Chapitre 1

Codage

1.1 Introduction

Le terme numérique est dérivé de la façon avec laquelle les ordinateurs effectuent les opérations en comptant les numéros. Pour plusieurs années, les applications de l'électronique numérique sont limitées aux systèmes informatiques. Aujourd'hui, la technologie numérique est appliqué dans plusieurs domaines en plus que le domaine des ordinateurs (télévision, militaire, médical, industrie). Au cours des années, la technologie numérique s'est progressé des circuits en tubes aux circuits intégrés contenant des millions de transistors.

1.1.1 Les quantités numériques et analogiques

Une quantité analogique est une quantité ayant des valeurs continues, par contre, une quantité numérique est une quantité ayant des valeurs discrètes. La plupart des choses qu'on peut mesurer quantitativement se produit en une forme analogique dans la nature (température, temps, pression, distance, voix). Et si on suppose qu'on mesure chaque heure la température ambiante, on va avoir des valeurs discrètes dans une une période de 24 heures. Nous avons effectivement converti une quantité analogique en une forme qui peut être numérisée en représentant chaque échantillon de valeur par un code numérique.

La représentation numérique a certains avantages par apport à la représentation analogique dans les applications électroniques. Premièrement, les données numériques (caractères, nombres, images, vidéos, audios, ...) peuvent

être traitées et transmises d'une façon plus efficace et rassurante que les données analogiques. Deuxièmement, les données numériques ont un grand avantage qu'on on a besoin de les enregistrer.

L'électronique numérique comporte les circuits et les systèmes dans lesquels il y a seulement deux états possibles . Ces états sont représentés par deux différents niveaux de voltage : Un qui est haut et l'autre qui est bas. Ces deux états peuvent aussi être représentés par le niveau du courant (air, eau), les bits et les gravures dans un CD ou DVD. Dans les systèmes numériques comme les ordinateurs, la combinaison de ces deux états est appelée codes, qui sont utilisés pour représenter les nombres, symboles, caractères, et des autres types d'informations. Le système numérique de ces deux états est appelé binaire, et ces deux nombres sont 0 et 1. Un nombre binaire est appelé alors, bit.

1.1.2 Les nombres binaires

Chacun des deux nombres dans le système binaire, 1 et 0, est appelé bit, qui est un raccourci des mots binary digit. Dans les circuits numériques, il existe deux niveaux de voltage qui sont utilisés pour représenter les deux bits. Généralement, 1 est représenté par le haut voltage et 0 est représenté par le bas voltage.

1.1.3 Vocabulaire

•Le bit (b minuscule dans les notations) est la plus petite unité d'information manipulable par une machine numérique, en numérotation binaire il représente 1 ou 0.

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

•Un mot (word) est une unité d'information composée généralement de 16 bits (2 caractères).

Les types d'informations traitées directement par un processeur ne sont pas très nombreux.

Les données :

— les entiers (naturels et relatifs).
— les flottants (simple et double).
— les caractères.

Le codage de ces trois types est actuellement définie formellement par des standards(normes spécifiés par des organisations internationales.).
Les instructions : dont le codage est spécifique au processeur.

L'information traitée par un ordinateur est toujours représentée sous la forme d'un ensemble de nombres écrits en base 2 (suite de 0 et de 1). L'unité d'information est le bit.

1.1.4 Unités de mesure en informatique

Préfixes binaires (préfixes CEI)

kibi (Ki)	=	2^{10}	$= 1024$
mébi (Mi)	=	2^{20}	
gibi (Gi)	=	2^{30}	
tébi (Ti)	=	2^{40}	
pébi (Pi)	=	2^{50}	
exbi (Ei)	=	2^{60}	
zébi (Zi)	=	2^{70}	
yobi (Yi)	=	2^{80}	

Préfixes décimaux (préfixes SI)

kilo (K)	=	10^3	$= 1024$
méga (M)	=	10^6	
giga (G)	=	10^9	
téra (T)	=	10^{12}	
péta (P)	=	10^{15}	
exa (E)	=	10^{18}	
zetta (Z)	=	10^{21}	
yotta (Y)	=	10^{24}	

Exemple 1

— 30 Ko $= 30 \times 1000\text{o} = 3 \times 10^4 \times 2^{-10}$ Kio $= 29.29$ Kio

— 52 Mio $= 52 \times 2^20\text{o} = 52 \times 2^{20} \times 10^{-3}$ Ko $= 53.248 \times 2^{10}$ Ko

— 10 Go $= 10 \times 10^9\text{o} = 8 \times 10^{10}$ b

— 1000000 b $= 10^6 \times 10^{-9} = 10^{-3}$ Gb

1.2 Les bases décimale, binaire et hexadécimale

Le codage de l'information consiste à établir une correspondance entre la représentation externe de l'information et sa représentation interne dans la machine. On utilise la représentation binaire car elle est simple et facile à réaliser.

Le système des nombres binaire et le codage numérique sont fondamentales pour les ordinateurs et l'électronique numérique en général. La numérotation permet de représenter un mot (ou nombre) par la juxtaposition ordonnée de variables (ou symboles)) pris parmi un ensemble. Dans un système de numérotation en base B, un nombre noté $N_{(B)}$ égal à :

$$N_{(B)} = \sum_{k=0}^{n-1} a_k \times B^k = a_{n-1}a_{n-2}...a_1a_0 \tag{1.1}$$

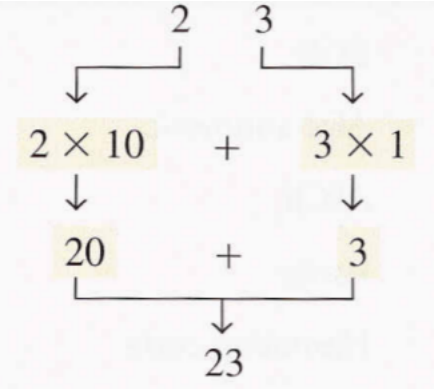
— B : base ou nombre de chiffres différents qu'utilise le système de numérotation.
— a_k : chiffre de rang k.
— B^k : Poids associée à a_k.

1.2.1 Les nombres décimaux

Vous êtes familier à utiliser le système des nombres décimaux parce que vous utilisez les nombres décimaux chaque jour. Même si les nombres décimaux sont communs, leur structure pondérée n'est pas comprise. On va revoir la structure des nombres décimaux pour comprendre mieux la structure du système des nombres binaires qui sont importants pour les ordinateurs et l'électronique numérique.

