Informatique et sciences du numérique

De Dimitri PIANETA

Edition 2016

AVANT-PROPOS

Ce livre est un complément des livres de cette matière de terminale S et approfondit par les connaissances ingénieurs en Informatique par l'auteur. L'auteur de ce livre est ancien de l'école d'ingénieur généraliste du Groupe ESEO d'Angers et diplômé d'un master 2 en mathématiques et Images de l'université de La Rochelle. Passionné par l'informatique depuis sa première technologie et devenu autodidacte en apprentissage scientifique et de langage informatique.

Revenons un siècle en arrière l'ordinateur n'existait pas et actuellement il y a plusieurs milliards d'ordinateur et maintenant des tablettes et des smartphones. L'ordinateur fait partie des autres appareils numériques comme les réseaux, les téléphones, les baladeurs, appareils photos, les robots, etc. ont changé la manière dont nous :

- Concevons et fabriquons des objets
- Échangeons des informatiques entre personnes comme le sms et le mail
- Gardons trace de notre passé avec les bases de données et les clouds
- Faisons de la science
- Créons et diffusons des œuvres d'art
- Administrons les états
- Etc.

Si les ordinateurs ont tout transformé, c'est parce qu'ils sont polyvalents, ils permettent de traiter des informations de manières très diverses. C'est en effet le même objet qui permet d'utiliser des logiciels de conception assistée par ordinateur, des machines à commande numérique, des logiciels de modélisations et de simulation, des encyclopédies, des courriers électroniques….

Table des matières

PREMIER PARTIE

Information numérique

1 Codage numérique

1.1 Le binaire

Le binaire est unité le plus petite en informatique. Elle est représentée de 0 et 1. On a comme unité bit (binary digit). Ce mot a été inventé par Claude Shannon en 1948.

Si on fait le parallèle à l'électricité avec deux courants dans un fil.

On obtient le tableau suivant :

Courant Fil 1	Courant Fil 2	Binaire	Décimal
Ne passe pas	Ne passe pas	00	0
Ne passe pas	passe	01	1
Passe	Ne passe pas	10	2
Passe	Passe	11	3

Le binaire comme l'indique est une base de deux comme le décimal utilise une base dix (10). En décimal, tous les nombres peuvent être représentés à l'aide de puissances de 10. Prenez le nombre 1234.

On utilise la **méthode de substitution** :

$1*10^3 +2*10^2+3*10^1+4*10^0=1234$

L'unité est représentée par 10^0, la dizaine par 10^1, la centaine par 10^2, et ainsi de suite. Pour convertir le binaire en une valeur décimale plus lisible, il faut utiliser les puissances de 2, la plus petite valeur étant la plus à droite et est appelée **bit du poids fiable**, la plus grande la plus à gauche et est appelée **bit de poids fort**.

Les informaticiens et mathématiciens utilisent une notation particulière en indice pour indiquer des conversions :

$11_{(2)} = 3_{(10)}$

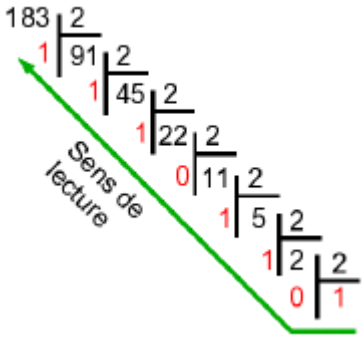
Il est possible de convertir du décimal en binaire avec des divisions successives (divisions euclidienne) : on divise les résultats sans la virgule successivement par deux. Le résultat binaire sera la juxtaposition des restes (0 ou 1) sauf pour le dernier.

Par exemple avec le nombre 183 :

- 183/2=91, reste 1
- 91/2=45, reste 1
- 45/2=22, reste 1
- 22/2=11, reste 0
- 11/2=5, reste 1
- 5/2=2, reste 1
- 2/2=1, reste 0

On remonte du dernier : 10110111

Schéma du calcul :



1.2 Les octets

Un ordinateur sait manipuler individuellement chaque bit d'une valeur. Mais les bits ne sont pas stockés individuellement dans une case mémoire. Ils sont regroupés, généralement par multiples de huit (8). Ainsi un ensemble de 8 bits est appelé un **octet**.

En anglais, on voit la notation B qui est Byte à ne pas confondre avec bit.

1.3 L'hexadécimal :

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

Si vous représentez le nombre 183 qui nécessite 8 bits soit un octet, sa conversion donne B7 en hexadécimal. On dit que : $183_{(10)}=B7_{(16)}$

Soit le calcul :

1.4 Un mot

Un mot (machine) est la quantité de bits standards manipulés par un microprocesseur (CPU). La taille du mot s'exprime en bits (ou en octet).

Un microprocesseur est d'autant plus performant que ses mots sont longs, car les données qu'il traite à chaque cycle sont plus nombreuses. C'est pourquoi on classe les microprocesseurs par la taille de leur mot : 8, 16, 32, 64 bits.

La valeur maximum d'un mot est :

2^{16}-1 = 65 535 = 0xFFF

1.5 Calculs avec le binaire

- Signe :

Le + est représenté par 0 et – par un 1.

Le nombre négatif est représenté par le complément à 2.

- ✓ Prendre la notation binaire positive du nombre
- ✓ Faire le complément à 1 (inverse des chiffres)
- ✓ Ajouter 1 (complément à 2) au nombre obtenu

Remarque : 1er bit indique nombre positif (0) ou négatif(1).

➔ Nombres représentés sur 1 octet allant de -128 (-2^7) à 127(2^7-1).

Exemple pour un entier négatif :

25_{10} sur un octet : 00011001_2

Complément à 1 : 11100110_2

Ajouter 1 (complément à 2) : 11100111_2

Ce qui nous donne alors -25_{10} sur 1 octet : 11100111_2

- Addition binaire (commutative) :

0 + 0 = 0

0 + 1 = 1

1 + 1 = 10

7		00000111	16		00010000
+4		00000100	+(-24)		11101000
+11		00001011	-8		11111000
15		00001111	-5		11111011
+(-6)		11111010	+(-9)		11110111
+9	1	00001001	-14	1	11110010

- Soustraction binaire

0 – 0 = 0

1 – 0 = 1

1 – 1 = 0

10 – 1 = 1

Même principe qu'en décimal.

12	00001100
-9	00001001
3	00000011

-25	11100111
-(+19)	00010011
-44	11010100

Addition avec conversion des nombres en cplt à 2 si besoin.

-120	10001000
-(-30)	00011110
-90	10100110

-25		11100111
-(+19)		11101101
-44	1	11010100

(Explication)

$10_{(2)}$

- $01_{(2)}$

= $01_{(2)}$ (2-1)

- Multiplication binaire :
  - Déterminer le signe du produit.
  - Exprimer tout nombre négatif en complément à 2 (conversion
  - des nombres négatifs en nombres positifs).
  - Effectuer le produit (même principe qu'en décimal).
  - Si le signe du produit est négatif, prendre le complément à 2
  - du résultat du produit.

0 x 0 =0

0 x 1 = 0

1 x 1 =1

- Division binaire :

0/1 = 0

1/1 = 1

10/1 = 10

- Conversion binaire-décimal :

Soient n le nombre de bits du nombre nb dans sa notation en base 2, b_i chaque bit du nombre et p_i le poids de chacun des bits (i allant de 0 à $n - 1$).

$$nb = \sum_{i=0}^{n-1} b_i \times p_i$$

Exemple : 00011001_2

$$0 \times 2^7 + 0 \times 2^6 + 0 \times 2^5 + 1 \times 2^4 + 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^0 = 2^4 + 2^3 + 2^0 = 25_{10}$$

Méthode de la division euclidienne par 2 (répétée) :

Exemple :

$$\begin{aligned} 25_{10} &= 12 \times 2 + 1 \\ &= (6 \times 2) \times 2 + 1 \\ &= (3 \times 2) \times 2^2 + 1 \\ &= (2 + 1) \times 2^3 + 1 \\ &= 2^4 + 2^3 + 2^0 \end{aligned}$$

Sur 1 octet : 25_{10} est égale à 00011001_2

1.6 Algèbre de Boole

- Ensemble b
  Constitué de 2 éléments :

$$B = \{vrai, faux\}$$

$$B = \{1, 0\}$$

$$B = \{\top, \bot\}$$

- Fonctions logiques de base :
  - Disjonction : a+ b
  - Conjonction : a.b
  - Négation : $\bar{a}$
- Table de vérité
  - Disjonction : OU

a	b	a+b
0	0	0
0	1	1
1	0	1
1	1	1

  - Conjonction : et

a	b	a.b
0	0	0
0	1	0
1	0	0
1	1	1

  - Conjonction : Non

a	$\bar{a}$
0	1
0	0

- Axiomes
  - Associativité

(a+b)+c = a+(b+c) = a+b+c
(a.b).c= a.(b.c)=a.b.c

  - Commutativité

a+b = b+a
a.b=b.a

  - Distributivité

a.(b+c) = a.b+a.c
a+(b.c) = (a+b).(a+c)

- Élément neutre : a+0=a, a.1 =a
- Complément : $a + \bar{a} = 1, a.\bar{a} = 0$

- Théorèmes :
  - Élément absorbant : a+1=1, a.0=0
  - Absorption :

    a+(a.b)=a

    a.(a+b)=a

  - Idempotence : a+a=a, a.a=a
  - Involution : $\bar{\bar{a}} = a$
  - De Morgan :

$$\overline{a + b} = \bar{a}.\bar{b}$$

$$\overline{a.b} = \bar{a} + \bar{b}$$

1.7 Calcule de la virgule flottante

Prenons exemple classique de la racine carré de deux avec 10 000 décimales avec la méthode de Héron ou algorithme de Babylone.

On va utiliser la formule classique :

$$x_{n+1} = \frac{1}{2}(x_n + \frac{A}{x_n})$$

On devra avoir comme résultat : $\sqrt{2}$ = 1, 414 213 562 373 095 048 …

Pour A est égale à 2, je pose i qui sont les valeurs de l'incrémentation de l'algorithme de 1 à 6 ici. Dans le tableau suivant montre le calcul de l'algorithme :

i	**½ x_n**	**½(A/x_n)**	**x_{n+1}**	**Valeur de x_{n+1}**
1	½	1	3/2	1,5
2	¾	2/3	17/12	1,41666
3	17/24	12/17	577/408	1,41421568
4	577/816	408/577	665857/470832	1,414213563746
5	665857/941664	470832/66587	88673108897/627013566048	1,4142135623730950488
6	88673108897/1254027132096	627013566048/886731088897	157258404803291863353217/111198484434986813793112	1,4142135623730950488

Algorithme en Python :

```
'''
Algorithme de Babylone

@author: Dimitri pianeta
'''
import math

def algoHeron(ndec, niter, A):
    u = 1
    Avalu = A

    for i in range(niter):
        u = (u +Avalu/u)/2
    return round(u,ndec)



if __name__ == '__main__':

        print(algoHeron(6,3,2))
```

On va maintenant étendre mon exemple précédent de la calcule de la racine carré avec la méthode de la vision euclidienne.

On utilise le principe de la division des nombres premiers.

