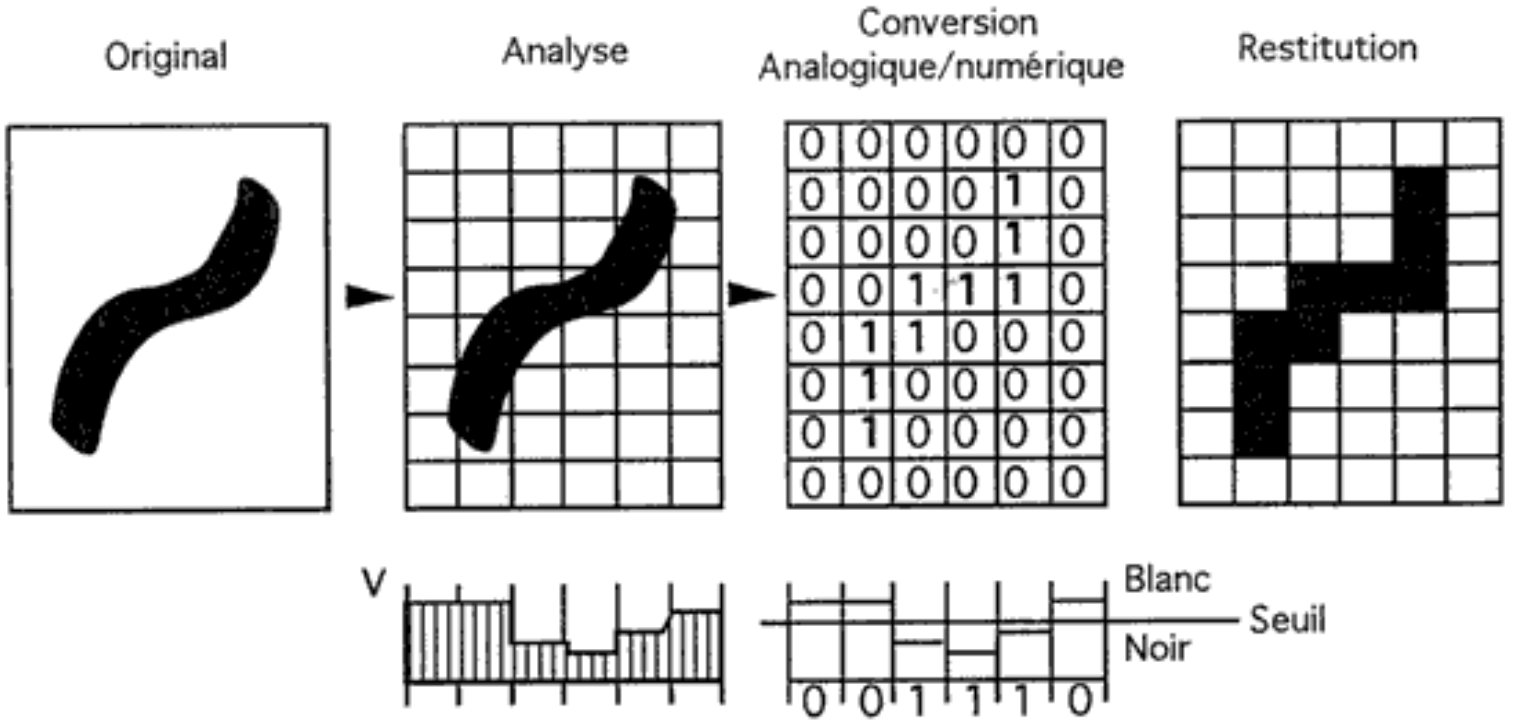
On peut maintenant construire l'arbre de HUFFMAN


24
0
1
9
15
0
1
0
1
00
5
7
8
0
1
0
1
0
1
3
1
E
L
4
4
0
1
0
1
0
1
0
1
A
U
P
V
100
101
T
I
2
2
0100
0101
0110
0111
1100
1101
0
1
0
1
F
O
R
J
11100
11101
11110
11111

SONS ET IMAGES

Pour rendre numérique une donnée analogique comme le son ou une image, on utilise en général, l'échantillonnage.




Original
Analyse
Conversion
Analogique/numérique
Restitution
0 0 0 0 0 0
0 0 0 0 1 0
0 0 0 0 1 0
0 0 1 1 1 0
0 1 1 0 0 0
0 1 0 0 0 0
0 1 0 0 0 0
0 0 0 0 0 0
V
Blanc
Seuil
Noir
0 0 1 1 1 0

The Magic Tricks

1	3	5	7	9	11	13	15
17	19	21	23	25	27	29	31
33	35	37	39	41	43	45	47
49	51	53	55	57	59	61	63

2	3	6	7	10	11	14	15
18	19	22	23	26	27	30	31
34	35	38	39	42	43	46	47
50	51	54	55	58	59	62	63

4	5	6	7	12	13	14	15
20	21	22	23	28	29	30	31
36	37	38	39	44	45	46	47
52	53	54	55	60	61	62	63