Dans le système des nombres décimaux chacun des dix chiffres, 0 à 9, représentent une certaine quantité. Comme vous savez, les dix chiffres ne vous limitent pas à exprimer juste dix nombres mais vous donnent la possibilité d'écrire autant de nombre que vous voulez dont les chiffres 0 à 9 prend différentes position et ainsi différentes magnitude de quantité.

Exemple 2



La position de chaque chiffre dans un nombre décimal indique la magnitude de quantité représenté et elle peut être assigné à un poids. Les poids de tous ces chiffres sont des puissances positives de dix qui augmente de gauche à droite en commençant par $10^0 = 1$.
$...10^5 10^4 10^3 10^2 10^1 10^0$

Pour les nombres fractionnaires, les pondérations sont des puissances négatives de dix et diminue de gauche à droite en commençant par 10^{-1}
$10^2 10^1 10^0 . 10^{-1} 10^{-2} 10^{-3} ...$
La valeur d'un nombre décimal est la somme des nombres après multiplication de chaque chiffre par son poids.

Exemple 3

Exprimer le nombre 58 sous la forme d'une somme des valeurs de chaque chiffre.

$$\begin{aligned} 58 &= (5 \times 10^1) + (8 \times 10^0) \\ &= (5 \times 10) + (8 \times 1) \\ &= 50 + 8 \end{aligned}$$

Exemple 4

Exprimer le nombre 586.23 sous la forme d'une somme des valeurs de chaque chiffre.

$$\begin{aligned} 586.23 &= (5 \times 10^2) + (8 \times 10^1) + (6 \times 10^0) + (2 \times 10^{-1}) + (3 \times 10^{-2} \\ &= (5 \times 100) + (8 \times 10) + (6 \times 1) + (2 \times 0.1) + (3 \times 0.01) \\ &= 500 + 80 + 6 + 0.2 + 0.03 \end{aligned}$$

Exercice 1

Quelle est le poids du chiffre 7 dans les nombres suivants ?

— (a) 1370
— (b) 6725
— (c) 7051
— (d) 58.72

Solution de l'exercice 1

— (a) 10^1
— (b) 10^2
— (c) 10^3
— (d) 10^{-1}

1.2.2 Les nombres binaires

Le système des nombres binaires est une autre manière pour représenter les quantités. Il est moins compliqué que le système binaire parce que il contient juste deux chiffres. Le système décimal avec ses dix chiffres est un système de base-dix ; tandis que le système binaire avec ses deux chiffres est un système de base-deux. Les deux chiffres binaires (bits) sont 1 et 0. La position d'un 1 ou d'un 0 dans un nombre binaire indique son poids.

Pour savoir comment on peut compter dans le système binaire, on va premièrement compter dans le système décimal.

— On commence par 0 on augmente jusqu'à 9.
— On commence alors une autre position de chiffre (vers la gauche) et on continue de 10 jusqu'à 99.
— On va ajouter une troisième position pour compter de 100 à 999.

Une situation comparable s'impose quand on compte en binaire, sauf que nous avons juste deux chiffres, appelés bits.

— 0 1
— 10 11
— 100 101 110 111

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

Comme on remarque dans le tableau ci-dessus, pour compter de 0 à 15 on besoin de 4 bits. En général avec n bits on peut compter jusqu'à un nombre égale à $2^n - 1$, qui est le plus large décimal.

Exemple 5

— Avec 5 chiffres on peut compter de 0 jusqu'à $2^5 - 1 = 31$.
— Avec 6 chiffres on peut compter de 0 jusqu'à $2^6 - 1 = 63$.

La structure pondéré des nombres binaires

Un nombre binaire est un nombre pondéré. Le plus à droit bit est le LSB (least significant bit), il a un poids de $2^0 = 1$. Les poids augmentent de droite vers la gauche par une puissance de 2 pour chaque bit. Le bit le plus à gauche est le MSB (most significant bit), son poids dépend de la taille du nombre binaire.

La structure des poids pour un nombre binaire est comme suit :

$$2^{n-1} \dots 2^3 2^2 2^1 2^0$$

2^8	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
256	128	64	32	16	8	4	2	1

Conversion du binaire au décimal

La valeur décimale d'un nombre binaire peut être trouvée en additionnant les poids de tous les bits qui sont 1 et on négligeant les poids des bits qui sont 0.

Exemple 6

— Convertir le nombre binaire 110110 au décimal.
— Convertir le nombre binaire 10010001 au décimal.
$110110_{(2)} = 2^5 + 2^4 + 2^2 + 2^1 = 54_{10}$
$10010001_{(2)} = 2^7 + 2^4 + 2^0 = 145_{10}$

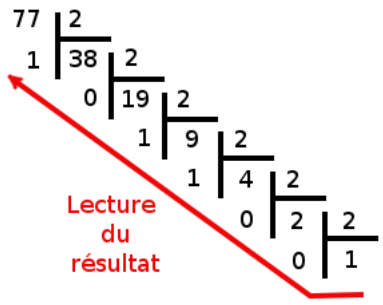
La conversion du décimal au binaire

La méthode des somme des poids Une façon de trouver le nombre binaire équivalent à un nombre décimal est de déterminer la suite des poids binaires dont la somme est égale à ce nombre décimal.

Exemple 7
$9 = 8 + 1$ ou $9 = 2^3 + 2^0$ qui est $1 \times 2^3 + 0 \times 2^2 + 0 \times 2^1 + 1 \times 2^0$ on obtient alors 1001

— (a) 12
— (b) 25
— (c) 58
— (d) 82
$12_{10} = 8 + 4 = 2^3 + 2^2 = 1100_2$
$25_{10} = 16 + 8 + 1 = 2^4 + 2^3 + 2^0 = 11001_2$
$58_{10} = 32 + 16 + 8 + 2 = 2^5 + 2^4 + 2^3 + 2^1 = 111010_2$
$82_{10} = 64 + 16 + 2 = 2^6 + 2^4 = 1010000_2$

La méthode de la division euclidienne par 2



$77_{(10)} = 1001101_{(2)}$

Exemple 8

Convertir chaque nombre décimal en binaire en utilisant les deux méthodes

— 23
— 57
— 45

1.2.3 Les nombres hexadécimaux

Le système des nombres des hexadécimaux a une base de seize, il est composé de seize caractères (numéros et alphabet). La plupart des systèmes numérique traitent les données binaires sous forme de groupes de multiple de quatre bits ce qui rend le nombre hexadécimal très approprié grâce à sa forme dont chaque chiffre est un nombre binaire de 4 bits.

