http://ow.ly/2xKdB

Chapitre 3

Codage de l'information

3.1. Vocabulaire

Quelle que soit la nature de l'information traitée par un ordinateur (image, son, texte, vidéo), elle l'est toujours sous la forme d'un ensemble de nombres écrits en base 2, par exemple 01001011.

Le terme bit (b minuscule dans les notations) signifie « binary digit », c'est-à-dire 0 ou 1 en numérotation binaire. Il s'agit de la plus petite unité d'information manipulable par une machine numérique. Il est possible de représenter physiquement cette information binaire par un signal électrique ou magnétique, qui, au-delà d'un certain seuil, correspond à la valeur 1.

L'octet (en anglais *byte* ou B majuscule dans les notations) est une unité d'information composée de 8 bits. Il permet par exemple de stocker un caractère comme une lettre ou un chiffre.

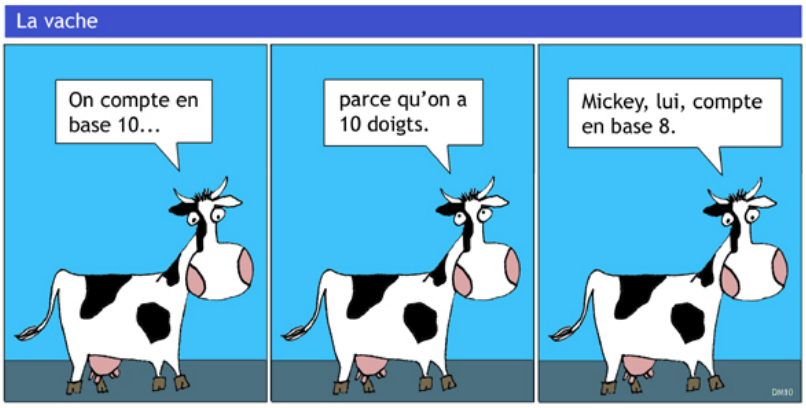
Une unité d'information composée de 16 bits est généralement appelée mot (en anglais *word*).

Une unité d'information de 32 bits de longueur est appelée mot double (en anglais *double word*, d'où l'appellation *dword*).

Beaucoup d'informaticiens ont appris que 1 kilooctet valait 1024 octets. Or, depuis décembre 1998, l'organisme international *IEC* a statué sur la question[1]. Voici les unités standardisées :

- Un kilooctet (ko) = 1000 octets
- Un mégaoctet (Mo) = 10^6 octets
- Un gigaoctet (Go) = 10^9 octets
- Un téraoctet (To) = 10^{12} octets
- Un pétaoctet (Po) = 10^{15} octets

3.2. Les bases décimale, binaire et hexadécimale



Nous utilisons le système décimal (base 10) dans nos activités quotidiennes. Ce système est basé sur dix symboles, de 0 à 9, avec une unité supérieure (dizaine, centaine, etc.) à chaque fois que dix

1 http://physics.nist.gov/cuu/Units/binary.html

unités sont comptabilisées. C'est un système positionnel, c'est-à-dire que l'endroit où se trouve le symbole définit sa valeur. Ainsi, le 2 de 523 n'a pas la même valeur que le 2 de 132. En fait, 523 est l'abréviation de $5 \cdot 10^2 + 2 \cdot 10^1 + 3 \cdot 10^0$. On peut selon ce principe imaginer une infinité de systèmes numériques fondés sur des bases différentes.

En informatique, outre la base 10, on utilise très fréquemment le système binaire (base 2) puisque l'algèbre booléenne est à la base de l'électronique numérique. Deux symboles suffisent : 0 et 1.

On utilise aussi très souvent le système hexadécimal (base 16) du fait de sa simplicité d'utilisation et de représentation pour les mots machines (il est bien plus simple d'utilisation que le binaire). Il faut alors six symboles supplémentaires : A (qui représente le 10), B (11), C (12), D (13), E (14) et F (15)

Le tableau ci-dessous montre la représentation des nombres de 0 à 15 dans les bases 10, 2 et 16.

Décimal	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Binaire	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
Hexadécimal	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F

3.2.1. Conversion décimal - binaire

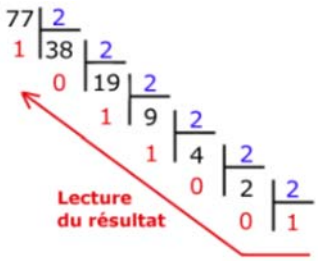
Convertissons 01001101 en décimal à l'aide du schéma ci-dessous :

$$2^7 \quad 2^6 \quad 2^5 \quad 2^4 \quad 2^3 \quad 2^2 \quad 2^1 \quad 2^0$$

$$0 \quad 1 \quad 0 \quad 0 \quad 1 \quad 1 \quad 0 \quad 1$$

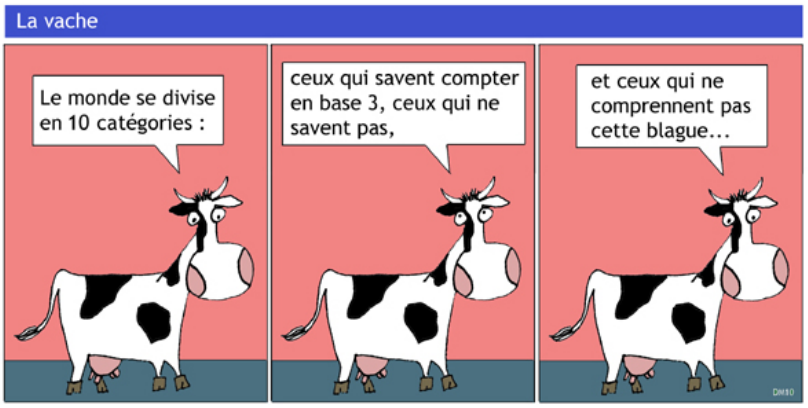
Le nombre en base 10 est $2^6 + 2^3 + 2^2 + 2^0 = 64 + 8 + 4 + 1 = 77$.