Soit $\sqrt{780.14}$:

On réécrit sous la forme d'une division :

(1)

780.14 | 2

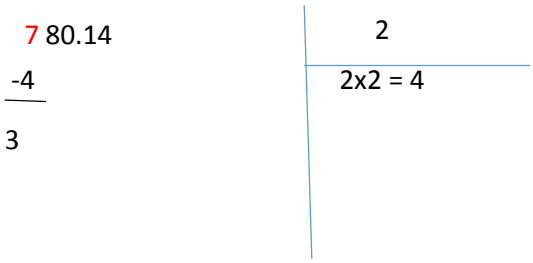
2 x 2 =4

On cherche à trouver le plus grand entier n dont le carré est inférieur ou égal à la paire plus à gauche. En effet, on commence toujours avec la « tranche » (paire) le plus à gauche, que ce soit une paire ou pas. Trouvez le plus grand carré est inférieur ou égal à cette paire et prenez la racine carrée de ce carré parfait. Ce nombre, c'est n. Inscrivez n en haut à droite et inscrivez son carré (n^2) en bas à droite de la division.

Dans notre cas, la tranche qui est le plus à gauche est 7. Nous savons que :

$2^2 \leq 7 \leq 3^2$, on peut alors dire que n=2 c'est le plus grand entier dont le carré est inférieur ou égal à 7. Inscrivez 2 dans le secteur supérieur droit. C'est le premier chiffre de votre réponse finale. Inscrivez 4 (carré de 2) en bas à droite de la division. Ce nombre aura son importante à la prochaine étape.

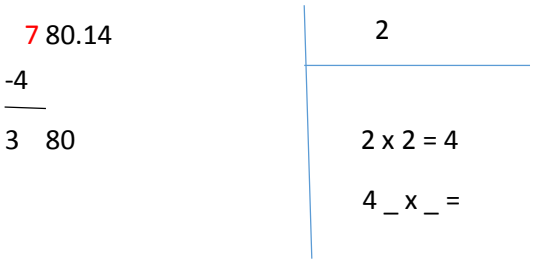
(2)



Soustrayez le nombre que vous venez de calculer de la paire la plus à gauche. Ce carré que vous venez de calculer, vous devez le retrancher de la tranche de départ. Inscrivez ce nombre juste sous la première tranche et faites la soustraction. Inscrivez les résultats en dessous.

Dans notre exemple, il faut écrire 4 sous 7, et faite la soustraction, ce qui nous donne 3.

(3)

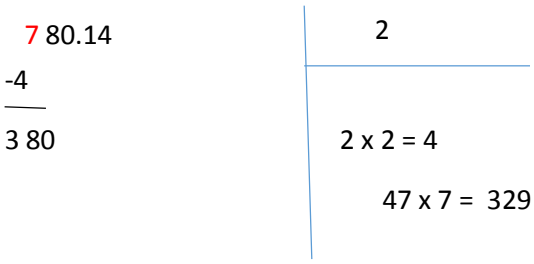


Abaissez la prochaine paire. Il faut maintenant abaisser au niveau du 3 de la soustraction la paire suivante, celle qui se trouve juste à gauche. Multipliez mentalement par 2 le nombre qui se trouve dans le coin supérieur droit de la division et inscrivez le résultat dans l'espace en bas à droite de la même division, en dessous de votre première ligne de calcul. Sur cette deuxième ligne, à côté du chiffre que vous avez mis, ajoutez '"_ x _="'.

- Dans notre exemple, la prochaine paire est "80". Inscrivez "80" à côté du reste qu'on a trouvé, soit le 3. On multiplie mentalement par 2 le nombre qui se trouve dans le coin supérieur droit de la division. Ici, c'est 2, donc 2 x 2 = 4. Inscrivez "'4"' en bas à droite, sur la deuxième ligne, puis on ajoute **_ x _=**

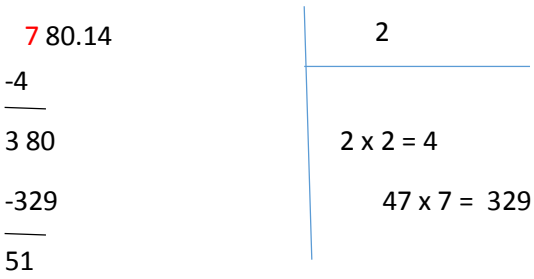
(4) **Reste à remplir les blancs.** Ces deux "trous" doivent être remplis avec le même entier. La multiplication devra donner un résultat inférieur ou égal au nombre qui se trouve à gauche.

- Dans notre exemple, si on met par exemple 8, on obtient : 4(8) x 8 = 48 x 8 = 384. C'est plus grand que 380. 8 est donc trop grand mais de peu. Essayons 7. On a donc : 4(7) x 7 = 329. Parfait ! 329 est inférieur à 380. Inscrivez alors 7 dans le secteur supérieur droit. C'est le second chiffre de votre résultat, la future racine carrée de 780,14.



(5) **Comme précédemment, soustrayez ce produit (en bas à droite) du nombre initial (à gauche).** Posez l'opération et faites-la. Marquez le résultat.

- Dans notre exemple, vous devez ôter 329 de 380, ce qui nous donne **51**.



(6)

Abaissez la paire suivante du nombre initial. Quand vous allez arriver à la virgule, mettez une virgule dans le coin supérieur droit. On multiplie mentalement le nombre de ce coin-là par 2 et ce résultat, on l'inscrit dans la partie inférieure droite, sur la troisième ligne et on ajoute ("_ x _").

- Dans notre exemple, on rencontre en effet la virgule de 780,14, donc, on en met une après le 27 en haut à droite. On abaisse à gauche la nouvelle paire -14 -. On multiplie 27 par 2, soit 54 qu'on inscrit dans la partie inférieure droite, sur la troisième ligne, ce qui nous donne : "54 _ x _=".

7 80.14	2
-4	
3 80	2 x 2 = 4
-329	47 x 7 = 329
51 14	54_x_=

(7)

Comme tout à l'heure, il reste à remplir les blancs. Ces deux "trous" doivent être remplis avec le même entier. La multiplication devra donner un résultat inférieur ou égal au nombre qui se trouve à gauche.

- Dans notre exemple, 549 x 9 = 4 941, ce qui est inférieur à 5 114. Il n'y a pas d'autre réponse possible ! Vous mettez donc un 9 dans le coin supérieur droit et vous ôtez 4 941 de 5 114 dans la partie gauche : 5 114 - 4 941 = 173.

7 80.14	2
-4	
3 80	2 x 2 = 4
-329	47 x 7 = 329
5114	549x9=4941
- 4941	
173 00	

(8) Si vous devez continuer, il faut alors abaisser une paire de 0, puis répétez les étapes 3, 4, et 5.

1.8 Codage de l'information

Codage du texte :

Les ordinateurs est code historiquement en ACII (American Standard Code for Information Interchange). Il s'agit d'un code sur 7 bits permettant de représenter les lettres utilisées dans les langues anglo-saxonnnes- dons pas les caractères accentués, ainsi que des caractères de contrôles utilisés par les télétypes[1] à l'époque. Des extensions à ce code ont été ajoutées en utilisant huit bits par la suite pour permettre la représentation des caractères nécessaires à l'écriture de l'ensemble des langues, aucune extension ne permettait un codage exhaustif.

L'évolution des technologies a permis l'émergence d'un nouveau standard, appelé Unicode. Cette norme défini les caractères et symboles graphiques utiles, leurs représentations et leurs codages suivant la taille des mots utilisés (8 bits, 16 bits et 32 bits).

Le codage ASCII :

- 128 caractères (jeu minimal) : 33 non imprimables, 94 imprimables et l'espace (considéré comme invisible)
- Des limitations claires (US-ASCII), liées à son histoire
- Jusque fin 2007, l'encodage le plus répandu (UTF-8 depuis)
- 32 premiers caractères de contrôle
    - Caractère 10 : LF (Line Feed) permet de faire avancer le papier dans une imprimante
    - Caractère 8 : BS (backspace, retour arrière)
- Pas de description de la structure d'une page (langage de balisage)
- Beaucoup d'extensions et de propositions nationales pour utiliser le huitième bit et coder les caractères suivants
- Certaines extensions non compatibles avec le jeu minimal

[1] C'est un appareil permettant la génération et la réception de messages via des signaux électriques.

Table Ascii :

Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char
0	0	[NULL]	32	20	[SPACE]	64	40	@	96	60	`
1	1	[START OF HEADING]	33	21	!	65	41	A	97	61	a
2	2	[START OF TEXT]	34	22	"	66	42	B	98	62	b
3	3	[END OF TEXT]	35	23	#	67	43	C	99	63	c
4	4	[END OF TRANSMISSION]	36	24	$	68	44	D	100	64	d
5	5	[ENQUIRY]	37	25	%	69	45	E	101	65	e
6	6	[ACKNOWLEDGE]	38	26	&	70	46	F	102	66	f
7	7	[BELL]	39	27	'	71	47	G	103	67	g
8	8	[BACKSPACE]	40	28	(	72	48	H	104	68	h
9	9	[HORIZONTAL TAB]	41	29	)	73	49	I	105	69	i
10	A	[LINE FEED]	42	2A	*	74	4A	J	106	6A	j
11	B	[VERTICAL TAB]	43	2B	+	75	4B	K	107	6B	k
12	C	[FORM FEED]	44	2C	,	76	4C	L	108	6C	l
13	D	[CARRIAGE RETURN]	45	2D	-	77	4D	M	109	6D	m
14	E	[SHIFT OUT]	46	2E	.	78	4E	N	110	6E	n
15	F	[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	[NEGATIVE 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	[ENG OF 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	[SUBSTITUTE]	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	[DEL]

Codage URF-8 :

Signifie Universal Character Set Transformation Format – 8 bits

- Permet de représenter les (milliers de) de caractères du répertoire universel (UCS), commun à la norme IS/CEI 10646 et au standard Unicode[2].
- Compatible avec US-ASCII
- UTF-8 est approuvé par l'IETF[3] et le W3C[4] .
- UTF-8 inventé en 1992 par K. Thompson (Unix[5])

Le principe de fonctionnement de l'UTF-8 :

1) Les points de code ayant une valeur (scalaire, en base 10) comprise entre 0 et 127 sont codés sur un seul octet dont le bit de poids fort est nul.
2) Les points de code de valeur supérieure à 127 sont codés sur plusieurs octets :
    - Les bits de poids forts[6] du premier octet forment une suite de 1 de longueur égale au nombre d'octets utilisés pour coder le caractère
    - Les octets suivants ont 10 comme bits de poids forts

[2] On l'appelle aussi Universal Code. Unicode est un standard informatique qui permet des échanges de textes dans différentes langues, à un niveau mondial.
[3] Internet Engineering Task Force
[4] World Wide Web Consortium
[5] Unix est le système d'exploitation sur la base du langage linux.
[6] L'octet à gauche du mot binaire

Exemples : Cette exemple explique le terme point de code citée ci-dessus et la conversion en valeur scalaire et sa représentation en codage UTF-8 en binaire.