8	9	10	11	12	13	14	15
24	25	26	27	28	29	30	31
40	41	42	43	44	45	46	47
56	57	58	59	60	61	62	63

16	17	18	19	20	21	22	23
24	25	26	27	28	29	30	31
48	49	50	51	52	53	54	55
56	57	58	59	60	61	62	63

32	33	34	35	36	37	38	39
40	41	42	43	44	45	46	47
48	49	50	51	52	53	54	55
56	57	58	59	60	61	62	63

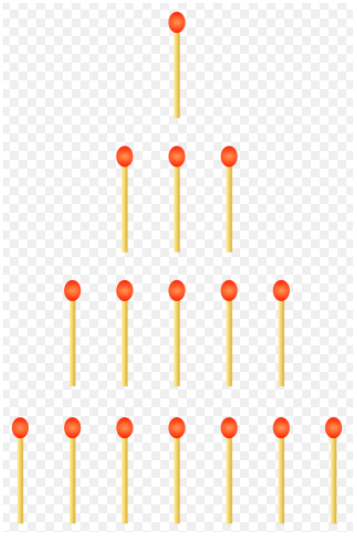
Le jeu de Marienbad

Une utilisation ludique du binaire et pourquoi pas une idée de projet....

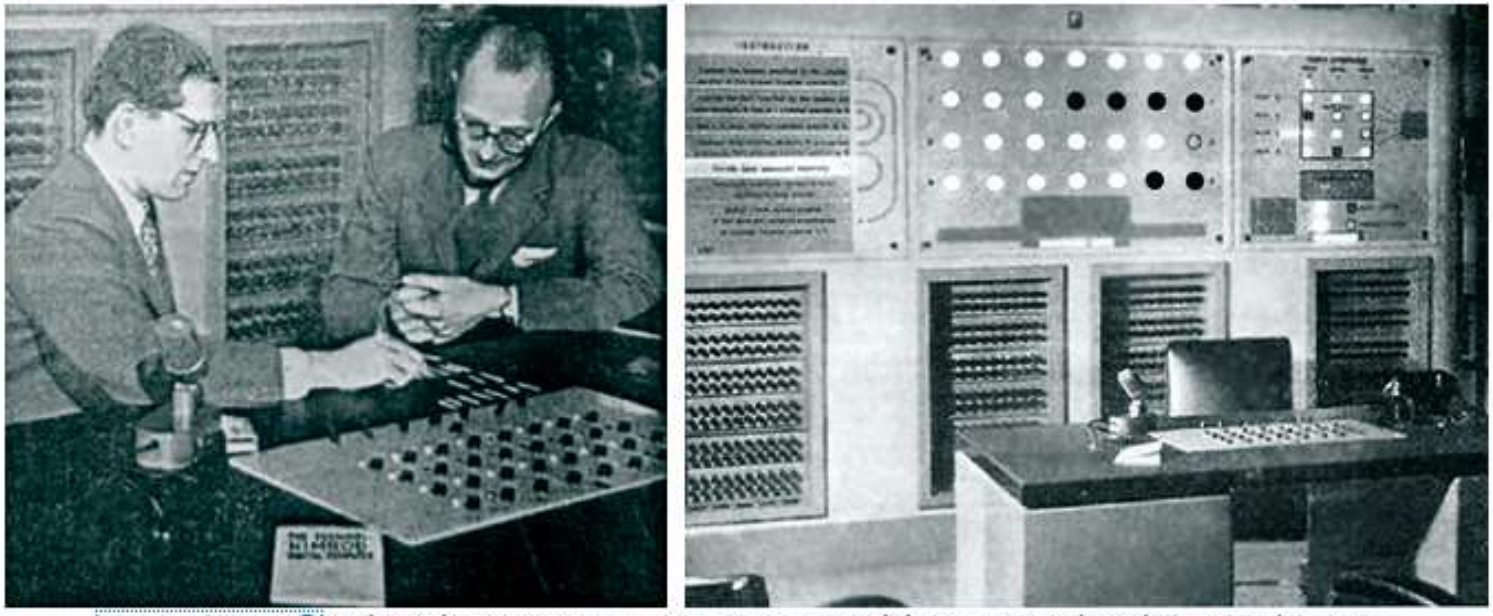
Ce jeu, qui apparaît dans le film l'année dernière à Marienbad, s'appelle aussi le jeu de NIM.
Connu depuis longtemps, il s'appelle fan tan en Chine et tiouk tiouk en Afrique.

Voici la règle du jeu :

- Au départ, des allumettes sont réparties en plusieurs tas (ou piles).
- Deux personnes vont jouer à tour de rôle. Chacune peut enlever un nombre quelconque d'allumettes (au moins une) dans un et un seul tas de son choix.
- Le joueur gagnant est celui qui enlève la dernière allumette.



STRATÉGIE GAGNANTE



Machine *Nimrod* spécialisée dans le *Nim*, construite par la Société *Ferranti* et présentée en 1951 à Londres.