Allons maintenant dans l'autre sens et écrivons 77 en base 2. Il s'agit de faire une suite de divisions euclidiennes par 2. Le résultat sera la juxtaposition des restes. Le schéma ci-dessous explique la méthode mieux qu'un long discours :



77 s'écrit donc en base 2 : 1001101.

Si on l'écrit sur un octet, cela donne : 01001101.



3.2.2. Conversion hexadécimal - binaire

Pour convertir un nombre binaire en hexadécimal, il suffit de faire des groupes de quatre bits (en commençant depuis la droite). Par exemple, convertissons 1001101 :

Binaire	0100	1101
Pseudo-décimal	4	13
Hexadécimal	4	D

1001101 s'écrit donc en base 16 : 4D.
Pour convertir d'hexadécimal en binaire, il suffit de lire ce tableau de bas en haut.

Exercice 3.1

Donnez la méthode pour passer de la base décimale à la base hexadécimale (dans les deux sens).

Exercice 3.2

Complétez le tableau ci-dessous. L'indice indique la base dans laquelle le nombre est écrit.

	Bases		
	2	10	16
1001010110_2			
2002_{10}			
$A1C4_{16}$			

Exercice 3.3

Écrivez en Python un programme permettant de convertir un nombre d'une base de départ *d* vers une base d'arrivée *a* (*d* et *a* compris entre 2 et 16).

3.3. Représentation des nombres entiers

3.3.1. Représentation d'un entier naturel

Un entier naturel est un nombre entier positif ou nul. Le choix à faire (c'est-à-dire le nombre de bits à utiliser) dépend de la fourchette des nombres que l'on désire utiliser. Pour coder des nombres entiers naturels compris entre 0 et 255, il nous suffira de 8 bits (un octet) car 2^8=256. D'une manière générale un codage sur *n* bits pourra permettre de représenter des nombres entiers naturels compris entre 0 et 2^n-1.

Exemples : $9 = 00000101_2$, $128 = 10000000_2$, etc.

3.3.2. Représentation d'un entier relatif

Un entier relatif est un entier pouvant être négatif. Il faut donc coder le nombre de telle façon que l'on puisse savoir s'il s'agit d'un nombre positif ou d'un nombre négatif, et il faut de plus que les règles d'addition soient conservées. L'astuce consiste à utiliser un codage que l'on appelle complément à deux. Cette représentation permet d'effectuer les opérations arithmétiques usuelles naturellement.

- **Un entier relatif positif** ou nul sera représenté en binaire (base 2) comme un entier naturel, à la seule différence que le bit de poids fort (le bit situé à l'extrême gauche) représente le signe. Il faut donc s'assurer pour un entier positif ou nul qu'il est à zéro (0 correspond à un signe positif, 1 à un signe négatif). Ainsi, si on code un entier naturel sur 4 bits, le nombre le plus grand sera 0111 (c'est-à-dire 7 en base décimale).
  - Sur 8 bits (1 octet), l'intervalle de codage est [−128, 127].
  - Sur 16 bits (2 octets), l'intervalle de codage est [−32768, 32767].
  - Sur 32 bits (4 octets), l'intervalle de codage est [−2147483648, 2147483647].

  D'une manière générale le plus grand entier relatif positif codé sur n bits sera $2^{n-1}-1$.

- **Un entier relatif négatif** sera représenté grâce au codage en complément à deux.

Principe du complément à deux

1. Écrire la valeur absolue du nombre en base 2. Le bit de poids fort doit être égal à 0.
2. Inverser les bits : les 0 deviennent des 1 et vice versa. On fait ce qu'on appelle le complément à un.
3. On ajoute 1 au résultat (les dépassements sont ignorés).

Cette opération correspond au calcul de $2^n-|x|$, où n est la longueur de la représentation et $|x|$ la valeur absolue du nombre à coder.

Ainsi −1 s'écrit comme $256-1=255=11111111_2$, pour les nombres sur 8 bits.

Exemple

On désire coder la valeur −19 sur 8 bits. Il suffit :

1. d'écrire 19 en binaire : 00010011
2. d'écrire son complément à 1 : 11101100
3. et d'ajouter 1 : 11101101

La représentation binaire de −19 sur 8 bits est donc 11101101.

On remarquera qu'en additionnant un nombre et son complément à deux on obtient 0. En effet, 00010011 + 11101101 = 00000000 (avec une retenue de 1 qui est éliminée).

Truc

Pour transformer de tête un nombre binaire en son complément à deux, on parcourt le nombre de droite à gauche en laissant inchangés les bits jusqu'au premier 1 (compris), puis on inverse tous les bits suivants. Prenons comme exemple le nombre 20 : 00010100.

1. On garde la partie à droite telle quelle : 00010**100**
2. On inverse la partie de gauche après le premier un : **11101**100
3. Et voici −20 : 11101100

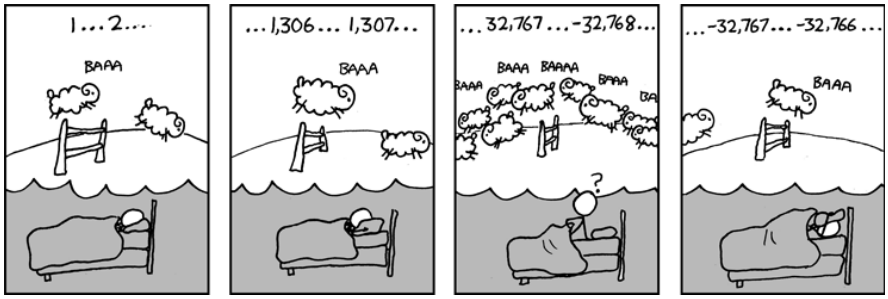
Le 4 juin 1996, une fusée Ariane 5 a explosé 40 secondes après l'allumage. La fusée et son chargement avaient coûté 500 millions de dollars. La commission d'enquête a rendu son rapport au bout de deux semaines. Il s'agissait d'une erreur de programmation dans le système inertiel de référence. À un moment donné, un nombre codé en virgule flottante sur 64 bits (qui représentait la vitesse horizontale de la fusée par rapport à la plate-forme de tir) était converti en un entier sur 16 bits. Malheureusement, le nombre en question était plus grand que 32768 (le plus grand entier que l'on peut coder sur 16 bits) et la conversion a été incorrecte.