Type	Caractère	Point de code (hexadécimal)	Valeur scalaire		Codage UTF-8	
			décimal	binaire	binaire	hexadécimal
Contrôles	[NUL]	U+0000	0	*0000000*	**0***0000000*	00
	[US]	U+001F	31	*00*11111	**0***00*11111	1F
Texte	[SP]	U+0020	32	*0*100000	**0***0*100000	20
	A	U+0041	65	1000001	**0**1000001	41
	~	U+007E	126	1111110	**0**1111110	7E
Contrôles	[DEL]	U+007F	127	1111111	**0**1111111	7F
	[PAD]	U+0080	128	*000*10 000000	**110***000*10 **10**000000	C2 80
	[APC]	U+009F	159	*000*10 011111	**110***000*10 **10**011111	C2 9F
Texte	[NBSP]	U+00A0	160	*000*10 100000	**110***000*10 **10**100000	C2 A0
	é	U+00E9	233	*000*11 101001	**110***00*011 **10**101001	C3 A9
	ß	U+07FF	2047	11111 111111	**110**11111 **10**111111	DF BF
	✀	U+0800	2048	*0000* 100000 000000	**1110***0000* **10**100000 **10**000000	E0 A0 80
	€	U+20AC	8 364	*00*10 000010 101100	**1110***00*10 **10**000010 **10**101100	E2 82 AC
	퟿	U+D7FF	55 295	1101 011111 111111	**1110**1101 **10**011111 **10**111111	ED 9F BF

2 Représentation numérique des données analogiques

2.1 Domaines d'applications :

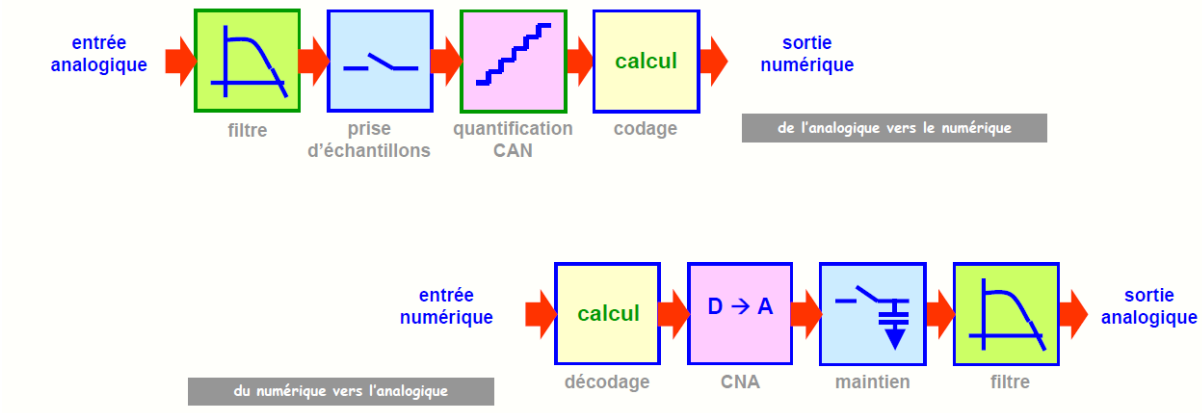
Dans ce chapitre, les notions sont souvent utiliser pour les traiteurs de signal 1D, 2D[7] et 3D.

Je vais décomposer les domaines le plus utilisé la numérisation des données numériques :

- Signal : synthèse de son, reconnaissance vocale
- Image : imagerie médicale, image satellitaire, astronomie, vision par ordinateur
- Modèles 3D : C.A.O[8], jeux vidéo, animation
- Autres : simulation numérique, modélisation numérique

2.2 De l'analogique vers le numérique

Les signaux usuels (musique, image, signaux biologiques, données issues de capteurs …) sont le plus souvent des grandeurs analogiques continues qu'il est difficile de stocker ou de transmettre sans dégradation.



2.3 Échantillonnage et représentation de données analogiques

Dans un premier, je vais vous éclaircir la notion analogique. Les données analogiques sont les données physiques comme un signal électrique, signal sonore (une onde).

On part d'un signal s(t) qui peut être vu comme continu, par exemple une onde acoustique.

L'**Échantillonnage :** consiste à transmettre un signal en capturant des valeurs à intervalles régulier. Par exemple un signal $t_1<t_2<\ldots<t_N$ de valeurs du signal $S_n = s(t_n) \in \mathbb{R}$.

Quantification du signal : Les valeurs S_n sont codées sur un alphabet de B bits.

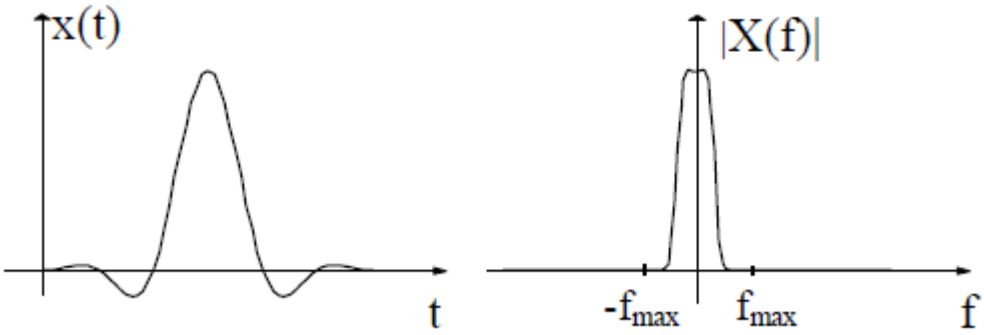
Démontrons l'échantillonnage d'un signal :

[7] D comme dimension. 1D pour les traiteurs de signales numériques et 2D pour le traitement de l'image numériques.
[8] Conception assistée par ordinateur

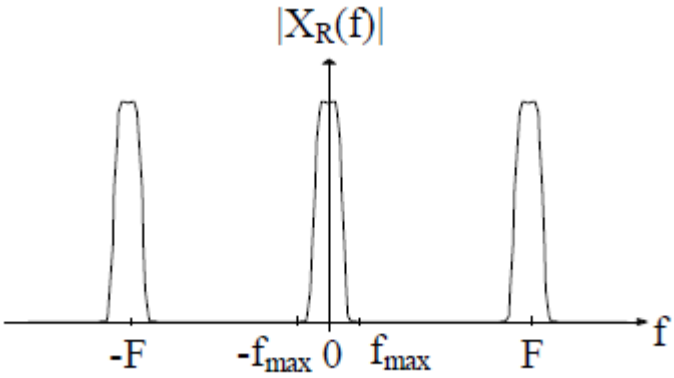
Le théorème de Shannon nous dit : *Lorsqu'un signal x(t) a un spectre à support borné [X(f)=0 pour un $|f| > f_{max}$], il est possible d'échantillonner ce signal sans perdre d'information : il suffit pour cela choisir une fréquence d'échantillonnage $f_e > 2f_{max}$. On pourra alors reconstruire x(t) parfaitement à partir des échantillons $x(nT_e)$ avec T_e (période en s)= $1/f_e$.*

Commençons ma démo :

Soit un signal analogique x(t) dont le spectre est à support borné, c'est-à-dire que sa transformée de Fourier[9] X(f) est de telle que X(f) = 0 pour $|f| > f_{max}$.



On fabrique le signal $X_R(f)$ à partir de X(f) en répétant X(f) avec la période $F \geq 2f_{max}$.



Le signal $X_R(f)$ étant périodique de période F, on peut calculer sa série de Fourier telle que :

$X_R(f) = \sum_{n=-\infty}^{+\infty} C_n e^{-j2\pi n\frac{f}{F}}$ (1)

Avec le coefficient de Fourier : $C_n = \frac{1}{F}\int_{-\frac{F}{2}}^{\frac{F}{2}} X_R(f)\, e^{j2\pi n\frac{f}{F}} df$ pour $f \in [-\frac{F}{2}, \frac{F}{2}]$ et $X_R(f)=X(f)$.

D'où $C_n = \frac{1}{F}\int_{-\frac{F}{2}}^{\frac{F}{2}} X_R(f)\, e^{j2\pi n\frac{f}{F}} df = \frac{1}{F}\int_{-\infty}^{+\infty} X(f) e^{j2\pi n\frac{f}{F}}\, df.$

On reconnaît ici la transformée de Fourier inverse de X(f), c'est-à-dire x(t), calculée au point t=n/F :

[9] Voir l'annexe sur les notations mathématiques pour la formule. La transformée permet de transformée le signal analogique en seconde vers une fréquence en Hz (s^{-1}).

$$C_n = \frac{1}{F} x\left(\frac{n}{F}\right)$$

Et en remplaçant C_n par sa valeur dans l'expression (1) :

$$X_R(f) = \sum_{n=-\infty}^{+\infty} C_n e^{-j2\pi n\frac{f}{F}} = \sum_{n=-\infty}^{+\infty} \frac{1}{F} x\left(\frac{n}{F}\right) e^{-j2\pi n\frac{f}{F}}$$

Ainsi $X(f) = \begin{cases} X_R(f) \ si\ f \in \left[-\frac{F}{2}, \frac{F}{2}\right] \\ 0 \ ailleurs \end{cases}$

X(f) est donc parfaitement défini par la connaissance des valeurs de x(t) aux instants t= n/F/. Il en est de même de x(t). Le théorème de Shannon est ici démontré.

Explicitons la relation liant x(t) et les valeurs x(n/F) :

$$x(t) = \int_{-\infty}^{+\infty} X(f) e^{j2\pi ft} df = \int_{-\frac{F}{2}}^{+\frac{F}{2}} X_R(f)\, e^{j2\pi ft} df = \frac{1}{F} \int_{-\frac{F}{2}}^{+\frac{F}{2}} \left(\sum_{n=-\infty}^{+\infty} x\left(\frac{n}{F}\right) e^{-j2\pi ft} \right) e^{j2\pi ft} df$$

Soit :

$$x(t) = \frac{1}{F} \sum_{n=-\infty}^{+\infty} x\left(\frac{n}{f}\right) \int_{-\frac{F}{2}}^{+\frac{F}{2}} e^{j2\pi f\left(t-\frac{n}{F}\right)} df$$

Donc : $x(t) = \frac{1}{F}\sum_{n=-\infty}^{+\infty} x\left(\frac{n}{f}\right) sinc\left(F\left(t-\frac{n}{F}\right)\right)$

Où l'on note que : $sinc(x) = \frac{sin(\pi x)}{(\pi x)}$.

En résumé, si $F>2f_{max}$, la connaissance de la suite x(n/F) est suffisante pour déterminer parfaitement x(t) ou X(f) et :

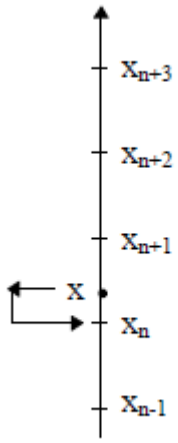
$x(t) = \frac{1}{F}\sum_{n=-\infty}^{+\infty} x\left(\frac{n}{f}\right) sinc\left(F\left(t-\frac{n}{F}\right)\right)$

$$X(f) = \begin{cases} \sum_{n=-\infty}^{+\infty} \frac{1}{F} x\left(\frac{n}{f}\right) e^{-\frac{j2\pi nt}{F}} si\ |f| \le f_{max} \\ 0\ ailleurs \end{cases}$$

Quantification :

Définition :

Quantifier une valeur x réelle appartenant à une intervalle $[-x_{max}, x_{max}]$, consiste à remplacer cette valeur x par la valeur $Q(x)= x_n$ la plus proche de x choisie dans un ensemble fini (ou dénombrable) de N valeurs réelles notées (avec n entre 0 et N-1).