– Le NIMROD

La Nim-Addition

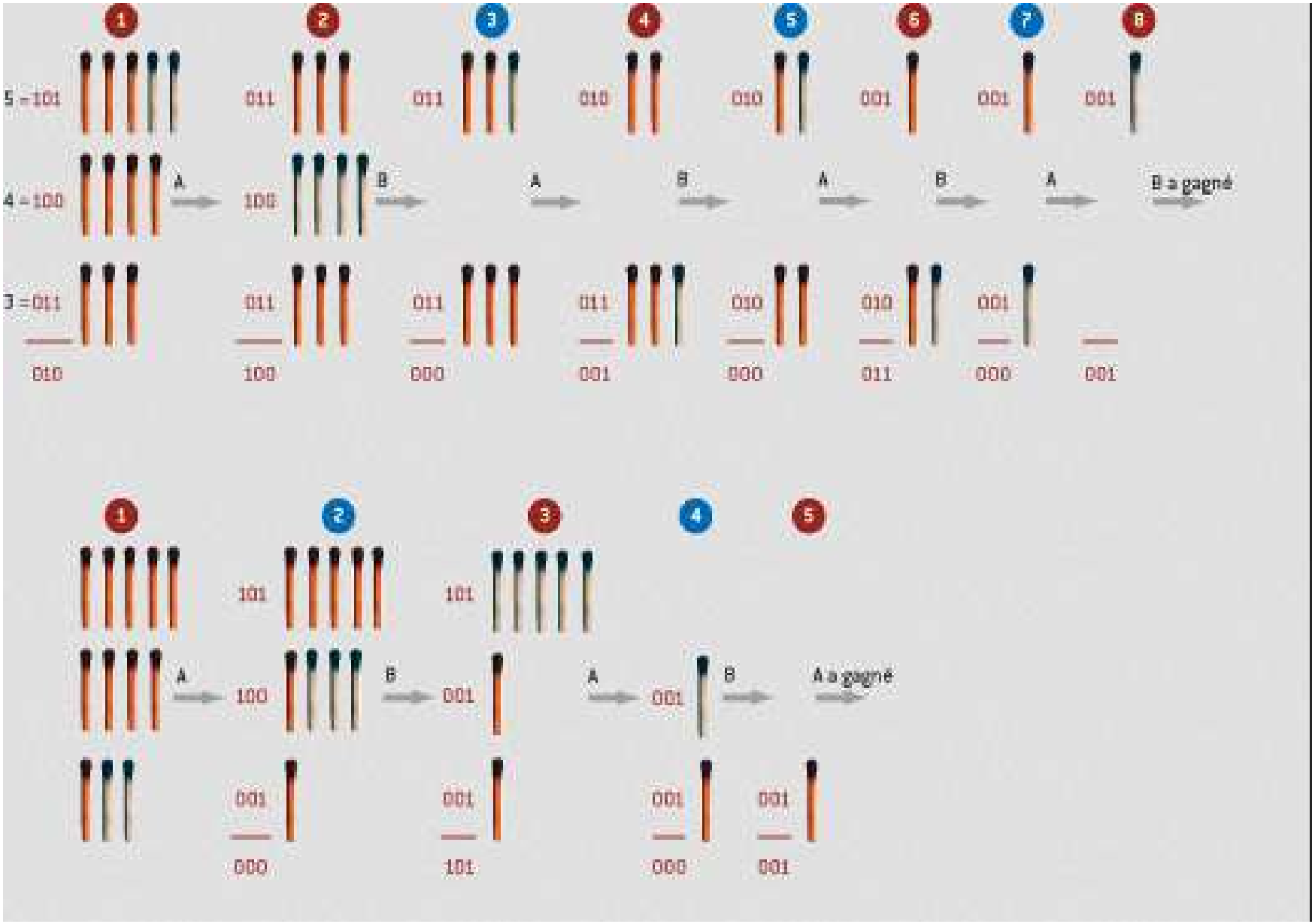
La Nim-addition [4] consiste à disposer les chiffres binaires comme on le fait quand on pose une addition habituelle, puis à additionner en colonnes, sans tenir compte des retenues. Le nombre binaire obtenu retraduit en entier est le résultat :

$$\begin{array}{rl} & 1011 (11) \\ + & 1001 \ \ (9) \\ + & \ \ 111 \ \ (7) \\ \hline = & 0101 \ \ (5) \end{array}$$

Pour gagner, il suffit de se retrouver en position de Nim-Addition nulle. Toutes les configurations de départs ne sont donc pas gagnantes.

4. On retrouve cette Nim-Addition dans le théorème de Sprague-Grundy et dans toute la théorie des jeux combinatoires

UN EXEMPLE


1
2
3
4
5
6
7
8
5 = 101
011
011
010
010
001
001
001
4 = 100
A
100
B
A
B
A
B
A
B a gagné
3 = 011
011
011
011
010
010
001
010
100
000
001
000
011
000
001
1
2
3
4
5
101
101
A
100
B
001
A
001
B
A a gagné
001
001
001
001
000
101
000
001