Le système hexadécimale (base 16) est basé sur seize symboles, de 0 à 9 et de A à F.

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

Compter en hexadécimal

— de 0_{16} à F_{16}
— de 10_{16} à FF_{16}
— de 100_{16} à FFF_{16}

La conversion du binaire au hexadécimal

Il est très simple de convertir un nombre binaire en hexadécimal, il faut simplement faire des groupes de 4 bits en commençant de droite à gauche et puis on remplace chaque groupe par son équivalent en symbole décimal.

Exemple 9

— 1100101001010111
— 111111000101101001
1100 1010 0101 0111 = $CA57_{16}$
11 1111 0001 0110 1001 = $3F169_{16}$

La conversion de l'hexadécimal au binaire

Pour convertir un nombre de l'hexadécimal au binaire on va faire l'opération inverse, c-à-d on va remplacer chaque symbole hexadécimal par 4 bits qui lui conviennent.

Exemple 10
— $10A4_{16}$
— $CF8E_{16}$
— 9742_{16}
$10A4_{16} = 0001\ 0000\ 1010\ 0100 = 1000010100100_2$
$CF8E_{16} = 1100\ 1111\ 1000\ 1110 = 1100111110001110_2$
$9742_{16} = 1001\ 0111\ 0100\ 0010 = 1001011101000010_2$

La conversion de l'hexadécimal au décimal

Pour trouver l'équivalent décimal d'un nombre hexadécimal il faut premièrement convertir le nombre hexadécimal au binaire et après convertir le binaire en décimal.

Exemple 11
— $1C_{16}$
— $A85_{16}$
$1C_{16} = 11100_2 = 28_{10}$
$A85_{16} = 101010000101_2 = 2693_{10}$
Une autre façon pour effectuer cette conversion est de multiplier la valeur décimal de chaque chiffre hexadécimal par son poids et faire après la somme de tous.

Exemple 12
— $E5_{16}$
— $B2F8_{16}$
$E5_{16} = 14 \times 16^1 + 5 \times 16^0 = 229_{10}$
$B2F8_{16} = 11 \times 16^3 + 2 \times 16^2 + 15 \times 16^1 + 8 \times 16^0 = 45816_{10}$

La conversion du décimal au hexadécimal

On peut passer par la conversion du décimal au binaire puis convertir le binaire en hexadécimal, si non on peut faire directement des divisions par

16 comme dans le cas des divisions d'un nombre décimal par 2 pour trouver l'équivalent d'un décimal en binaire.

Exemple 13

— 100
— 564
$100_{10} = 1100100_2 = 0110\ 0100 = 64_{16}$
$564_{10} = 1000110100_2 = 0001\ 0001\ 0100 = 234_{16}$

1.3 Représentation des nombres entiers

1.3.1 Représentation d'un entier naturel

Un entier naturel est un nombre entier positif ou nul. Pour coder des nombres entiers naturels compris entre 0 et 255, il nous suffira 8 bits ($2^8 = 256$. D'une manière générale un codage sur n bits permet de représenter des nombres entiers naturels compris entre 0 et $2^n - 1$.

Exemple
$9 = 00001001_{(2)}$, $128 = 10000000_{(2)}$

1.3.2 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.

Un entier relatif positif ou nul sera représenté en binaire comme un entier naturel, à la seule différence que le bit de poids fort représente le signe.

Exemple
Si on code un entier naturel sur 4 bits, le nombre le plus grand sera $0111_{(2)} = 7_{(10)}$

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

1. Écrire la valeur absolue du nombre en base 2. Le bit du poids fort doit être égale à 0.
2. Inverser les bits.
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.

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.

Exemple
Calculer la valeur de -19 sur $8bits$:

1. $|-19|_{(10)} = 00010011$
2. 11101100
3. 11101101

$11101101 + 00010011 = 00000000$ (avec un retenu de 1 qui est éliminée).

Exemple
— Calculer la valeur de -46 sur $8bits$
— Calculer la valeur de -78 sur $8bits$

1.4 Représentation d'un nombre réel

En base 10, l'expression 652.375 est une manière abrégé d'écrire : $6.10^2 + 5.10^1 + 2.10^0 + 3.10^{-1} + 7.10^{-2} + 5.10^{-3}$
Il en va de même pour la base 2. Ainsi, l'expression 111.101 signifie : $1.2^2 + 1.2^1 + 0.2^0 + 1.2^{-1} + 0.2^{-2} + 1.2^{-3}$.

1.4.1 Conversion de binaire en décimal

La conversion se fait rapidement d'un nombre réel de la base 2 vers la base 10.

$$\begin{aligned} 110.101_{(2)} &= 1.2^2 + 1.2^1 + 0.2^0 + 1.2^{-1} + 1.2^{-3} \\ &= 4 + 2 + 0.5 + 0.125 \\ &= 6.625_{(10)} \end{aligned}$$

1.4.2 Conversion de décimal en binaire

— La partie entière se transforme comme au par avant.
— La partie décimale se fait selon le schéma suivant :
$0.437 \times 2 = \mathbf{0}.874$
$0.874 \times 2 = \mathbf{1}.748$
$0.748 \times 2 = \mathbf{1}.496$
$0.496 \times 2 = \mathbf{0}.992$
$0.992 \times 2 = \mathbf{1}.984$
$0.984 \times 2 = \mathbf{1}.968$
$\mathbf{0}.421875_{(10)}$

1.4.3 La norme IEEE 754

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.

Certains règles sont à respecter :

— l'exposant 00000000 est interdit.
— l'exposant 111111 est interdit.
— il faut rajouter 127 (01111111) à l'exposant pour une conversion de décimal vers un nombre réel binaire.

Un nombre réel x s'écrit sous la forme :

$$x = (-1)^S \times 2^{E-127} \times (1 + F) \tag{1.2}$$

— S est le bit de signe.
— 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.

— $6.625_{10} = 110.1010_{(2)}$
— Nous mettons ce nombre sous la forme 1.partie fractionnaire 10101000000000000000000.
— Exposant $= 127 + 2 = 129_{(10)} = 10000001_{(2)}$.