La valeur quantifiée de x : Q(x) est donné par :

$$\forall x \in [x_n, x_{n+1}]$$

Si $|x - x_n| < |x - x_{n+1}|$ $\quad Q(x) = x_n$

Si $|x - x_n| > |x - x_{n+1}|$ $\quad Q(x) = x_{n+1}$

Si x = $\frac{x_n + x_{n+1}}{2}$ $\quad Q(x) = x_n \; ou \; x_{n+1}$ selon les systèmes

La quantification
Un signal numérique ne peut prendre que certaines valeurs : c'est la quantification.
Elle s'exprime en bits.
Cette quantification est assurée par un convertisseur (CAN). Chaque valeur est arrondie à la valeur permise la plus proche par défaut.
On appelle alors **résolution ou pas** l'écart (constant) entre deux valeurs permises successives.

Exemple :

Type de support	Quantification choisie	Nombre de valeur
CD audio	16 bits	65 536
DVD	24 bits	16 777 216
Téléphonie	8 bits	256
Radio numérique	8 bits	256

On appelle **calibre** l'intervalle des valeurs mesurables des tensions analogiques à numériser (par exemple ±5 V).
On appelle **plage** d'un convertisseur, la largeur de l'intervalle entre la plus petite et la plus grande valeur du calibre. (pour un calibre de ±5 V, la plage est alors de 10 V).
Le pas *p* d'un convertisseur de *n* bits et de plage donnée, est alors défini par :

p =plage/2^n

<u>Cas pratiques :</u>

- **Pour le son**
  - Fréquences perçues par l'oreille humaine 20<f<2200 Hz
  - Fréquence échantillonnage 44 kHz 5CD) jusqu'à 192kHz
  - Quantification sur 16 (CD), 24 ou 32 bits.
- **L'image**
  - dpi **:** dots per inch (1 inch =2,54 cm).

- On parle de dpi uniquement au moment de l'affichage sur un support

2.4 Conversion :

On utilise pour la numérisation de ce signal analogique par des capteurs dit convertisseur analogique vers numérique[10]. (Figure 1)

Figure 1: montre que le signal x(t) est analogique puis traiter pour donnés un signal numérique y

On a vu que la conversion de ce signal est fait des convertisseurs analogique – numérique (CAN-ADC pour Analog to Digital Converter) et le sens inverse numérique – analogique (CNA ou DAC pour Digital to Analog). Le rôle d'un CAN est de convertir un signal analogique en un signal numérique pouvant être traité par une logique numérique, et le rôle d'un CNA est de reconvertir le signal numérique une fois traité en signal analogique. (Figure 2)

Figure 2 montre la chaîne d'une conversion d'un signal

2.4.1) Principe de la conversion CAN :

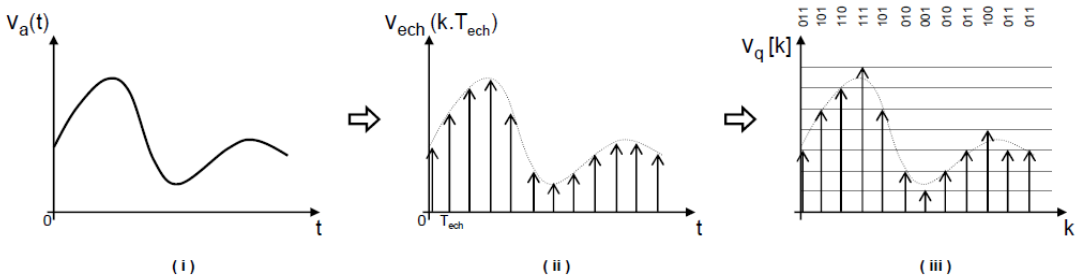
Définition : Un convertisseur analogique – numérique (CAN) est un dispositif électronique permettant la conversion d'un signal analogique en un signal numérique.

Cette première définition pour être complète en appelle deux autres, celles des signaux analogiques et numériques :

Signal analogique : signal continu en temps et en amplitude.
Signal numérique : signal échantillonné et quantifié, discret en temps et en amplitude.

Conceptuellement, la conversion analogique – numérique peut être divisée en trois étapes :

L'échantillonnage temporel, la **quantification** et le **codage**.



[10] On utilise un circuit électronique de conversion de signal pour pourvoir être traité en donnée binaire qui sera compréhensible pour le calculateur. On appelle le circuit électronique le CAN.

Figure 3 (i) signal analogique, (ii) signal échantillonné, (iii) puis quantifiée

Un signal analogique, va(t) continu en temps et en amplitude (i) est échantillonné à une **période d'échantillonnage** constante Tech. On obtient alors un signal échantillonné vech(k.Tech) discret en temps et continu en amplitude (ii). Ce dernier est ensuite quantifié, on obtient alors un signal numérique vq[k] discret en temps et en amplitude (iii). La quantification est liée à la **résolution** du CAN (son nombre de bits) ; dans l'exemple précédent vq[k] peut prendre huit amplitudes différentes (soit 23, 3 étant le nombre de bits du CAN). La figure II.1.iii présente également le code numérique sur trois bits (en code binaire naturel) associé à vq[k] en fonction du temps.

2.5 Les formats

Selon les supports, il existe différents formats :

- Images
  - Matricielle ou vectorielle
  - Propriétaire ou livre, ouvert ou fermé
  - Normalisé ou non
  - Compression avec ou sans perte

- Audios
  - Sans compression (WAV, AIFF)
  - Compression avec (MP3, OGG) ou sans perte (FLAC)

- Vidéos
  - Codecs : MJPEG, DV, MPEG
  - Conteneurs vidéos : AVI, QuicktTime, MP4, OGM

2.6 Transcodage

Le **transcodage**, en vidéo ou en audio, est le fait de changer le format de codage d'un média (voir aussi codage et codec) utilisé pour comprimer ou encapsuler un média audio ou vidéo dans un fichier ; ou transporter un signal analogique ou numérique. On notera qu'il ne s'agit pas d'un codage au sens strict du terme car le plus souvent la transformation comporte des pertes.

2.7 Opérations numériques

Quelques Exemples

- Modifications élémentaires, format, taille, fréquence, contraste
- Filtrage (son, image)
- Algorithmes pour le traitement d'images : Débruitage (filtre moyen, médian), défloutage, inpainting
- Segmentation par seuillage ou morphologie mathématique

Histogrammes et opérations élementaires

- Histogramme : Distribution des intensités de l'image.
- Seuillage : binarisation obtenue à partir d'un seuil.
- Changement de contraste : Transformation (linéaire) pour augmenter (ou réduire) la dynamique d'une image.

Tatouage numérique

- Dissimuler de l'information dans une image. Dans le domaine spatial (dans un fond uniforme par exemple), ou dans le domaine spectral pour plus de robustesse.

Inpainting

- Consiste à retrouver des parties manquantes d'une image ou à en faire disparaître d'autres. Grand nombre de méthodes, au formalisme parfois compliqué (équation aux dérivées partielles) ou parfois plus intuitif (moyennes non-locales).

3 Compressions

3.1 Qu'est-ce que la compression ?

La compression est utilisé pour audio, la vidéo, fichiers binaires et les images….

On opère dans les fichiers un changement de format avec suppression de la redondance (répétition), conservation de l'information pertinente (perçue).

Elle permet de diminuer l'espace de stockage,….

Définition 1 :

Tout signal (fichier) numérique, l'acquisition par CAN impliquant un échantillonnage et une quantification.

Objectif de la compression :

(1) Augmenter l'efficacité du stockage sur support dédié (DVD[11], super Audio CD) ou sur mémoires (clés USB, IPOD,…)

Exemple (Quelques codecs) :

- Audio : PCM(12 titres sur un CD de 700Mo)→ MP3(12 titres à 256kbit/s sur 70Mo)
- Vidéo :DVD MPEG2(1 vidéo de 720*576 sur 4,7Go)→ DivX(1 vidéo sur 700 Mo)
- Radiographie : SPIHT-3D

[11] Digital Versatile Disc

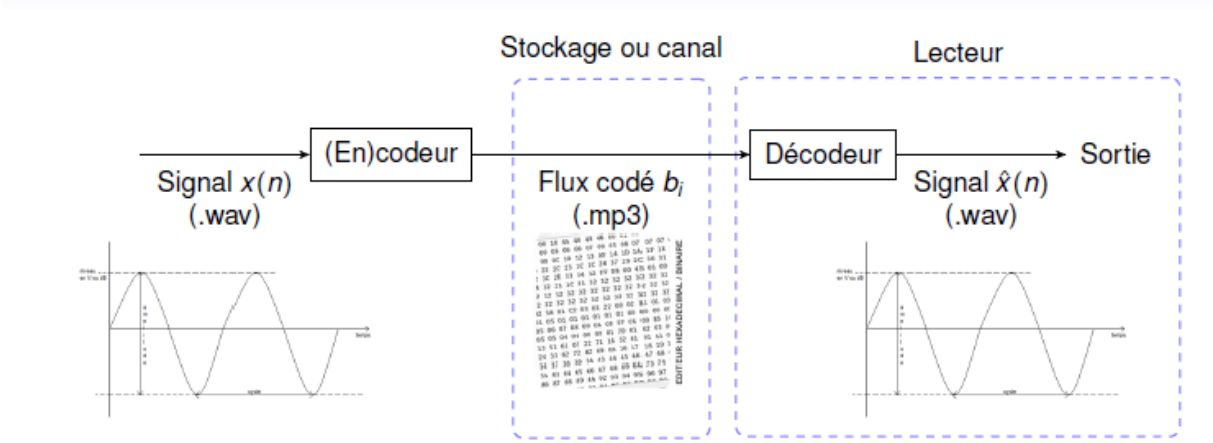
(2) Augmenter l'efficacité de la transmission sur canal de diffusion (TV/radio numérique) ou de communications (téléphonie, visiophonie sur réseaux RTC[12], RNIS[13], IP[14], GSM[15])
(3) Profiter des avantages de la communication numérique : fiabilité, contrôle, cryptage…
(4) Améliorer la description d'un signal pour augmenter l'efficacité d'un traitement aval

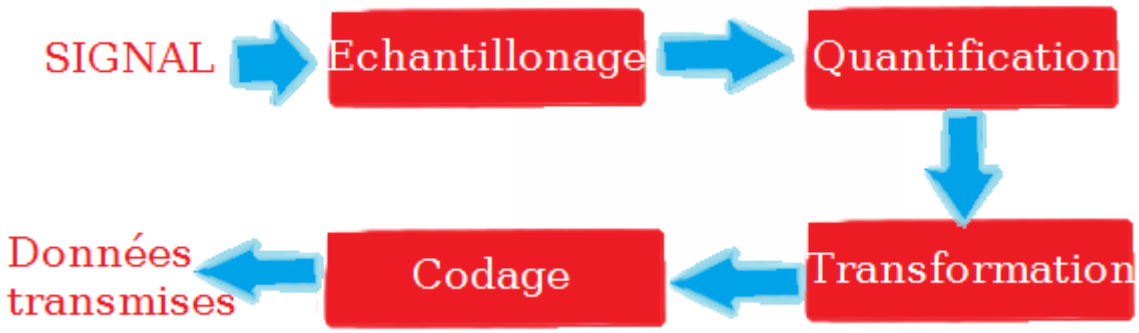
3.2 Définitions :

(1) Rechercher **l'efficacité maximum** (de moindre redondance) dans la représentation de l'information utile portée par les signaux.
(2) **Coder** : Convertir le signal en une représentation numérique efficace (séquence binaire la plus économique possible), le flux codé (bitstream[16]).
    ⇨ **Un problème d'optimisation sous contraintes qui aboutit à un système de compression**.