Exercice 3.4

1. Codez les entiers relatifs suivants sur 8 bits (16 si nécessaire) : 456, −1, −56, −5642.
2. Que valent en base dix les trois entiers relatifs suivants :
   01101100
   11101101
   1010101010101010 ?

Exercice 3.5

Expliquez ce rêve étrange (source de l'image : http://xkcd.com/571).



3.4. Représentation des nombres réels

En base 10, l'expression 652,375 est une manière abrégée d'écrire :
$6 \cdot 10^2 + 5 \cdot 10^1 + 2 \cdot 10^0 + 3 \cdot 10^{-1} + 7 \cdot 10^{-2} + 5 \cdot 10^{-3}$.
Il en va de même pour la base 2. Ainsi, l'expression 110,101 signifie :
$1 \cdot 2^2 + 1 \cdot 2^1 + 0 \cdot 2^0 + 1 \cdot 2^{-1} + 0 \cdot 2^{-2} + 1 \cdot 2^{-3}$.

3.4.1. Conversion de binaire en décimal

On peut ainsi facilement convertir un nombre réel de la base 2 vers la base 10. Par exemple :
$110{,}101_2 = 1 \cdot 2^2 + 1 \cdot 2^1 + 0 \cdot 2^0 + 1 \cdot 2^{-1} + 0 \cdot 2^{-2} + 1 \cdot 2^{-3} = 4 + 2 + 0{,}5 + 0{,}125 = 6{,}625_{10}$.

Exercice 3.6

Transformez $0{,}0101010101_2$ en base 10.
Transformez $11100{,}10001_2$ en base 10.

3.4.2. Conversion de décimal en binaire

Le passage de base 10 en base 2 est plus subtil. Par exemple : convertissons 1234,347 en base 2.

- La partie entière se transforme comme au § 3.2.1 : $1234_{10} = 10011010010_2$
- On transforme la partie décimale selon le schéma suivant :

0,347·2 = **0**,694	0,347 = 0,**0**...
0,694·2 = **1**,388	0,347 = 0,0**1**...
0,388·2 = **0**,766	0,347 = 0,01**0**...
0,766·2 = **1**,552	0,347 = 0,010**1**...
0,552·2 = **1**,104	0.347 = 0,0101**1**...
0,104·2 = **0**,208	0,347 = 0,01011**0**...
0,208·2 = **0**,416	0,347 = 0,010110**0**...
0,416·2 = **0**,832	0,347 = 0,0101100**0**...
0,832·2 = **1**,664	0,347 = 0,01011000**1**...
0,664·2 = **1**,328	0,347 = 0,010110001**1**...
0,328·2 = **0**,656	0,347 = 0,0101100011**0**...

On continue ainsi jusqu'à la précision désirée...
Attention ! Un nombre à développement décimal fini en base 10 ne l'est pas forcément en base 2.

Exercice 3.7

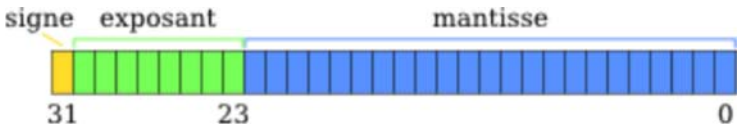
Transformez $0{,}5625_{10}$ en base 2.
Transformez $0{,}15_{10}$ en base 2.
Transformer $12{,}9_{10}$ en base 2.

3.4.2. La norme IEEE 754

IEEE, que l'on peut prononcer « i 3 e » : *Institute of Electrical and Electronics Engineers*

La norme IEEE 754 définit la façon de coder un nombre réel. Cette norme se propose de coder le nombre sur 32 bits et définit trois composantes :

- le signe est représenté par un seul bit, le bit de poids fort
- l'exposant est codé sur les 8 bits consécutifs au signe
- la mantisse (les bits situés après la virgule) sur les 23 bits restants



Certaines conditions sont toutefois à respecter pour les exposants :

- l'exposant 00000000 est interdit.
- l'exposant 11111111 est interdit. On s'en sert toutefois pour signaler des erreurs, on appelle alors cette configuration du nombre NaN, ce qui signifie « Not a number ».
- il faut rajouter 127 (01111111) à l'exposant pour une conversion de décimal vers un nombre réel binaire. Les exposants peuvent ainsi aller de −254 à 255.

La formule d'expression des nombres réels est ainsi la suivante :

$$(-1)^S * 2^(E - 127) * (1 + F)$$

- S est le bit de signe et l'on comprend alors pourquoi 0 est positif (−1^0=1),
- E est l'exposant auquel on doit bien ajouter 127 pour obtenir son équivalent codé,
- F est la partie fractionnaire.

Exemple

Traduisons en binaire, en utilisant la norme IEEE 754, le nombre −6,625.

- Codons d'abord la valeur absolue en binaire : $6{,}625_{10} = 110{,}1010_2$
- Nous mettons ce nombre sous la forme : **1, *partie fractionnaire***
  $110{,}1010 = 1{,}\mathbf{101010} \cdot 2^2$ (2^2 décale la virgule de 2 chiffres vers la droite)
- La partie fractionnaire étendue sur 23 bits est donc **101 0100 0000 0000 0000 0000.**
- **Exposant** = $127 + 2 = 129_{10} = 1000\ 0001_2$

Signe Exposant Mantisse

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

En hexadécimal : C0 D4 00 00.