Signe	Exposant	Mantisse
1	10000001	10101000000000000000000

1.5 Opérations arithmétiques

Les opérations arithmétiques s'effectuent en base quelconque B de la même manière qu'en base 10. Une retenue apparaît lorsqu'on dépasse la valeur B de la base.

1.5.1 Addition

Cas de deux nombres positifs

Effectuons l'addition $(+14)_{(10)}$ et $(+15)_{(10)}$
$(+14)_{(10)} = \mathbf{0}0001110_{(2)}$
$(+15)_{(10)} = \mathbf{0}0001111_{(2)}$

$$\begin{array}{r} 00001110 \\ +\ 00001111 \\ \hline 00011101 \end{array}$$

Cas de deux nombres de signes différents

Effectuons l'addition des nombres suivants :
$(+19)_{(10)} = \mathbf{0}0010011_{(2)}$
$(-14)_{(10)} = \mathbf{1}1110010_{(2)}$

$$\begin{array}{r} 00010011 \\ +\ 11110010 \\ \hline 00000101 \end{array}$$

Effectuons l'addition des nombres suivants :
$(-19)_{(10)} = \mathbf{1}1101101_{(2)}$

$(+14)_{(10)} = \mathbf{0}0001110_{(2)}$

$$\begin{array}{r} \mathbf{11101101} \\ +\ \mathbf{00001110} \\ \hline \mathbf{11111011} \end{array}$$

Effectuons l'addition de $(-19)_{(10)}$ et $(-14)_{(10)}$
$(-19)_{(10)} = \mathbf{1}1101101_{(2)}$
$(-14)_{(10)} = \mathbf{1}1110010_{(2)}$

$$\begin{array}{r} \mathbf{11101101} \\ +\ \mathbf{11110010} \\ \hline \mathbf{11011111} \end{array}$$

1.5.2 Soustraction

La soustraction en binaire se fait de la même façon qu'une addition. Si l'on soustrait un bit à un autre à zéro, on soustrait une retenue pour le bit de poids plus élevé.
$0 - 1 = 1$

Exemple
$20_{(10)} - 10_{(10)}$
$20_{(10)} = 00010100_{(2)}$
$10_{(10)} = 00001010_{(2)}$

$$\begin{array}{r} \mathbf{00010100} \\ -\ \mathbf{00001010} \\ \hline \mathbf{00001010} \end{array}$$

1.5.3 Multiplication

La multiplication en binaire est la même chose que la multiplication en décimale.

Exemple

$7_{(10)} \times 3_{(10)}$
$7_{(10)} = 00000111_{(2)}$
$3_{(10)} = 00000011_{(2)}$

```
  00000111
x 00000011
-----------
  00000111
 00000111.
-----------
 00010101
```

1.5.4 Division

La division en binaire est effectuée de la même manière qu'on décimal.

Exemple

Effectuons la division de $21_{(10)}$ par $7_{(10)}$

```
00010101 | 00000111
  - 0111 | 000011
---------|
   00111 |
 - 00111 |
---------|
   00000
```

1.6 Le code ASCII

Dec	Hex	Name	Char	Ctrl-char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char
0	0	Null	NUL	CTRL-@	32	20	Space	64	40	@	96	60	`
1	1	Start of heading	SOH	CTRL-A	33	21	!	65	41	A	97	61	a
2	2	Start of text	STX	CTRL-B	34	22	"	66	42	B	98	62	b
3	3	End of text	ETX	CTRL-C	35	23	#	67	43	C	99	63	c
4	4	End of xmit	EOT	CTRL-D	36	24	$	68	44	D	100	64	d
5	5	Enquiry	ENQ	CTRL-E	37	25	%	69	45	E	101	65	e
6	6	Acknowledge	ACK	CTRL-F	38	26	&	70	46	F	102	66	f
7	7	Bell	BEL	CTRL-G	39	27	'	71	47	G	103	67	g
8	8	Backspace	BS	CTRL-H	40	28	(	72	48	H	104	68	h
9	9	Horizontal tab	HT	CTRL-I	41	29	)	73	49	I	105	69	i
10	0A	Line feed	LF	CTRL-J	42	2A	*	74	4A	J	106	6A	j
11	0B	Vertical tab	VT	CTRL-K	43	2B	+	75	4B	K	107	6B	k
12	0C	Form feed	FF	CTRL-L	44	2C	,	76	4C	L	108	6C	l
13	0D	Carriage feed	CR	CTRL-M	45	2D	-	77	4D	M	109	6D	m
14	0E	Shift out	SO	CTRL-N	46	2E	.	78	4E	N	110	6E	n
15	0F	Shift in	SI	CTRL-O	47	2F	/	79	4F	O	111	6F	o
16	10	Data line escape	DLE	CTRL-P	48	30	0	80	50	P	112	70	p
17	11	Device control 1	DC1	CTRL-Q	49	31	1	81	51	Q	113	71	q
18	12	Device control 2	DC2	CTRL-R	50	32	2	82	52	R	114	72	r
19	13	Device control 3	DC3	CTRL-S	51	33	3	83	53	S	115	73	s
20	14	Device control 4	DC4	CTRL-T	52	34	4	84	54	T	116	74	t
21	15	Neg acknowledge	NAK	CTRL-U	53	35	5	85	55	U	117	75	u
22	16	Synchronous idle	SYN	CTRL-V	54	36	6	86	56	V	118	76	v
23	17	End of xmit block	ETB	CTRL-W	55	37	7	87	57	W	119	77	w
24	18	Cancel	CAN	CTRL-X	56	38	8	88	58	X	120	78	x
25	19	End of medium	EM	CTRL-Y	57	39	9	89	59	Y	121	79	y
26	1A	Substitute	SUB	CTRL-Z	58	3A	:	90	5A	Z	122	7A	z
27	1B	Escape	ESC	CTRL-[	59	3B	;	91	5B	[	123	7B	{
28	1C	File separator	FS	CTRL-\	60	3C	<	92	5C	\	124	7C	|
29	1D	Group separator	GS	CTRL-]	61	3D	=	93	5D	]	125	7D	}
30	1E	Record separator	RS	CTRL-^	62	3E	>	94	5E	^	126	7E	~
31	1F	Unit separator	US	CTRL-_	63	3F	?	95	5F	_	127	7F	DEL

ASCII est une norme qui établit une correspondance entre une représentation binaire des caractères de l'alphabet latin et les symboles, les signes, qui constituent cet alphabet.