Vocabulaire :

Le codec est encodeur + décodeur





[12] Réseau Téléphonique Commuté, liaison de 2Mbit/s
[13] Réseau Numérique à l'Intégration de Services, liaison 2Mbit/s
[14] Internet Protocol
[15] Global System of Mobile, liaison 9,6 kbit/s
[16] Flux codé : représentation du signal compressé

3.3 Critère en jeu

Définitions :

- ⇨ **Débit** : Ressource binaire (nombre de bits) utilisée pour coder 1s de signal
- ⇨ **Taux de compression** : rapport de débits sans et avec compression
- ⇨ **Qualité** : perçue du signal restitué par rapport au signal original, évaluée par des mesures :
  - o des distorsions/dégradations
  - o du « bruit de codage » (différence entre le signal et le signal compressé)

Pour statuer sur la **transparence** du codec et sur le confort de l'utilisateur.

- ⇨ **Complexité** : Sophistication de l'algorithme, incluant :
  - o La charge de calcul par unité de temps : faisabilité du temps-réel, puissance du microprocesseur
  - o L'occupation des mémoires du système : capacité des mémoires
  - o Retard algorithme ou délai de restitution dû aux calculs : faisabilité du temps-réel

3.4 Classes applicatives :

- ⇨ Compression sans perte (noiseless coding) : compression réversible (permettant la reconstruction du signal original à l'identique)

Exemple :

- Imagerie médicale pour diagnostic
- Mélomane : FLAC

- ⇨ Compression avec pertes (noisy coding) : compression irréversible avec élimination de l'information « inutile » au prix de l'introduction de dégradations (ne permettant pas de reconstruire le signal d'origine à l'identique).

Exemple :

- ⇨ Audio (44, 1.10^3*16*2 = 1.4 Mbit/s) sur réseau GSM (13.2 kbit/s) pour reconnaissance automatique de morceaux

3.5 Codage sans perte

<u>Rappel de théorie de l'information :</u>

On suppose qu'un signal est la suite de N réalisations d'une variable aléatoire discrète X, appelée source d'information simple (sans mémoire).

X prend ses valeurs dans un alphabet A = {x_0,x_1,x_2,…, x_{s-1}) de S symboles et suit une certaine loi de probabilité imposant :

$$\{p_i, p_x(x_i) = Prob(X = x_i), \forall i = 0..S-1\}.$$

Exemples :

Prenons un texte « Bonjour », 7 réalisations de X dans A = alphabet latin (avec C=128)

Codage sans perte ou entropie d'une source X :

Le codage d'une source X est une application (bijective) qui, à chaque symbole x_i pris par X dans l'alphabet A, associe un mot de code binaire ci de longueur l_i :

$$\begin{cases} A \longmapsto C \\ x_i \to c_i \end{cases}$$

Codage à longueur fixe : le BCD[17]

Chaque symbole xi est codé (indépendamment de la loi de probabilité de X) par la représentation $b_{J-1}…b_0$ de son indice i en base 2 sur L = [$\log_2$(s)] bits avec : $i = \sum_{j=0}^{L-1} bj \quad 2^j$

Exemple : Prenons un alphabet à S = 8 symboles

x_i	x_0	x_1	x_2	x_3	x_4	x_5	x_6	x_7
$b_2 b_1 b_0$	000	001	010	011	100	101	110	111

Ce qui nous donne :

Code 1 :	x_0	x_1	x_2	x_3
	00	01	10	11

Code 2 :	x_0	x_1	x_2	x_3
	0	10	110	111

Example (X : Loi uniforme)

Message	x_0	x_2	x_1	x_2	x_3	...
Message codé avec code 1	00	10	01	10	11	...
Message codé avec code 2	0	110	10	110	111	...

[17] Binary coded decimal

Code 1 :	x_0	x_1	x_2	x_3
	00	01	10	11

Code 2 :	x_0	x_1	x_2	x_3
	0	10	110	111

Example (X : Loi non uniforme avec $p_0 = \frac{3}{4}$ et $p_{1,2,3} = \frac{1}{12}$)

Message	x_0	x_2	x_0	x_0	x_3	...
Message codé avec code 1	00	10	00	00	11	...
Message codé avec code 2	0	110	0	0	111	...

Plus un symbole est probable, plus il y a d'intérêt à le coder avec peu de bits ⇒ il apporte peu d'**information**

CODAGE STATIQUE :

Codage Huffman :

Principe

- ⇨ Un élément est représenté par une séquence de bits de longueur proportionnel à la probabilité d'apparition → code à longueur variable
- ⇨ Optimal pour les codes à nombre de bits entier
- ⇨ Aucun élément du code ne peut être le préfixe d'un autre élément (code instantanée)
- ⇨ Méthode basée sur la construction d'un arbre basé sur les probabilités d'apparition d'un élément.
  - ⇨ Le codage de Huffman produit le code dont la longueur se rapproche le plus de l'entropie.

Prenons un exemple :

Un cas ou les probabilités suivantes sont associées à des éléments :

a1	a2	a3	a4	a5	a6	a7
0.38	0.24	0.1	0.1	0.1	0.05	0.03

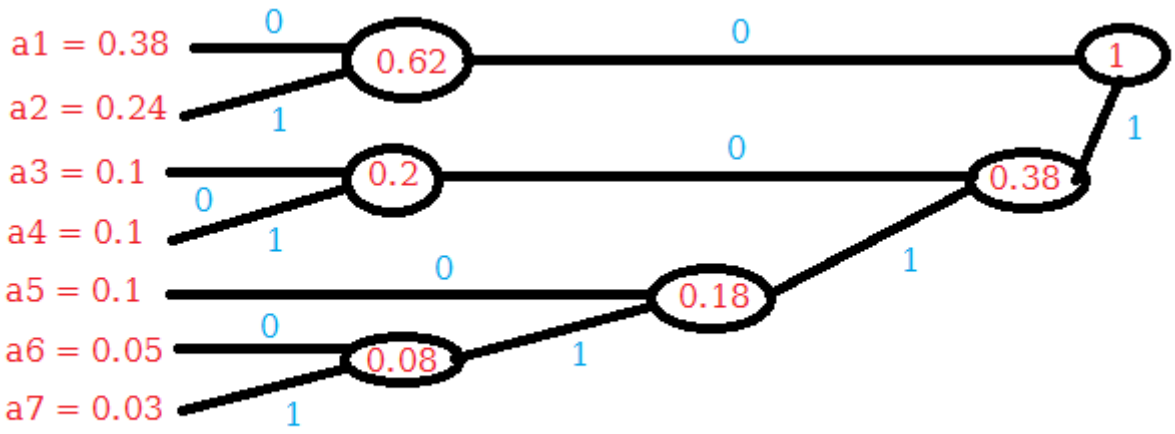
On commence par calculer H, la moyenne par symbole :

$H = -\Sigma\, pi * \log_2(pi)$:

$H = -[(0.38*\log_2(0.38)) + (0.24*\log_2(0.24)) + 3*(0.1*\log_2(0.1)) + (0.05*\log_2(0.05)) + (0.03*\log_2(0.03))]$

$H = 2.39$ *sh/s* (Shannon/ seconde)

Maintenant nous allons passer à la détermination du code de chaque symbole.



Bon, comme ça, on dirait que c'est compliqué, mais ce n'est pas le cas. En fait, on doit additionner les probabilités pour arriver à 1. A chaque fois qu'une branche rejoint l'autre, on lui associe le nombre **1**, et quand une branche reste au même niveau, on lui associe le nombre **0**.

Ensuite, pour la lecture du code, on part du résultat total de la somme (notre 1 final), et on découle jusqu'à arriver vers la probabilité de départ. Exemple : Ici le code du symbole de probabilité a1 sera égal à 00, car il y a deux zéros entre le 1 final et la probabilité a1. Pour a4, ça sera 101, pour a5, 110; etc …

Voici le tableau final trouvé :

Si	**Pi**	**Code**	**Li**
a1	0.38	00	2
a2	0.24	01	2
a3	0.1	100	3
a4	0.1	101	3
a5	0.1	110	3
a6	0.05	1110	4
a7	0.03	1111	4

Petite explication :

- **Si** est le nom de la probabilité associée au symbole transmis
- **Pi** la probabilité d'apparition du symbole associée
- **Code** le code binaire qui représente le symbole
- **Li** la longueur du code (le nombre de chiffres du code)

Bon, c'est presque fini, maintenant on arrive au calcul de l'efficacité de compression du message, qui se traduit par :

$Ec = H / \Sigma(P_i * L_i)$

Dans notre cas, Ec donne donc :

$Ec = 2.39 / (0.38*2 + 0.24*2 + 3*(0.1*3) + 0.05*4 + 0.03*4)$ <=> $Ec = 2.39 / 2.46 = 97.6\ \%$

Notre efficacité sera donc de 97.6 %.

Exemple de symbole :

On cherche à minimiser la longueur moyenne du symbole. Puisque les probabilités P = {P0, P1, ..., PN-1} sont fixées par la source une fois pour toute, on va affecter aux symboles les plus probables les mots-codes les plus courts (voir : on essaye d'égaliser les pi.log2(pi) ; pour avoir L = H, on doit obtenir li = -log2(pi)). C'est le principe de l'alphabet de Morse. On travaille pour l'instant sur chaque symbole pris indépendamment.
La méthode de Shannon-Fano est l'application directe de la synthèse de codes à décodage instantané.

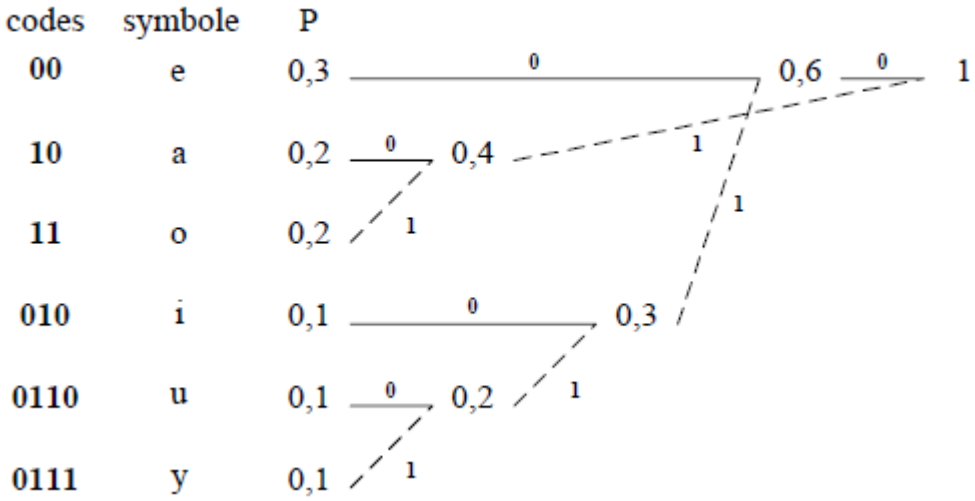
1. On établit la liste des symboles à coder et leur probabilité d'apparition.
2. On les classe par ordre décroissant (par pure commodité).
3. On les sépare en deux sous-ensembles ayant approximativement la même probabilité.
4. On affecte le bit 0 aux symboles composant le premier sous-ensemble et le bit 1 aux symboles composant le deuxième sous-ensemble.
5. On répète les opérations 3 et 4 jusqu'à l'obtention de sous-ensembles ne contenant qu'un seul symbole.

symbole	e	a	o	i	u	y
probabilité	0,3	0,2	0,2	0,1	0,1	0,1
étape 1	**0** (*0,5*)		**1** (*0,5*)			
étape 2	**0**	**1**	**0** (*0,3*)		**1** (*0,2*)	
étape 3			**0**	**1**	**0**	**1**
codes	**00**	**01**	**100**	**101**	**110**	**111**

Résultats : l'entropie H de la source est égale à 2,45 sh/sym. Avec un code de longueur fixe 3 bits, on a L= 3 bits/sym. Avec le code de Shannon-Fano, on obtient L= 2,5 bits/sym ce qui est proche de l'entropie.