Le 25 février 1991, pendant la Guerre du Golfe, une batterie américaine de missiles Patriot, à Dharan (Arabie Saoudite), a échoué dans l'interception d'un missile Scud irakien. Le Scud a frappé un baraquement de l'armée américaine et a tué 28 soldats. La commission d'enquête a conclu à un calcul incorrect du temps de parcours, dû à un problème d'arrondi. Les nombres étaient représentés en virgule fixe sur 24 bits. Le temps était compté par l'horloge interne du système en dixième de seconde. Malheureusement, 1/10 n'a pas d'écriture finie dans le système binaire : 1/10 = 0,1 (dans le système décimal) = 0,0001100110011001100110011... (dans le système binaire). L'ordinateur de bord arrondissait 1/10 à 24 chiffres, d'où une petite erreur dans le décompte du temps pour chaque 1/10 de seconde. Au moment de l'attaque, la batterie de missile Patriot était allumée depuis environ 100 heures, ce qui a entraîné une accumulation des erreurs d'arrondi de 0,34 s. Pendant ce temps, un missile Scud parcourt environ 500 m, ce qui explique que le Patriot soit passé à côté de sa cible.

3.5. Le code ASCII

La norme ASCII[2] (on prononce généralement « aski ») établit une correspondance entre une représentation binaire des caractères de l'alphabet latin et les symboles, les signes, qui constituent cet alphabet. Par exemple, le caractère *a* est associé à 1100001 (97) et *A* à 1000001 (65).

Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char
0	00	Null	32	20	Space	64	40	@	96	60	`
1	01	Start of heading	33	21	!	65	41	A	97	61	a
2	02	Start of text	34	22	"	66	42	B	98	62	b
3	03	End of text	35	23	#	67	43	C	99	63	c
4	04	End of transmit	36	24	$	68	44	D	100	64	d
5	05	Enquiry	37	25	%	69	45	E	101	65	e
6	06	Acknowledge	38	26	&	70	46	F	102	66	f
7	07	Audible bell	39	27	'	71	47	G	103	67	g
8	08	Backspace	40	28	(	72	48	H	104	68	h
9	09	Horizontal tab	41	29	)	73	49	I	105	69	i
10	0A	Line feed	42	2A	*	74	4A	J	106	6A	j
11	0B	Vertical tab	43	2B	+	75	4B	K	107	6B	k
12	0C	Form feed	44	2C	,	76	4C	L	108	6C	l
13	0D	Carriage return	45	2D	-	77	4D	M	109	6D	m
14	0E	Shift out	46	2E	.	78	4E	N	110	6E	n
15	0F	Shift in	47	2F	/	79	4F	O	111	6F	o
16	10	Data link escape	48	30	0	80	50	P	112	70	p
17	11	Device control 1	49	31	1	81	51	Q	113	71	q
18	12	Device control 2	50	32	2	82	52	R	114	72	r
19	13	Device control 3	51	33	3	83	53	S	115	73	s
20	14	Device control 4	52	34	4	84	54	T	116	74	t
21	15	Neg. acknowledge	53	35	5	85	55	U	117	75	u
22	16	Synchronous idle	54	36	6	86	56	V	118	76	v
23	17	End trans. block	55	37	7	87	57	W	119	77	w
24	18	Cancel	56	38	8	88	58	X	120	78	x
25	19	End of medium	57	39	9	89	59	Y	121	79	y
26	1A	Substitution	58	3A	:	90	5A	Z	122	7A	z
27	1B	Escape	59	3B	;	91	5B	[	123	7B	{
28	1C	File separator	60	3C	<	92	5C	\	124	7C	|
29	1D	Group separator	61	3D	=	93	5D	]	125	7D	}
30	1E	Record separator	62	3E	>	94	5E	^	126	7E	~
31	1F	Unit separator	63	3F	?	95	5F	_	127	7F	□

La norme ASCII permet ainsi à toutes sortes de machines de stocker, analyser et communiquer de l'information textuelle. En particulier, la quasi-totalité des ordinateurs personnels et des stations de travail utilisent l'encodage ASCII. Le code ASCII de base représentait les caractères sur 7 bits (c'est-à-dire 128 caractères possibles, de 0 à 127).

- Les codes 0 à 31 ne sont pas des caractères. On les appelle caractères de contrôle car ils permettent de faire des actions telles que :
  - retour à la ligne (*Carriage return*)
  - bip sonore (*Audible bell*)
- Les codes 65 à 90 représentent les majuscules
- Les codes 97 à 122 représentent les minuscules (il suffit de modifier le 6ème bit pour passer de majuscules à minuscules, c'est-à-dire ajouter 32 au code ASCII en base décimale).

2 American Standard Code for Information Interchange

Le code ASCII a été mis au point pour la langue anglaise, il ne contient donc pas de caractères accentués, ni de caractères spécifiques à une langue. Le code ASCII a donc été étendu à 8 bits pour pouvoir coder plus de caractères (on parle d'ailleurs de code ASCII étendu...). Cette norme s'appelle ISO-8859 et se décline par exemple en ISO-8859-1 lorsqu'elle étend l'ASCII avec les caractères accentués d'Europe occidentale, et qui est souvent appelée Latin-1 ou Europe occidentale.

Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char
128	80	Ç	160	A0	á	192	C0	└	224	E0	α
129	81	ü	161	A1	í	193	C1	┴	225	E1	ß
130	82	é	162	A2	ó	194	C2	┬	226	E2	Γ
131	83	â	163	A3	ú	195	C3	├	227	E3	π
132	84	ä	164	A4	ñ	196	C4	─	228	E4	Σ
133	85	à	165	A5	Ñ	197	C5	┼	229	E5	σ
134	86	å	166	A6	ª	198	C6	╞	230	E6	µ
135	87	ç	167	A7	º	199	C7	╟	231	E7	τ
136	88	ê	168	A8	¿	200	C8	╚	232	E8	Φ
137	89	ë	169	A9	⌐	201	C9	╔	233	E9	Θ
138	8A	è	170	AA	¬	202	CA	╩	234	EA	Ω
139	8B	ï	171	AB	½	203	CB	╦	235	EB	δ
140	8C	î	172	AC	¼	204	CC	╠	236	EC	∞
141	8D	ì	173	AD	¡	205	CD	═	237	ED	ø
142	8E	Ä	174	AE	«	206	CE	╬	238	EE	ε
143	8F	Å	175	AF	»	207	CF	╧	239	EF	∩
144	90	É	176	B0	░	208	D0	╨	240	F0	≡
145	91	æ	177	B1	▒	209	D1	╤	241	F1	±
146	92	Æ	178	B2	▓	210	D2	╥	242	F2	≥
147	93	ô	179	B3	│	211	D3	╙	243	F3	≤
148	94	ö	180	B4	┤	212	D4	╘	244	F4	⌠
149	95	ò	181	B5	╡	213	D5	╒	245	F5	⌡
150	96	û	182	B6	╢	214	D6	╓	246	F6	÷
151	97	ù	183	B7	╖	215	D7	╫	247	F7	≈
152	98	ÿ	184	B8	╕	216	D8	╪	248	F8	°
153	99	Ö	185	B9	╣	217	D9	┘	249	F9	∙
154	9A	Ü	186	BA	║	218	DA	┌	250	FA	·
155	9B	¢	187	BB	╗	219	DB	█	251	FB	√
156	9C	£	188	BC	╝	220	DC	▄	252	FC	ⁿ
157	9D	¥	189	BD	╜	221	DD	▌	253	FD	²
158	9E	₧	190	BE	╛	222	DE	▐	254	FE	■
159	9F	ƒ	191	BF	┐	223	DF	▀	255	FF	