Exemple

Le caractère a est associé à 1100001 (97) et A à 1000001 (65).

Grâce à la norme ASCII, toutes les sortes de machines peuvent stocker, analyser et communiquer de l'information textuelle. La quasi-totalité des ordinateurs (personnels ou stations de travail) utilisent l'encodage ASCII dont la forme de base représentait les caractères sur 7 bits (de 0 à 127, en total 128 caractères).

— Les codes de 0 à 31 sont des caractères de contrôle puisqu'ils permettent de faire des actions comme le retour à la ligne.
— Les codes de 48 à 57 représentent les chiffres 0, 1 ,2 , ... , 9.
— Les codes de 65 à 90 représentent les majuscules.

— Les codes de 97 à 122 représentent les minuscules.

Le code ASCII sur 7 bits est dédié à la langue anglaise, c'est à dire ne contient pas de caractères accentués, ni de caractères spécifiques à une langue. C'est après que le code ASCII a été étendu à 8 bits pour pouvoir coder plus de caractères (le code ASCII étendu).

ASCII control characters		
00	NULL	(Null character)
01	SOH	(Start of Header)
02	STX	(Start of Text)
03	ETX	(End of Text)
04	EOT	(End of Trans.)
05	ENQ	(Enquiry)
06	ACK	(Acknowledgement)
07	BEL	(Bell)
08	BS	(Backspace)
09	HT	(Horizontal Tab)
10	LF	(Line feed)
11	VT	(Vertical Tab)
12	FF	(Form feed)
13	CR	(Carriage return)
14	SO	(Shift Out)
15	SI	(Shift In)
16	DLE	(Data link escape)
17	DC1	(Device control 1)
18	DC2	(Device control 2)
19	DC3	(Device control 3)
20	DC4	(Device control 4)
21	NAK	(Negative acknowl.)
22	SYN	(Synchronous idle)
23	ETB	(End of trans. block)
24	CAN	(Cancel)
25	EM	(End of medium)
26	SUB	(Substitute)
27	ESC	(Escape)
28	FS	(File separator)
29	GS	(Group separator)
30	RS	(Record separator)
31	US	(Unit separator)
127	DEL	(Delete)

ASCII printable characters					
32	space	64	@	96	`
33	!	65	A	97	a
34	"	66	B	98	b
35	#	67	C	99	c
36	$	68	D	100	d
37	%	69	E	101	e
38	&	70	F	102	f
39	'	71	G	103	g
40	(	72	H	104	h
41	)	73	I	105	i
42	*	74	J	106	j
43	+	75	K	107	k
44	,	76	L	108	l
45	-	77	M	109	m
46	.	78	N	110	n
47	/	79	O	111	o
48	0	80	P	112	p
49	1	81	Q	113	q
50	2	82	R	114	r
51	3	83	S	115	s
52	4	84	T	116	t
53	5	85	U	117	u
54	6	86	V	118	v
55	7	87	W	119	w
56	8	88	X	120	x
57	9	89	Y	121	y
58	:	90	Z	122	z
59	;	91	[	123	{
60	<	92	\	124	|
61	=	93	]	125	}
62	>	94	^	126	~
63	?	95	_		

Extended ASCII characters							
128	Ç	160	á	192	└	224	Ó
129	ü	161	í	193	┴	225	ß
130	é	162	ó	194	┬	226	Ô
131	â	163	ú	195	├	227	Ò
132	ä	164	ñ	196	─	228	õ
133	à	165	Ñ	197	┼	229	Õ
134	å	166	ª	198	ã	230	µ
135	ç	167	º	199	Ã	231	þ
136	ê	168	¿	200	╚	232	Þ
137	ë	169	®	201	╔	233	Ú
138	è	170	¬	202	╩	234	Û
139	ï	171	½	203	╦	235	Ù
140	î	172	¼	204	╠	236	ý
141	ì	173	¡	205	═	237	Ý
142	Ä	174	«	206	╬	238	¯
143	Å	175	»	207	¤	239	´
144	É	176	░	208	ð	240	≡
145	æ	177	▒	209	Ð	241	±
146	Æ	178	▓	210	Ê	242	‗
147	ô	179	│	211	Ë	243	¾
148	ö	180	┤	212	È	244	¶
149	ò	181	Á	213	ı	245	§
150	û	182	Â	214	Í	246	÷
151	ù	183	À	215	Î	247	¸
152	ÿ	184	©	216	Ï	248	°
153	Ö	185	╣	217	┘	249	¨
154	Ü	186	║	218	┌	250	·
155	ø	187	╗	219	█	251	¹
156	£	188	╝	220	▄	252	³
157	Ø	189	¢	221	¦	253	²
158	×	190	¥	222	Ì	254	■
159	ƒ	191	┐	223	▀	255	nbsp

Il existe d'autres normes que l'ASCII, par exemple l'Unicode, il représente une version unifiée des différents encodages de caractères complétant L'ASCII et permet l'encodage de caractères autres que ceux de l'alphabet latin. Unicode définit des dizaines de milliers de codes.

1.7 Codes détecteurs/correcteurs d'erreurs

De nos jours, les communications jouent un rôle essentiel dans notre vie quotidienne. On veut avoir des communications de haut débit mais aussi de

qualité. C'est là ou vient le rôle des codes correcteurs d'erreurs qui corrigent les erreurs de transmission d'un message sur une voie de communication peu fiable.

On a vu dans ce qui précède que l'information est codé par des 0 et des 1 et elle est transmise sous forme de courants par exemple, ce qui provoque les erreurs de transmissions qui peuvent être des remplacements de 0 par 1 et inversement.

On peut détecter ou corriger des erreurs en utilisant la redondance de l'information pour être capable de détecter si des erreurs ont eu lieu et être capable de les corriger au mieux.

Exemple

Message transmis	Message reçu
0624589685	0624589686
zero six vingt-quatre cinquante-huit quatre-vingt seize quatre-vingt cinq	zer0 slx vingt-quatte cinquante-huil quatre-vintt seife quttre-vingt cinq

D'après le tableau ci-dessus, on remarque que l'information réduite est difficile à corriger alors qu'il est possible de détecter et corriger l'information redondante.