La méthode de Huffman est assez similaire à la méthode précédente.

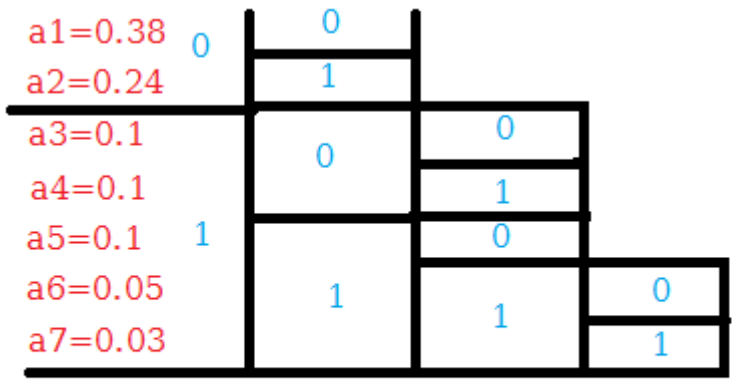
1. On établit la liste des symboles à coder et leur probabilité d'apparition.
2. On les classe par ordre décroissant (par pure commodité).
3. On remplace les deux symboles ayant la probabilité la plus faible par un nouveau symbole dont la probabilité est la somme des deux probabilités précédentes.
4. On répète les opérations 2 et 3 jusqu'à l'obtention d'un symbole de probabilité égale à 1.
5. On forme le code en parcourant l'arbre ainsi créé en allant de la racine (symbole de probabilité égale à 1) vers les feuilles (symboles de départ). On attribue 0 aux branches horizontales et 1 aux branches obliques.



Résultats : l'entropie H de la source est égale à 2,45 sh/sym. Avec un code de longueur fixe 3 bits, on a L= 3 bits/sym. Avec le code de Huffman, on obtient L= 2,5 bits/sym, ce qui est proche de l'entropie.

Codage Shannon –Fano :

Le plus dur est fait : p Le codage de Shannon-Fano ressemble énormément à Huffman, la seule différence réside dans le fait qu'on choisi arbitrairement les codes associés aux probabilités des symboles. Pour être plus clair, on va à chaque fois découper en deux parties les plus égales, et attribuer à chaque partie un **0** et un **1**. Le schéma ci-dessous représente l'arborescence des codes de chaque probabilité.



On choisit arbitrairement deux parties, pour qu'elles aient à chaque fois la valeur la plus approchante l'une de l'autre, et tant que les parties sont divisibles, on continue ainsi. Dans le tableau, on prend ici **a1** et **a2**, (dont la somme fait 0.62) et tout le reste, (dont la somme fait 0.38). On les sépare en deux, on met le zéro dans la partie du dessus, et les 1 dans la partie du bas. Puis on recommence pour chaque morceau de probabilité, on sépare le groupe du dessus en 2, et celui du dessous entre **a3** et **a4** pour le chiffre 0, et **a5,a6** et **a7** pour 1.

Lorsqu'on a séparé tout les groupes, il suffit de tracer des lignes entre chaque probabilité (de a1 à a7 ici), et de lire en partant de la droite, la ligne correspondante. Pour **a7**, on lit 1111.

On remarquera qu'ici on retrouve le même résultat qu'avec la méthode de Huffman, mais ceci, parce que nous avons déterminé à chaque fois les deux blocs étant les plus égaux l'un envers l'autre. Si l'évaluation est mauvaise, on trouvera forcément un résultat inférieur à le méthode de Huffman, qui est donc plus performante.

J'espère que cette petite introduction pourra aider certains à comprendre. Je vous invite à me laisser un commentaire, si vous voyez une erreur, ou tout simplement pour des questions, ou autre.

Codage arithmétique :

C'est un code statistique comme Huffman, mais de conception plus récente. Le principe général a été énoncé par Elias en 1963. Il donne, en général, des résultats supérieurs d'environ 10 % au codage de Huffman au prix d'une complexité algorithmique beaucoup plus élevée.
Les seuls algorithmes utilisables pour implémenter le codage arithmétique, les algorithmes du Skew-Soder et du Q-Coder, ont été développés et brevetés par IBM en 1988. Leur utilisation doit donc faire l'objet de paiement de royalties.

Principe

- ⇨ Un ensemble de valeur est codé par un nombre flottant ∈ [0,1[.

⇨ L'intervalle [0,1[est découpé en sous-intervalles associés à chaque éléments a_i de taille proportionnelle à la probabilité d'apparition $p(a_i)$ de l'élément.

Algorithme

1. Borne inférieure $I_0 = 0$; largeur $D = 1$
2. Prendre le nouvel élément a_i gf
3. $I_0 = I_0 + D \times I_0(a_i)$; $D = D \times D(a_i)$
4. Répéter les étapes 2 et 3 pour tous les éléments à coder
5. Code = nombre I.

Exemple 1 :

Le codage de HUFFMAN a un défaut: si on considère un code possédant un symbole apparaissant avec une probabilité `0.9`, le codage de HUFFMAN se limitera à 1 bit par symbole sans tenir compte de la quantité d'information contenu dans celui-ci. Le codage arithmétique permet de s'affranchir de cette limite et de coder les symboles sur un nombre non-entier de bits. Ceci parait à première vue surprenant; considérons donc un exemple pratique: la chaîne "compressée"

Symbole	Occurrences	Probabilité
c	1	$0,1$
e	3	$0,3$
m	1	$0,1$
o	1	$0,1$
p	1	$0,1$
r	1	$0,1$
s	2	$0,2$

NOTA BENE: le nombre de caractères de l'exemple ici est 10, ce qui simplifie grandement les manipulations numériques de l'exemple puisque les nombres auront un nombre fini de décimales! Bien sûr, ce procédé s'applique à des chaînes de longueur quelconque car il repose sur des intervalles et non sur les représentations exactes de nombres.

On décide maintenant d'associer à chaque symbole 'd' un *domaine* dans l'espace des probabilités, le but étant d'associer un symbole à un domaine. La table d'associations sera transmise avec le message encodé (comme dans le cas de HUFFMAN statique):

Symbole	Probabilité	Domaine
c	$0,1$	$0,0 \leq \mathrm{d} < 0,1$

e	$0,3$	$0,1 \leq \mathtt{d} < 0,4$
o	$0,1$	$0,4 \leq \mathtt{d} < 0,5$
m	$0,1$	$0,5 \leq \mathtt{d} < 0,6$
p	$0,1$	$0,6 \leq \mathtt{d} < 0,7$
r	$0,1$	$0,7 \leq \mathtt{d} < 0,8$
s	$0,2$	$0,8 \leq \mathtt{d} < 1,0$

Par exemple, le message original ("compresse") est encodé comme suit:

Symbole	Borne Inférieure	Borne Supérieure
	0	1
c	0, 0	0, 1
o	0, 04	0, 05
m	0, 045	0, 046
p	0, 0456	0, 0457
r	0, 04567	0, 04568
e	0, 045671	0, 045674
s	0, 0456718	0, 0456720
s	0, 0456726	0, 0456728
e	0, 04567262	0, 04567268
e	0, 045672626	0, 045672644

Ainsi 0,045672626 est la représentation arithmétique du message "compressee". En pratique, n'importe quel nombre compris entre 0,045672626 et 0,045672644 représente le bon message.

Autres exemples :

Symboles	Probabilité	Partition Initiale
a	0,2	[0 - 0,2 [
e	0,3	[0,2 - 0,5 [
i	0,1	[0,5 - 0,6 [
o	0,2	[0,6 - 0,8 [
u	0,1	[0,8 - 0,9 [
!	0,1	[0,9 - 1 [

Partition initiale de l'intervalle [0 -1[pour l'alphabet {a,e,i,o,u,!}

Initialement [0 - 1 [
e → [0,2 - 0,5 [
e a → [0,2 - 0,26 [
e a i → [0,23 - 0,236 [
e a i i → [0,233 - 0,2336 [
e a i i ! → [0,23354 - 0,2336 [

Construction du code du message eaii!
⇒ tout réel ∈ [0,23354 - 0,2336[

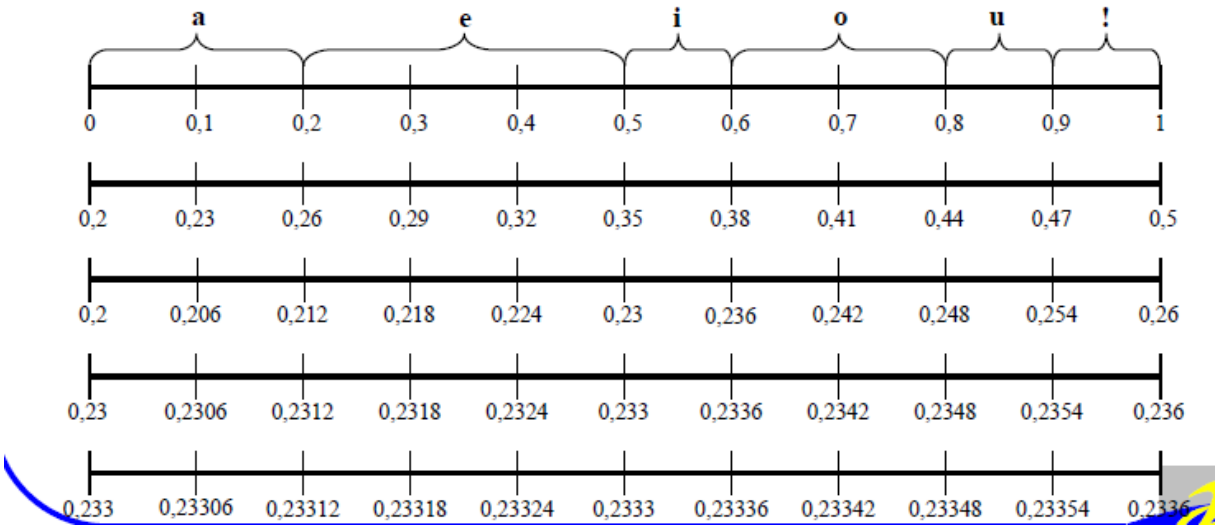
Le Codage Arithmétique

Codage de eaii!