Il existe d'autres normes que l'ASCII, comme l'Unicode par exemple, qui présentent l'avantage de proposer une version unifiée des différents encodages de caractères complétant l'ASCII mais aussi de permettre l'encodage de caractères autres que ceux de l'alphabet latin. Unicode définit des dizaines de milliers de codes, mais les 128 premiers restent compatibles avec ASCII.

Dans le codage UTF-8, chaque point de code est codé sur une suite d'un à quatre octets. Il a été conçu pour être compatible avec certains logiciels originellement prévus pour traiter des caractères d'un seul octet.

Toutes ces normes différentes et leurs incompatibilités partielles sont la cause des problèmes que l'on rencontre parfois avec les caractères accentués. C'est pour cette raison qu'il vaut mieux, quand on écrit des courriels à l'étranger, n'utiliser que des caractères non accentués.

3.6. Codes correcteurs d'erreurs

Un code correcteur est une technique de codage basée sur la redondance. Elle est destinée à corriger les erreurs de transmission d'un message sur une voie de communication peu fiable.

La théorie des codes correcteurs ne se limite pas qu'aux communications classiques (radio, câble coaxial, fibre optique, etc.) mais également aux supports de stockage comme les disques compacts, la mémoire RAM et d'autres applications où l'intégrité des données est importante.

Comment détecter et/ou corriger des erreurs ?

On peut transmettre un nombre soit en chiffres, soit en lettres :

1. On envoie « 0324614103 ». S'il y a des erreurs de transmission, par exemple si je reçois « 0323614203 », je ne peux pas les détecter.
2. On envoie « zéro trente-deux quatre cent soixante et un quarante et un zéro trois ». S'il y a des erreurs de transmission, par exemple si je reçois « zérb trente-deu quate cent soixante en un quaranhe et on zéro tros », je suis capable de corriger les erreurs et de retrouver le bon numéro.

Dans le premier cas, l'information est la plus concise possible. Dans le deuxième cas au contraire, le message contient plus d'informations que nécessaire. C'est cette redondance qui permet la détection et la correction d'erreurs.

Pourquoi ces codes ?

- Des canaux de transmission imparfaits entraînant des erreurs lors des échanges de données.
- La probabilité d'erreur sur une ligne téléphonique est de 10^{-7} (cela peut même atteindre 10^{-4}). Avec un taux d'erreur de 10^{-6} et une connexion à 1 Mo/s, en moyenne 8 bits erronés sont transmis chaque seconde...

Principe général

- Chaque suite de bits à transmettre est augmentée par une autre suite de bits dite « de redondance » ou « de contrôle ».
- Pour chaque suite de k bits transmise, on ajoute r bits. On dit alors que l'on utilise un code $C(n, k)$ avec $n = k + r$.
- À la réception, les bits ajoutés permettent d'effectuer des contrôles.

Un exemple : le code ISBN

L'ISBN (*International Standard Book Number*) est un numéro international qui permet d'identifier, de manière unique, chaque livre publié. Il est destiné à simplifier la gestion informatique des livres dans les bibliothèques, librairies, etc.

Le numéro ISBN-10 se compose de quatre segments, trois segments de longueur variable et un segment de longueur fixe, la longueur totale de l'ISBN comprend dix chiffres (le 1er janvier 2007, la longueur a été étendue à 13 chiffres en ajoutant un groupe initial de 3 chiffres).

Si les quatre segments d'un ancien code ISBN à 10 chiffres sont notés A - B - C - D :

- A identifie un groupe de codes pour un pays, une zone géographique ou une zone de langue.
- B identifie l'éditeur de la publication.
- C correspond au numéro d'ordre de l'ouvrage chez l'éditeur.
- D est un chiffre-clé calculé à partir des chiffres précédents et qui permet de vérifier qu'il n'y a pas d'erreurs. Outre les chiffres de 0 à 9, cette clé de contrôle peut prendre la valeur X, qui représente le nombre 10.

Calcul du chiffre-clé d'un numéro ISBN-10

- On attribue une pondération à chaque position (de 10 à 2 en allant en sens décroissant) et on fait la somme des produits ainsi obtenus.
- On conserve le reste de la division euclidienne de ce nombre par 11. La clé s'obtient en retranchant ce nombre à 11. Si le reste de la division euclidienne est 0, la clé de contrôle n'est pas 11 (11 - 0 = 11) mais 0.
- De même, si le reste de la division euclidienne est 1, la clé de contrôle n'est pas 10 mais la lettre X.

Le nombre 11 étant premier, une erreur portant sur un chiffre entraînera automatiquement une incohérence du code de contrôle. La vérification du code de contrôle peut se faire en effectuant le même calcul sur le code ISBN complet, en appliquant la pondération 1 au dixième chiffre de la clé de contrôle (si ce chiffre-clé est *X*, on lui attribue la valeur 10) : la somme pondérée doit alors être un multiple de 11.