Principe général

— Chaque suite de bits à transmettre est augmentée par une autre suite de bits de "redondance" ou de contrôle.
— Pour chaque suite de k bits transmise, on ajoute r bits. On note $n = k + r$ et on dit qu'on utilise un code $C(n, k)$
— Les bits ajoutés permettent d'effectuer le contrôle à la réception.

1.7.1 Distance de Hamming

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

Exemple
La distance de Hamming entre 1011110011 et 1011110**10**1 est de 2 car deux bits sont différents

1.7.2 Somme de contrôle

L'un des contrôles par redondance est la somme de contrôle, elle permet de détecter les erreurs, mais pas forcément de les corriger. Le principe est d'ajouter aux données qu'on veut envoyer des éléments dépendants de ces dernières et simples à calculer. Ces éléments accompagne les données lors d'une transmission ou bien lors du stockage sur un support donné. Ainsi, en répétant les mêmes opérations, il sera possible de comparer le résultat à la somme de contrôle originale et conclure sur la corruption potentielle du message.

Bit de parité

Transmettons sept bits auxquels on ajoute un bit de parité définit comme suit :

— Si la somme des autres bits est paire, le bit de parité est égal à zéro.

— Si la somme des autres bits est impair, le bit de parité est égal à un.

Dans ce cas on parle de parité paire.

Exemple
On transmet 1010001 (7 bits), et on ajoutons le bit de parité la donnée devient **1**1010001 (8 bits). L'inconvénient de cette approche est qu'elle permet de détecter juste les nombres d'erreurs impaires et pas un nombre d'erreurs paire.

1.7.3 Le code ISBN

ISBN (International Standard Book Number) est un numéro international qui identifie d'une manière unique chaque livre publié, son but est de simplifier la gestion informatique des livres dans les bibliothèques et autre lieus contenant des livres.

ISBN 10

L'ISBN-10 comporte 4 segments A - B - C - D séparés par des tirets (ISBN 2-1234-5680-2

- A : de un à cinq caractères pour désigner la zone géographique. Le 0 et 1 sont réservés à la zone anglophone, 2 pour la zone francophone, 3 pour la zone germanophone, .. etc
- B : de un à sept caractères pour identifier l'éditeur.
- C : de un à six caractères pour identifier l'ouvrage chez l'éditeur.
- D : Un code clé de vérification sur un chiffre de 0 à 9 ou X (qui désigne 10), il est obtenu en faisant un calcul sur les autres chiffres.

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

- On attribue une pondération à chaque position (de 10 à 2 dans le 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 est 11 la clé de contrôle est 0 par convention.
- Si le reste de la division est 10 la clé de contrôle est X par convention.

Exemple

Pour le numéro ISBN (à 9 chiffres) 2-940043-41, quelle est la clé de contrôle ?

Code ISBN	2	9	4	0	0	4	3	4	1
Pondération	10	9	8	7	6	5	4	3	2
Produit	20	81	32	0	0	20	12	12	2

La somme des produits est alors **179**, le reste de la division par 11 est donc **3**. La clé de contrôle est donc 11-**3** = **8**. L'ISBN au complet devient 2-940043-41-**8**. La vérification de la clé complète à 10 chiffres donne la somme pondérée 179+8 = 187, qui est bien un multiple de 11.

ISBN 13

Depuis Janvier 2007, chaque livre comporte obligatoirement un code à barre à la norme EAN-13 (European Article Numbering) qui contient un code ISBN-13 sans les tirets. En fait ISBN-13 reprend les 9 premiers chiffres de ISBN-10 précédé de 978 pour tous les livres publiés avant fin 2006. Les nouveaux éditeurs peuvent recevoir des codes qui commencent aussi par 979 et qui n'ont pas un équivalent en ISBN-10. Le dernier chiffre est un code de

contrôle de 0 à 9 calculé à partir des premiers chiffres (Exemple dans la figure ce-dessous).

Générer ISBN-13(EAN-13) à partir d'un ISBN

- Garder les 9 premiers chiffres de l'ISBN.
- Ajouter le préfixe 978 ou 979.
- Entrer les facteurs de pondération constants associés à chaque position de l'EAN (1 3 1 3 1 3 1 3 1 3 1 3).
- Multiplier chaque chiffre par le facteur de pondération qui lui est associé.
- Diviser la somme par le nombre du module (10) pour trouver le reste.
- Si le reste est 0, le numéro de contrôle est aussi 0. Sinon, le numéro de contrôle est 10 moins le reste trouvé et on ajoute ce suffixe à la fin.

Exemple

Convertissons l'ISBN 0-8436-1072-7

ISBN				0	8	4	3	6	1	0	7	2
Ajout préfixe 978	9	7	8	0	8	4	3	6	1	0	7	2
Facteurs	1	3	1	3	1	3	1	3	1	3	1	3
Produits	9	21	8	0	8	12	3	18	1	0	7	6

La somme obtenu est 93, le reste de la division de ce résultat par 10 est 3, donc 10 - 3 = 7 et le chiffre de contrôle est alors 7.
Le code EAN-13 (ISBN-13) est alors 978-0-8436-1072-7.

1.7.4 Code de Hamming

Un code de Hamming permet la détection et la correction automatique d'une erreur si elle ne porte que sur un bit 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.

On fixe un entier k et on code chaque bloc de $m = 2^k - k - 1$ bits par un bloc de $n = 2^k - 1$ bits obtenu en ajoutant à certaines positions dans un bloc de m bits (dits bits d'information) un ensemble de k bits de correction,c'est un code C(n,m).

Position des k bits de correction

Les k bits de correction sont placés dans le bloc envoyé aux positions d'indice une puissance de 2 en comptant à partir de la gauche. Ainsi, en notant k_1 k_2 k_3 les bits de correction et m_1 m_2 m_3 m_4 les bits de données, le bloc envoyé est : $A = k_1 k_2 m_1 k_3 m_2 m_3 m_4$.

Calcul des k bits de correction

Les k bits de correction sont calculés en utilisant une matrice de parité H, représentée ci-dessous pour k=3.

$$H = \begin{pmatrix} 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{pmatrix} \tag{1.3}$$

La colonne d'indice i de H est la représentation en binaire de i. Les n-m bits de correction sont choisis de telle façon que si le bloc a est écrit comme un vecteur colonne, on ait

$$HA = 0 \tag{1.4}$$

où les opérations sont faites en binaire avec la règle 1+1=0. Les bits de correction sont ceux situés aux indices puissance de 2.