⇒ tout réel ∈ [0,23354 - 0,2336[

Symboles	Probabilité	Partition Initiale
a	0,2	[0 - 0,2 [
e	0,3	[0,2 - 0,5 [
i	0,1	[0,5 - 0,6 [
o	0,2	[0,6 - 0,8 [
u	0,1	[0,8 - 0,9 [
!	0,1	[0,9 - 1 [

Initialement [0 - 1
e → [0,2 - 0,5
e a → [0,2 - 0,26
e a i → [0,23 - 0,236
e a i i → [0,233 - 0,2336
e a i i ! → [0,23354 - 0,2336



Exemple 2 :

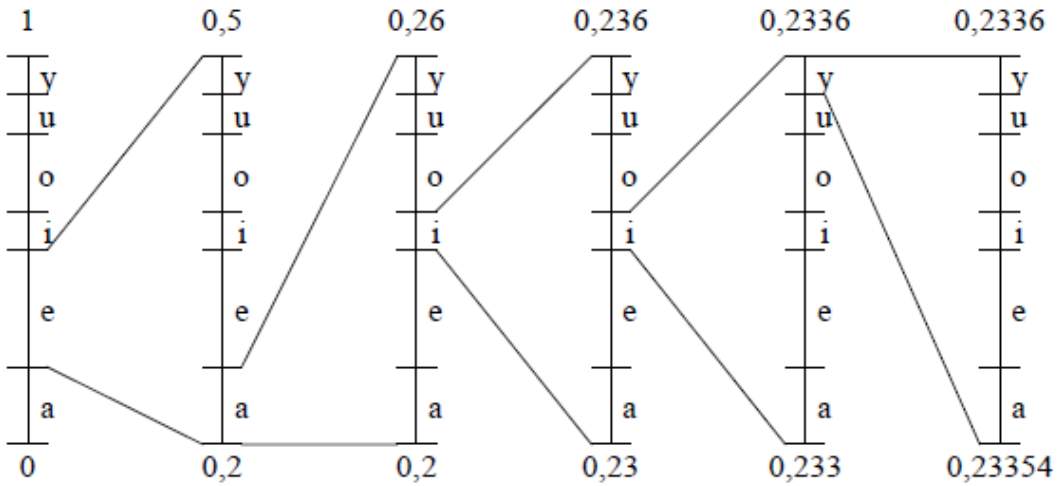
Détermination des probabilités P_i de chaque symbole S_i et calcul de l' intervalle $[a_i, b_i[$.
Symbole Probabilité Intervalle

Symbole	Probabilité	Intervalle
a	0,2	[0, 0.2[
e	0,3	[0.2, 0.5[
i	0,1	[0.5, 0.6[
o	0,2	[0.6, 0.8[
u	0,1	[0.8, 0.9[
y	0,1	[0.9, 1[

Codage du message 《 eaiiy 》 par exemple.

caractère	Intervalle	s	Max	Min
-	-	-	1	0
e	[0.2, 0.5[	1	0.5	0.2
a	[0, 0.2[	0.3	0.26	0.2
i	[0.5, 0.6[	0.06	0.236	0.23
i	[0.5, 0.6[	0.006	0.2336	0.233
y	[0.9, 1[	0.0006	0.2336	0.23354

Le message compressé est représenté par les bornes Min et Max finalement obtenues. Il n'est pas obligatoire de transmettre ces deux bornes, une seule suffit ou bien n'importe quelle valeur entre ces deux bornes. On peut aussi voir le processus de codage de la manière suivante :



Principe du décodage du message. L'algorithme de décodage utilise le principe suivant :

1. La variable décimale N représente le message codé (borne Min dans cet exemple).
2. Décodage du message selon la procédure suivante :

Répéter {
    trouver le ième symbole tel que $N \in [a_i, b_i[$;
    le sauver;
    $s = b_i - a_i$;
    $N = N - a_i$;
    $N = N / s$;
} tant $N \neq 0$;

Exemple : reprenons l'exemple2 précédent

N	i	Caractère décodé	intervalle	s
0.23354 0.03354 (N=N-a_i) 0.1118 (N=N/s)	2	e	[0.2, 0.5[	0.3
0.1118 0,1118 (N=N-a_i) 0.559 (N=N/s)	1	a	[0, 0.2[	0.2
0.559 0,059 (N=N-a_i) 0.59 (N=N/s)	3	i	[0.5, 0.6[	0.1
0.59 0,09 (N=N-a_i) 0.9 (N=N/s)	3	i	[0.5, 0.6[	0.1
0.9 0 (N=N-a_i) 0 (N=N/s)	6	y	[0.9, 1[	0.1

Le codage arithmétique incrémental est une évolution du principe précédent. On n'attend pas la fin du codage pour transmettre le message comprimé. En effet, après quelques itérations, les premières décimales ne sont plus modifiées.

caractère	Max	Min	décimale transmise
-	1	0	-
e	0.5	0.2	0
a	0.26	0.2	2
i	0.236	0.23	3
i	0.2336	0.233	3
y	0.2336	0.23354	54

Il est possible d'effectuer les calculs avec un nombre de décimales limité et en utilisant des
entiers pour coder les bornes. On obtient alors le codage arithmétique binaire. Voyons son
principe sur un exemple :

· Les probabilités des symboles sont codées en puissance de 2 : avec 4 bits, on atteint une
précision de $2^{-4} = 0,0625$.

· On utilise une source à 4 symboles :

symbole	Probabilité P	Intervalle	P codée en binaire	Intervalle codé en binaire
S_1	1/2	[0, 0.5[	0.1	[0, 0.1[
S_2	1/4	[0.5, 0.75[	0.01	[0.1, 0.11[
S_3	1/8	[0.75, 0.875[	0.001	[0.11, 0.111[
S_4	1/8	[0.875, 1[	0.001	[0.111, 1[

On veut coder la chaîne « S_1 S_1 S_2 S_3 ».

symbole	Intervalle	Intervalle codé en binaire
S_1	[0, 1/2[	[0, 0.1[
S_1	[0, 0.1/4[	[0, 0.01[
S_2	[1/8, 3/16[	[0.001, 0.0011[
S_3	[11/64, 23/128[	[0.0010110, 0.0010111[

Le codage incrémental est possible (attention à la propagation de la retenue), mais le nombre de bits augmente avec la précision sur l'intervalle. Il faut renormaliser les intervalles. C'est ce
que font les algorithmes du Skew-Soder et du Q-Coder (avec en plus un codage adaptatif).

Codage de Lempel-Ziv :

Principe

- ⇨ **Famille d'algorithme conçu par Lempel-Ziv (1977) et finalisé par Welch (1984)**
- ⇨ **Basé sur la répétition de chaines de données dans une séquence**
- ⇨ **Utilise un dictionnaire construit dynamiquement**
- ⇨ **Utilisé par le format gzip**

Fonctionnement

⇨ L'ensemble des symboles (motifs unitaires) constituant la séquence complète font implicitement partie du dictionnaire

⇨ Les adresses supérieures sont affectées aux séquences originales (motifs complexes) rencontrées dans la séquence complète.

1. Lecture du premier symbole + mémorisation dans le tampon
2. Lecture du deuxième symbole placé dans le tampon
3. Si la chaîne présente dans le tampon n'a pas déjà été rencontrée elle est ajoutée au dictionnaire et affectée d'un nouveau code.
4. Le code correspondant au premier symbole est alors émis à défaut d'une chaîne plus longue.
5. Si la chaîne est déjà présente dans le dictionnaire rien n'est émis dans le fichier de sortie et le caractère suivant est alors lu afin de rechercher une chaîne de caractère plus longue.

⇨ **Le processus complet est décrit par l'exemple suivant :**

Exemple : chaîne à encoder : DAD DADA DADD

étape	Caractère lu	Caractère émis	tampon	adresse	séquence
1	D		D	0…255	Caract. ASCII
2	A	D	DA	256	DA
3	D	A	AD	257	AD
4	_	D	D_	258	D_
5	D	_	_D	259	_D
6	A		DA		
7	D	256	DAD	260	DAD
8	A		DA		
9	_	256	DA_	261	DA_
10	D		_D		
11	A	259	_DA	262	_DA
12	D		AD		
13	D	257	ADD	263	ADD
14	Y	D	DY	264	DY
15	_	Y	Y_	265	Y_

Exemple :

01101110010011001101…

- Le premier bit de la chaîne 0 devient le premier mot de la table. 0 est appelé bit d'innovation.

- 1 est mis dans la table comme second mot, 1 est bit d'innovation.
- le bit 1 (déjà dans la table) est associé à son successeur 0, donnant 10 qui devient un nouveau mot de la table. 1 est racine et 0 bit d'innovation.
- Itération de la procédure jusqu'à la fin de la chaîne.
- Contenu de la table contient : 0, 1, 10, 11, 100, 1001, 10011, 01…
- Chaque symbole est codé avec des mots de longueur fixe dont le dernier bit décrit le bit d'innovation (0 ou 1) et les premiers bits l'adresse de la racine.

Base du codage ZIP (winzip sur PC), utilisé dans le format GIF.

Codage adapté aux images de faible entropie et donc pas adapté aux images médicales

CODAGE SPACTIAL :

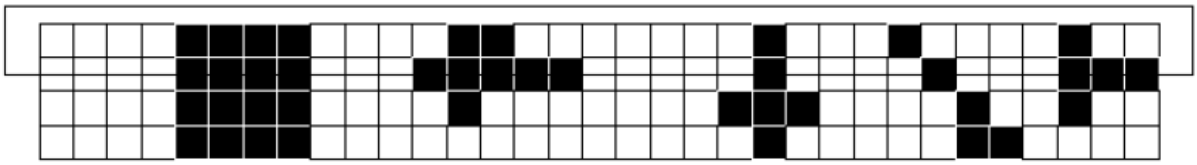
Codage par plage ou RLC[18]:

Principe

- ⇨ Méthode basée sur l'existence d'une répétition d'un même symbole
- ⇨ La valeur du symbole est alors codée ainsi que le nombre de répétition.
- ⇨ Bonne performances en présence d'un nombre réduit de valeurs possibles.

Utilisation

- ⇨ Images binaires (fax)
- ⇨ Dessins bitmap
- ⇨ Assez peu performante pour le codage d'image à niveau de gris



Code : 4,4,4,2,7,1,3,1,4,1,2

Exemple caractère :

C' est une méthode élémentaire permettant de prendre en compte la redondance entre symboles. Elle est efficace quand le message est composé de suites de symboles identiques.
Au lieu de coder indépendamment chaque symbole, on détermine des couples (nombre de symboles S consécutifs, S).

AABBBDDDEFF = 2A3B3D1E2F

Cette méthode est particulièrement efficace dans le cas d' une image composée de pixels noirs
(N) ou blancs (B) comme dans le cas du télécopieur. Avec trois bits, on peut par exemple coder :

[18] Run Length Coding

N	000
BN	001
BBN	010
BBBN	011
BBBBN	100
BBBBBN	101
BBBBBBN	110
BBBBBBBN	111

Si P est la probabilité d'un pixel noir, le taux de compression est inférieur ou égal à 1-3.P. Par exemple, si P = 0.1, on ne pourra pas espérer dépasser 70 % de taux de compression. Il faut pour que ce codage soit efficace que P soit petit (P < 1/3), sinon, il faut coder les blancs. Si
$P_{blanc} \approx P_{noir}$, alors ce codage augmente la redondance au lieu de la diminuer.