Exemple

Pour le numéro ISBN (à 9 chiffres) 2-35288-041, quelle est la clé de contrôle ?

Code ISBN	2	3	5	2	8	8	0	4	1
Pondération	10	9	8	7	6	5	4	3	2
Produit	20	27	40	14	48	40	0	12	2

La somme des produits est **203**, dont le reste de la division euclidienne par 11 est **5**. La clé de contrôle est donc 11 - 5 = **6**. L'ISBN au complet est : 2-35288-041-**6**.

La vérification de la clé complète à 10 chiffres donne la somme pondérée 203 + 6 = 209, qui est bien un multiple de 11.


Richard **Hamming**
(1915-1998)

3.6.1. La distance de Hamming

La distance de Hamming doit son nom à Richard **Hamming**. Elle est décrite dans un article fondateur pour la théorie des codes. Elle est utilisée en télécommunication pour compter le nombre de bits altérés dans la transmission d'un message d'une longueur donnée.

Exemple : la distance de Hamming entre 1011101 et 10**0**1**0**01 est 2 car deux bits sont différents.

Il est souhaitable d'avoir une certaine distance entre les mots envoyés, afin de détecter s'il y a eu une erreur de transmission. Par exemple, si l'on a trois messages à transmettre de trois bits, il vaut mieux choisir les codages qui sont à distance 2 les uns des autres, par exemple 000, 110 et 101. En effet, si un seul bit est altéré, on recevra un message impossible. Par contre, en utilisant 000, 001 et 010, un bit altéré pourrait passer inaperçu.

3.6.2. Somme de contrôle

La somme de contrôle (en anglais « checksum ») est un cas particulier de contrôle par redondance. Elle permet de détecter les erreurs, mais pas forcément de les corriger. Nous en avons déjà vu un exemple avec le code ISBN-10.

Le principe est d'ajouter aux données des éléments dépendant de ces dernières et simples à calculer. Cette redondance accompagne les données lors d'une transmission ou bien lors du stockage sur un support quelconque. Plus tard, il est possible de réaliser la même opération sur les données et de comparer le résultat à la somme de contrôle originale, et ainsi conclure sur la corruption potentielle du message.

Bit de parité

Transmettons sept bits auxquels viendra s'ajouter un bit de parité. On peut définir le bit de parité comme étant égal à zéro si la somme des autres bits est paire et à un dans le cas contraire. On parle

de parité paire. Si la somme des bits est impair, c'est qu'il y a eu une erreur de transmission.

Exemple : 1010001 (7 bits) devient **1**1010001 (8 bits)

Cette approche permet de détecter les nombres d'erreurs impaires, mais un nombre pair d'erreurs passera inaperçu.

3.6.3. Code de Hamming

Un code de Hamming permet la détection et la correction automatique d'une erreur si elle ne porte que sur une lettre du message. Un code de Hamming est **parfait**, ce qui signifie que pour une longueur de code donnée, il n'existe pas d'autre code plus compact ayant la même capacité de correction. En ce sens, son rendement est maximal.

Structure d'un code de Hamming

- les m bits du message à transmettre et les n bits de contrôle de parité.
- longueur totale : $2^n - 1$
- longueur du message : $m = (2^n - 1) - n$
- on parle de code $x - y$: x est la longueur totale du code ($n+m$) et y la longueur du message (m)
- les bits de contrôle de parité C_i sont en position 2^i pour $i = 0, 1, 2,...$
- les bits du message D_j occupe le reste du message.

7	6	5	4	3	2	1
D_3	D_2	D_1	C_2	D_0	C_1	C_0

Structure d'un code de Hamming 7-4

Exemples de code de Hamming

- un mot de code 7-4 a un coefficient d'efficacité de 4/7 = 57 %
- un mot de code 15-11 a un coefficient d'efficacité de 11/15 = 73 %
- un mot de code 31-26 a un coefficient d'efficacité de 26/31 = 83 %

Retrouver l'erreur dans un mot de Hamming

Si les bits de contrôle de parité C_2, C_1, C_0 ont tous la bonne valeur, il n'y a pas d'erreurs ; sinon la valeur des bits de contrôle indique la position de l'erreur entre 1 et 7. Le code de Hamming présenté ici ne permet de retrouver et corriger qu'une erreur.

Pour savoir quels bits sont vérifiés par chacun des bits de contrôle de parité, il faut construire le tableau ci-dessous.

On numérote les lignes de 1 à $x=2^n-1$ dans la colonne de droite (prenons comme exemple x=7), puis on convertit chaque nombre en binaire et l'on écrit chaque bit dans les colonnes de gauche. On colorie de la couleur de C_i les nombres de droite s'il y a un 1 dans dans la colonne C_i . Par exemple, 5 sera coloré en vert et en rouge, car sur la ligne du 5, il y a un 1 dans les colonnes C_2 et C_0.

C_2	C_1	C_0	décimal
0	0	1	1 ●
0	1	0	2 ●
0	1	1	3 ● ●
1	0	0	4 ●
1	0	1	5 ● ●
1	1	0	6 ● ●
1	1	1	7 ● ● ●

- C_2 (en vert) colore les bits 4, 5, 6, 7. Ce sont les bits qu'il vérifie.
- C_1 (en bleu) vérifie les bits 2, 3, 6, 7.
- C_0 (en rouge) vérifie les bits 1, 3, 5, 7.
- On constate que chaque bit de données est coloré d'une manière différente. Cela permettra de retrouver la position d'une erreur.

Exemple d'un code de Hamming 7-4

On souhaite envoyer le message 1010. Complétons le mot de Hamming correspondant :

7	6	5	4	3	2	1
1	0	1		0		

C_0 vaut 0 pour pouvoir rendre pair 1+1+0 (les bits d'indices 7, 5, 3).
C_1 vaut 1 pour pouvoir rendre pair 1+0+0 (les bits d'indices 7, 6, 3).
C_2 vaut 0 pour pouvoir rendre pair 1+0+1 (les bits d'indices 7, 6, 5).