Exemple

Soit m = [1110] le message qu'on veut transmettre

[1110] Codage -> A=[0010110] Transmission -> C=[0110110]

Le message m=[1110] est codé par le bloc A = $[a_1, a_2, a_3, a_4, a_5, a_6, a_7]$ = [0010110]

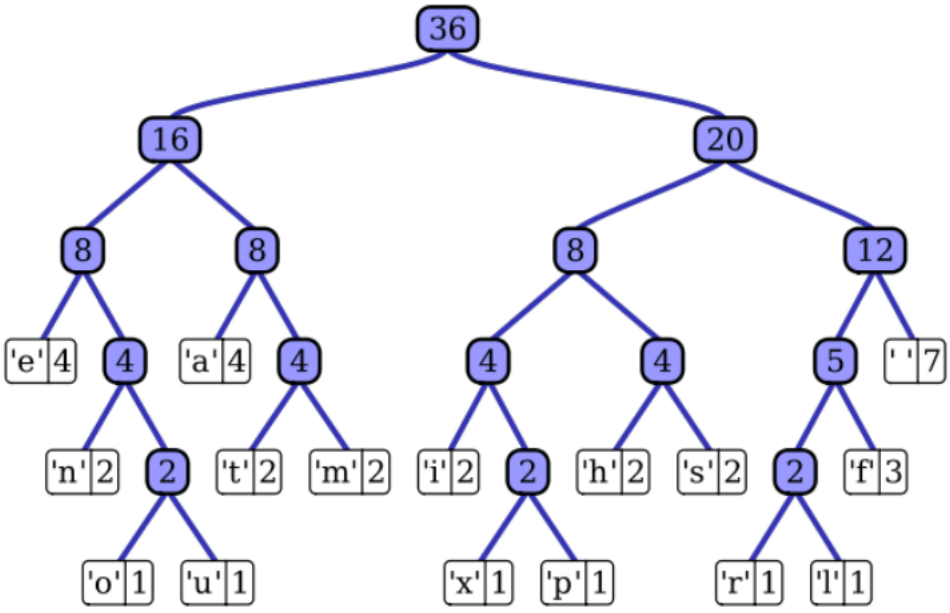
obtenu en ajoutant trois bits de correction a_1, a_2, a_4 définis par les équations :
$a_1 + a_3 + a_5 + a_7 = 0$
$a_2 + a_3 + a_6 + a_7 = 0$
$a_4 + a_5 + a_6 + a_7 = 0$

Détection et correction d'une erreur

On continue sur l'exemple précédent qui s'agit d'envoyer un message A=[0010110].
On considère qu'une erreur pendant la transmission a eu lieu et que le message A est devenu un message C=[0110110].
Le deuxième bit a été modifié par une erreur et on reçoit le bloc C=A+E avec E=[0100000]. Elle est corrigée au décodage en calculant le vecteur S=HC.
On obtient ainsi le vecteur HE qui est la deuxième colonne de H et on change donc le deuxième bit de C de 1 en 0.
Le vecteur HE indique la position d'une colonne dans H, le numéro de cette position n'est que le numéro de la position du bit erroné dans C.

1.8 Codage de Huffman [2]

Le codage de Huffman (1952) est une méthode de compression statistique de données qui permet de réduire la longueur du codage d'un alphabet. Cela signifie que l'on va d'abord calculer les fréquences des caractères du texte (ou les couleurs d'une image). Puis on va construire un arbre binaire (voir ci-dessous) dont les extrémités seront les lettres. Les lettres les plus fréquentes seront en haut de l'arbre, les moins fréquentes en bas. Le codage sera obtenu en parcourant l'arbre depuis le haut en allant vers la lettre : un mouvement à gauche sera codé 0 et un mouvement à droite sera codé 1. Ainsi, dans l'arbre ci-dessous, « a » sera codé « 010 », et « x » « 10010 ».



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

Le principe

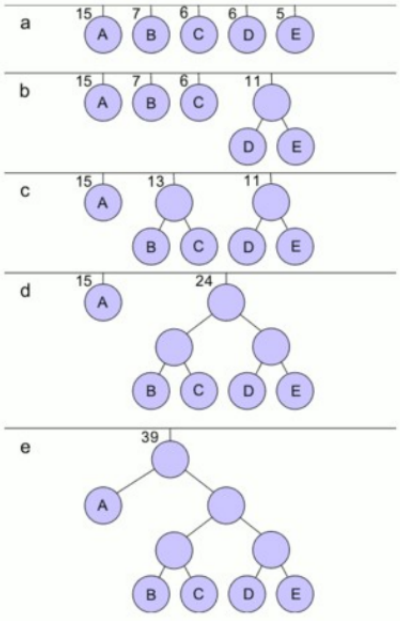
Huffman propose de recoder les données qui ont une occurrence très faible sur une longueur binaire supérieure à la moyenne, et recoder les données très fréquentes sur une longueur binaire très courte. Ainsi, pour les données rares, nous perdons quelques bits regagnés pour les données répétitives.

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. 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.

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.

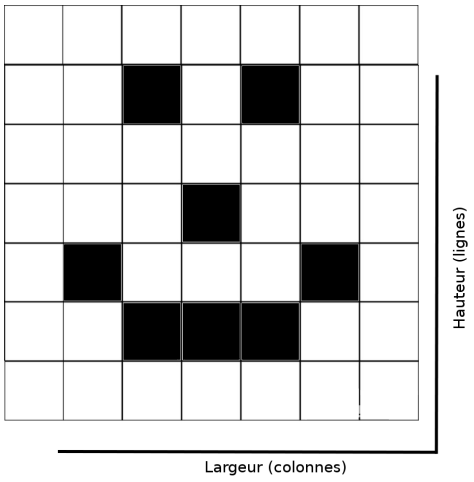
Malgré son ancienneté, cette méthode est toujours remise au goût du jour, et offre des performances appréciables.