7	6	5	4	3	2	1
1	0	1	**0**	0	**1**	**0**

Imaginons que l'on reçoive le mot 0010010 (le bit de poids fort a été altéré).
C_0 a la mauvaise valeur, car 0+1+0+0 est impair, donc il y a une erreur en position 7, 5, 3 ou 1.
C_1 a la mauvaise valeur, car 0+0+0+1 est impair, donc il y a une erreur en position 7, 6, 3 ou 2.
C_2 a la mauvaise valeur, car 0+0+1+0 est impair, donc il y a une erreur en position 7, 6, 5 ou 4.
Écrivons le nombre binaire $C_2C_1C_0$ où C_i vaut 0 si le bit de contrôle C_i a la bonne valeur et 1 sinon. On obtient ici 111, ce qui correspond à 7 en binaire. Le bit erroné est le numéro 7.

Que se passe-t-il si c'est un des bits de contrôle qui est altéré ? Imaginons que l'on ait reçu 1010011 (cette fois-ci, c'est le bit de poids faible qui a été altéré).
C_0 a la mauvaise valeur, car 1+1+0+1 est impair. Il y a une erreur en position 7, 5, 3 ou 1.
C_1 a la bonne valeur, car 1+0+0+1 est pair. Il n'y a pas d'erreur en position 7, 6, 3 et 2
C_2 a la bonne valeur, car 1+0+1+0 est pair. Il n'y a pas d'erreur en position 7, 6, 5 et 4.
Ici, $C_2C_1C_0$ vaut 001. Le bit erroné est donc le numéro 1.

Exercice 3.8

Vous voulez envoyer le mot 1011. Quels bits de contrôle devez-vous lui adjoindre et quelle séquence transmettrez-vous alors ?

Exercice 3.9

Y a-t-il une erreur dans le mot suivant (Hamming 7–4) : 1101101 ?

Exercice 3.10

Soit un mot de Hamming 15–11 suivant :

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
1	0	1	1	0	1	1	1	1	0	1	1	0	1	1

1. Quels sont les bits de contrôle de parité ?
2. Quelles positions contrôle chacun de ces bits ?
3. Quel est le message reçu ?
4. Est-ce que le message reçu correspond au message transmis ?

3.7. Codage de Huffman

Le codage de Huffman (1952) est une méthode de compression statistique de données qui permet de réduire la longueur du codage d'un alphabet. C'est un code de longueur variable optimal, c'est-à-dire que la longueur moyenne d'un texte codé est minimale. On observe des réductions de taille de l'ordre de 20 à 90%.


David A. **Huffman**
(1925-1999)

Le principe

Le principe de l'algorithme de Huffman consiste à recoder les octets rencontrés dans un ensemble de données source avec des valeurs de longueur binaire variable.

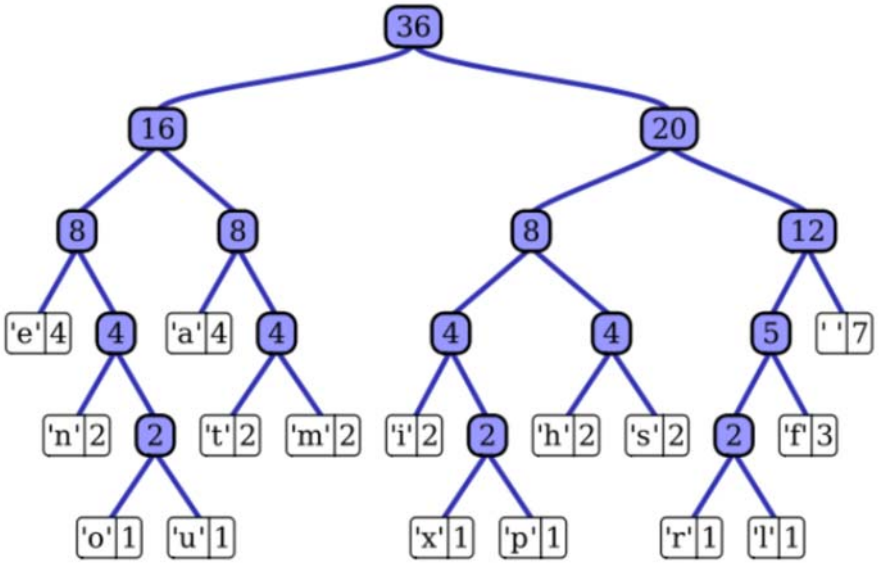
L'unité de traitement est ramenée au bit. Huffman propose de recoder les données qui ont une occurrence très faible sur une longueur binaire supérieure à la moyenne, et recoder les données très fréquentes sur une longueur binaire très courte.

Ainsi, pour les données rares, nous perdons quelques bits regagnés pour les données répétitives.

Par exemple, dans un fichier ASCII le « w » apparaissant 10 fois aura un code très long : 0101000001000. Ici la perte est de 40 bits (10 x 4 bits), car sans compression, il serait codé sur 8 bits au lieu de 12. Par contre, le caractère le plus fréquent comme le « e » avec 200 apparitions sera codé par 1. Le gain sera de 1400 bits (7 x 200 bits). On comprend l'intérêt d'une telle méthode.

De plus, le codage de Huffman a une propriété de préfixe : une séquence binaire ne peut jamais être à la fois représentative d'un élément codé et constituer le début du code d'un autre élément.

Si un caractère est représenté par la combinaison binaire 100 alors la combinaison 10001 ne peut être le code d'aucune autre information. Dans ce cas, l'algorithme de décodage interpréterait les 5 bits comme deux mots : 100-01. Cette caractéristique du codage de Huffman permet une codification à l'aide d'une structure d'arbre binaire :

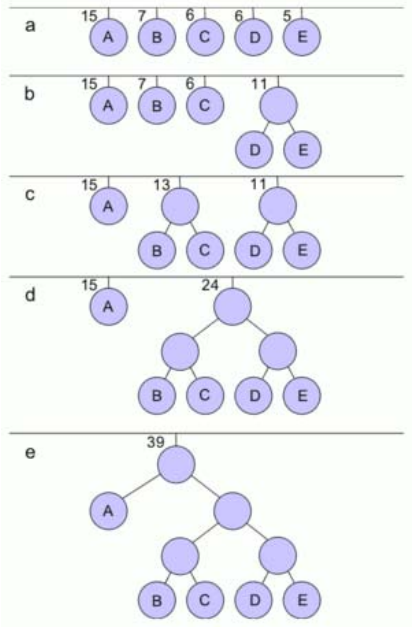


Un exemple d'arbre obtenu avec la phrase « this is an example of a huffman tree »

Méthode

Au départ, chaque lettre a une étiquette correspondant à sa fréquence (ou sa probabilité) d'apparition. Il y a autant d'arbres (à 1 sommet) que de lettres.

L'algorithme choisit à chaque étape les deux arbres d'étiquettes minimales, soit x et y, et les remplace par l'arbre formé de x et y et ayant comme étiquette la somme de l'étiquette de x et de l'étiquette de y. La figure ci-dessous représente les étapes de la construction d'un code de Huffman pour l'alphabet source {A, B, C, D, E}, avec les fréquences F(A)=15, F(B)=7, F(C)=6, F(D)=6 et F(E)=5.



Le code d'une lettre est alors déterminé en suivant le chemin depuis la racine de l'arbre jusqu'à la feuille associée à cette lettre en concaténant successivement un 0 ou un 1 selon que la branche suivie est à gauche ou à droite. Ainsi, sur la figure ci-dessus, A=0, B=100, C=101, D=110, E=111.

Par exemple, le mot « ABBE » serait codé 0100100111. Pour décoder, on lit simplement la chaîne de bits de gauche à droite. Le seul découpage possible, grâce à la propriété du préfixe, est 0-100-100-111.

Ce principe de compression est aussi utilisé dans le codage d'image TIFF (Tagged Image Format File) spécifié par *Microsoft Corporation* et *Aldus Corporation*.

Il existe des méthodes qui ne conservent pas exactement le contenu d'une image (méthodes non conservatives) mais dont la représentation visuelle reste correcte. Entre autres, il y a la méthode JPEG (Join Photographic Experts Group) qui utilise la compression de type Huffman pour coder les informations d'une image.

Malgré son ancienneté, cette méthode est toujours remise au goût du jour, et offre des performances appréciables.

Exercice 3.12

Construisez un codage de Huffman du message « ceciestuncodagedehuffman » (on a supprimé les espaces et la ponctuation pour simplifier la construction). Il y a plusieurs codages de Huffman possibles.

Vérifiez la propriété du préfixe.

3.8. QR Codes

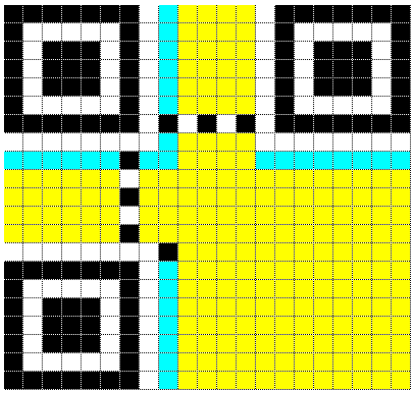
Le code QR (ou ***QR code*** en anglais) est un code-barres en deux dimensions (ou code à matrice) constitué de modules noirs disposés dans un carré à fond blanc. Ce cours contient un QR code à chaque début de chapitre. Le nom *QR* est l'acronyme de l'anglais *Quick Response*, car son contenu de données peut être décodé rapidement.

Le code QR a été créé par l'entreprise japonaise Denso-Wave en 1994 pour le suivi des pièces de voiture dans les usines de Toyota.

Les codes QR peuvent mémoriser des adresses web, du texte, des numéros de téléphone, des SMS ou autres types de données lisibles par les

smartphones et les téléphones mobiles équipés d'une application de lecture (lecteur de code QR ou *QR reader* en anglais).

L'avantage du code QR est sa facilité et rapidité d'utilisation et de création. Pour lire un code QR, il suffit de lancer l'application de lecture et viser le code dans le mobile. De nombreuses pages Web offrent ces applications pour mobiles, généralement sans frais.



En ce qui concerne l'écriture, il y a plusieurs sites web qui permettent de générer librement les codes QR, par exemple http://qrcode.kaywa.com/.

Ils peuvent stocker jusqu'à 7089 caractères numériques, 4296 caractères alphanumériques ou 2953 octets. Par rapport au code-barres traditionnel qui ne peut stocker que de 10 à 13 caractères, ils ont l'avantage de pouvoir stocker beaucoup d'informations tout en étant petits et rapides à scanner.

Le code QR est défini comme un standard ISO (IEC18004).

Les cases noires et blanches servent à l'orientation du QR code. Grâce à ces trois carrés, l'image peut être lue dans tous les sens.

L'information est codée selon le système Reed-Solomon et stockée dans la partie jaune. Certains types de codes restent lisibles avec 30% de dégradation.

Dans la partie cyan se trouvent des informations sur le format.

Ce QR-Code dessiné dans le sable est tout à fait valide. Essayez de le scanner !

Idem pour ce QR-code artistique !

Sources

[1] Dumas, Roch, Tannier, Varrette, *Théorie des codes : Compression, cryptage, correction*, Dunod, 2006

[2] Martin B., *Codage, cryptologie et applications*, Presses Polytechniques et Universitaires Romandes (PPUR), 2004

[3] Comment ça marche, « Représentation des nombres entiers et réels », <http://www.commentcamarche.net/contents/base/representation.php3>

[4] Wikipédia, « International Book Standard Number », <http://fr.wikipedia.org/wiki/ISBN>

[5] Duvallet Claude, « Les codes correcteurs et les codes détecteurs d'erreurs », <http://litis.univ-lehavre.fr/~duvallet/enseignements/Cours/LPRODA2I/UF9/LPRODA2I-TD2-UF9.pdf>

[6] Wikipédia, « Codage de Huffman », <http://fr.wikipedia.org/wiki/Codage_de_Huffman>