1.9 Image numérique

1.9.1 Composition et caractéristiques

Le pixel

Une image numérique est composée d'un ensemble de points qu'on appelle pixels (PICture ELement). L'ensemble de ces pixels forment un tableau à deux dimensions constituant une image :

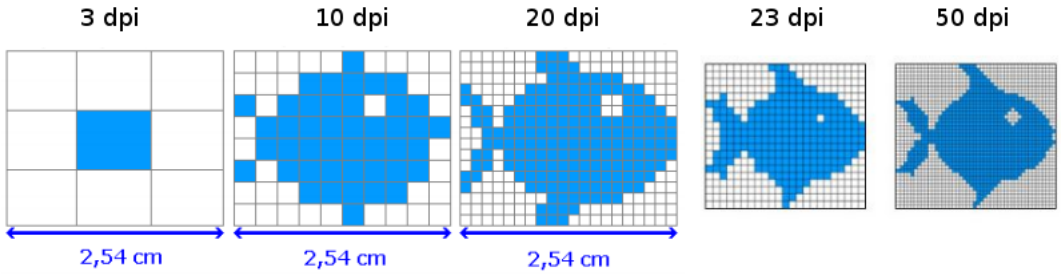


La définition

On appelle définition le nombre de pixels constituant une image, c'est en effet le nombre de colonnes de l'image multiplié par son nombre de lignes. Si une image possède 7 colonnes et 7 lignes sa définition est $7 \times 7 = 49$

La résolution

C'est le nombre de pixels contenu dans une longueur donnée (en pouce). Elle est exprimée en points par pouce (PPP, en anglais DPI : Dots Per Inch) sachant qu'un pouce est équivalent à 2.54 cm. La résolution permet ainsi d'établir le rapport entre la définition en pixels d'une image et sa dimension réelle sur un support physique (écran, papier ...)



1.9.2 Codage des couleurs ou profondeur des couleurs

En plus de sa définition, l'image numérique utilise d'avantage de mémoire pour coder ses couleurs selon le type de codage qu'elle possède, c'est ce qu'on appelle le codage de couleurs ou profondeur de couleurs qui est exprimée en bit par pixel (bpp) : 1, 4, 8, 16 bits ...

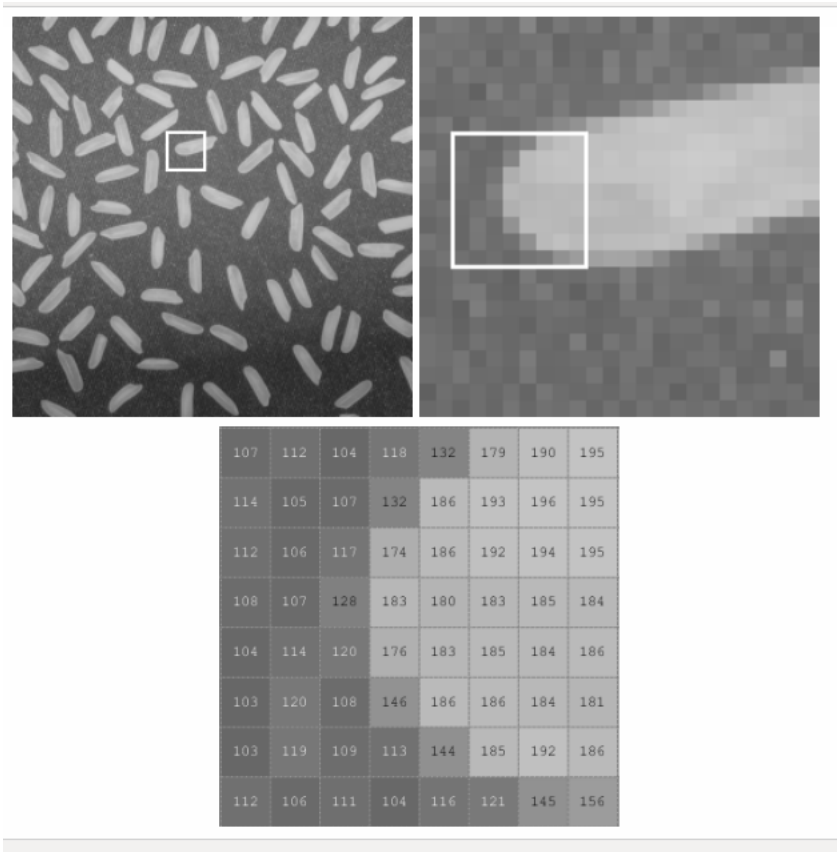
On peut définir exactement l'espace mémoire que peut occuper une image (son poids) en connaissant sa dimension et la mémoire nécessaire à l'affichage d'un pixel.

Exemple

Le poids d'une image de 640x480 codée sur 1 bit est 307200 bits = 38400 octets = 38.4 Ko

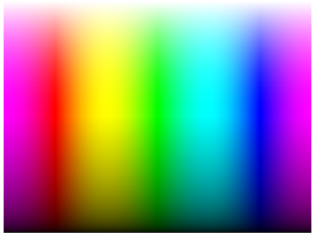
Les images les plus simples contiennent une valeur d'intensité, codée sur un nombre fini de niveaux de gris. Généralement, le nombre de niveaux de

gris est codé sur 256 valeurs, le noir correspondant à la valeur 0 et le blanc à la valeur 255 (voir figure ci-dessous)



Pour afficher les couleurs sur un écran on utilise le système RVB (Rouge, Vert, Bleu) qui permet en mélangeant des variations de ces trois couleurs de créer toutes les autres couleurs visibles. En général sur les ordinateurs modernes on choisit d'utiliser 256 nuances par couleur RVB, ce qui donne 256x256x256 couleurs possibles soit 16 777 216 possibilités de couleurs différentes.

En général, toutes ces couleurs sont représentées comme ça, à la manière d'un « arc-en-ciel » :



Bibliographie

[1] Digital Fundamentals, Thoms L. Floyd.

[2] Informatique (presque) débranchée, chapitre 3, Didier Müller.

[3] Mémoire de fin d'études : Application des codes correcteurs d'erreurs Reed Muller, Khadija Serir.

[4] Cours "Codes détecteurs correcteurs" de Arnaud Labourel, Université de Provence.

[5] http ://www.laboitealire.com/FAQRetrieve.aspx ?ID=46276

[6] http ://www.pfl-cepia.inra.fr/index.php ?page=tutoImg-niveaux-de-gris

[7] https ://www.laboiteverte.fr/toutes-les-couleurs-rvb-sont-dans-ces-images/

[8] https ://www.enseignement.polytechnique.fr/profs/informatique/Georges.Gonthier/pi97/perri