3.6 Compression avec pertes

Codage prédictif :

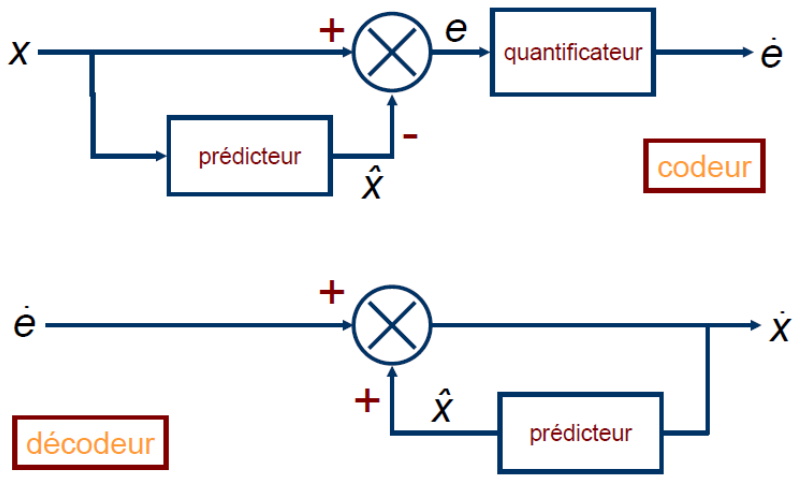
Principes

- ⇨ Prédiction de la valeur des pixels
- ⇨ Codage de la différence entre la valeur réelle et la valeur prédite du pixel
- ⇨ La différence est généralement quantifiée (de manière nonuniforme)
- ⇨ La prédiction peut être menée
- ⇨ Au sein d'une image
- ⇨ Entre deux images d'une même séquence

Exemple : compensation de mouvement (MPEG1/2)

Méthode DCPM (**D**ifferential **P**ulse **C**ode **M**odulation)

- ⇨ Codage MIDC en téléphonie (Modulation par Impulsion Différentielle Codée)
- ⇨ Prédiction d'un pixel par combinaison linéaire de ses voisins déjà transmis



Appliquer à l'image :

Principe : au moyen de l'historique de l'image, on va prédire une valeur des pixels. On envoie ensuite uniquement la différence entre la valeur prédite et la valeur réelle.
Pour permettre une compression cette différence est généralement quantifiée (de manière non uniforme).
On peut réaliser la prédiction

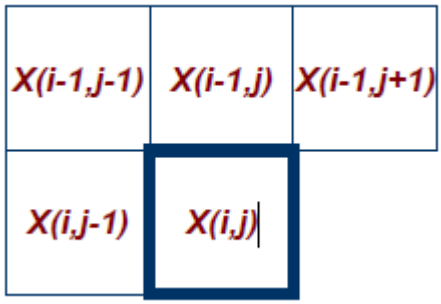
- ⇨ au sein d'une image, en utilisant les valeurs des pixels voisins pour prédire un pixel (prédiction intra)
- ⇨ entre images d'une séquence, en utilisant la corrélation forte entre images successives d'une vidéo (prédiction inter)

Cette dernière méthode est fortement utilisée par les standards de compression de séquence vidéo (MPEG1/2/4, etc).

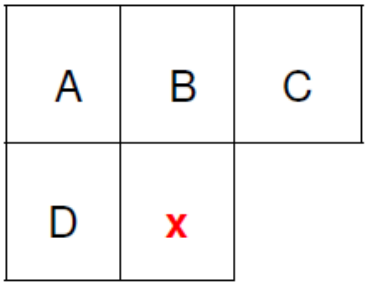
Prédicteurs

- ⇨ Prédiction d'un pixel
- ⇨ par combinaison linéaire de ses voisins déjà transmis
- ⇨ Généralement 4 voisins
- ⇨ Exemple

$$\hat{X}(i,j) = \frac{1}{4}[x(i-1,j-1) + x(i,j-1) + x(i+1,j-1) + x(i-1,j)]$$



Exemple 2 :



Pixels utilisés pour prédire **x**

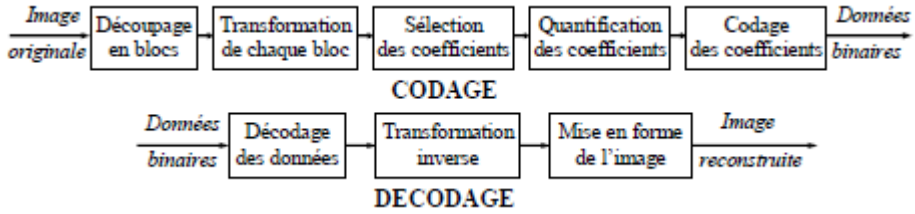
Comme les images sont des sources markoviennes, prédire un pixel par un de ses voisins améliore son entropie.

Composante	$H(x)$	$H(x\|D)$	$H(x\|B)$
Y	7.34	4.66	4.85
Cr	5.57	3.76	2.96
Cb	5.24	3.75	2.93

On prédit la valeur d'un pixel comme une combinaison linéaire des valeurs des pixels voisins déjà connus. (balayage par ligne).
Il est théoriquement possible de déterminer un ensemble optimal de coefficients pour une image donnée (au moyen des coefficients d'autocorrélation des pixels).
Mais calculer et envoyer ces coefficients est couteux, et on préfère généralement des coefficients constants.

Transformations orthogonales :

Cette transformation suit le schéma suivant :



Généralité sur les algorithmes de codage par transformation

Transformation optimale vis-à-vis de la théorie de l'information (minimisation de l'EQM)

- ***Transformation de KARHUNEN-LOEVE (TKL).***
  Constituée des vecteurs propres normalisés de la matrice de covariance du bloc.
- Transformation complexe non utilisée en pratique (pas d'algorithme rapide de calcul).

⇒ Utilisation de transformations ayant des vecteurs de base fixes :

- Walsh-Hadamard
- Fourier discrète
- Cosinus discrète

La transformation de WALSH-HADAMARD (TWH)

L'élément d'indice (i, j) de la matrice de transformation N*N est de la forme :

$$H_{N,i,j} = \left(\frac{1}{\sqrt{2}}\right)^N (-1)^{\sum_{k=0}^{n-1} i(k)j(k)}$$

Où i(k) et j(k) sont les k^{ème} bits dans la représentation binaire de i et j.

Si N puissance de 2 : matrice de transformation générée par n produits successifs de KRONECKER de la matrice de base B d'ordre 2 :

$B = \frac{1}{\sqrt{2}}\begin{vmatrix} 1 & 1 \\ 1 & -1 \end{vmatrix}$ Matrice de transformation ne comportant que des 0 et des 1.

→ **Méthode simple et réversible**

La transformation discrète de FOURIER (TFD)

$X(k,1) = \sum_{i=0}^{M-1} \sum_{j=0}^{N-1} x(i,,j) W_M^{ki} W_N^{lj}$ avec $W_N = e^{-j\frac{2\pi}{N}}$

La transformation cosinus discrète (TCD)

$$X(k,l) = \frac{2}{\sqrt{MN}} c(k)c(l) \sum_{i=0}^{M-1} \sum_{j=0}^{N-1} x(i,j) \cos\left(\frac{(2i+)k\pi}{2M}\right) \cos\left(\frac{(2j+)l\pi}{2N}\right)$$

Avec

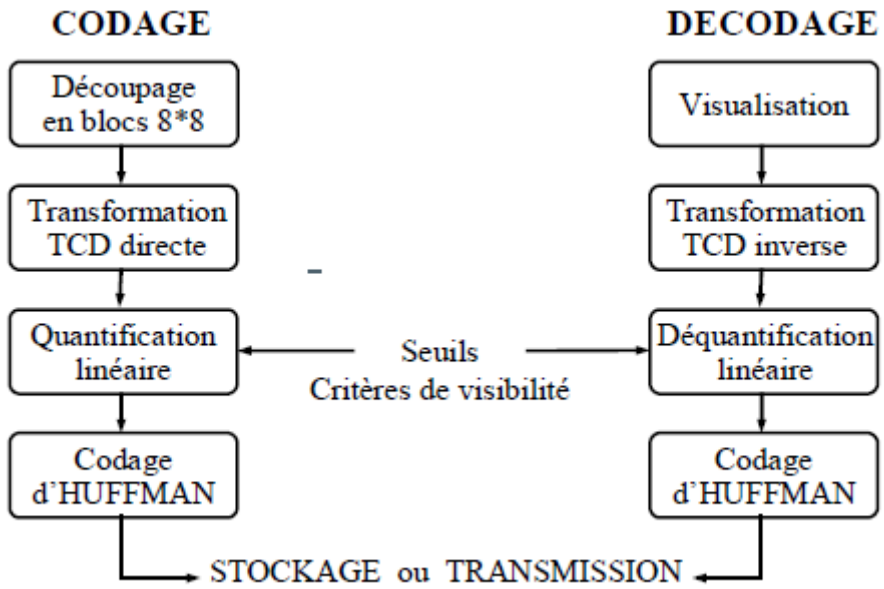
$c(\lambda) = \frac{1}{\sqrt{2}}$ $pour\ \lambda = 0$

$$c(\lambda) = 1 \quad sinon$$

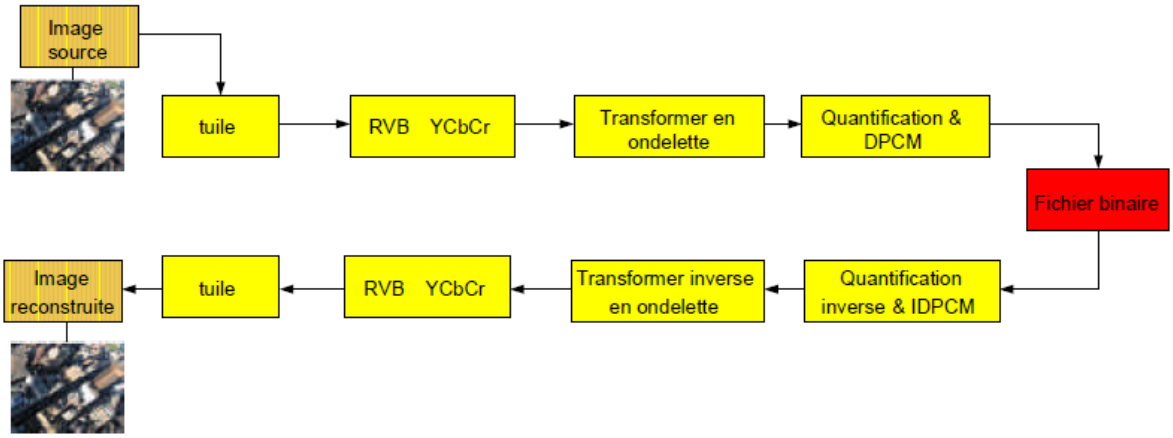
Transformation la plus proche, de par ses résultats et erreurs entre l'image reconstruite et l'image originale, de la transformation de KARHUNEN-LOEVE.

3.7 Applications :

⇨ **JPEG : méthode de codage par TCD adaptative**



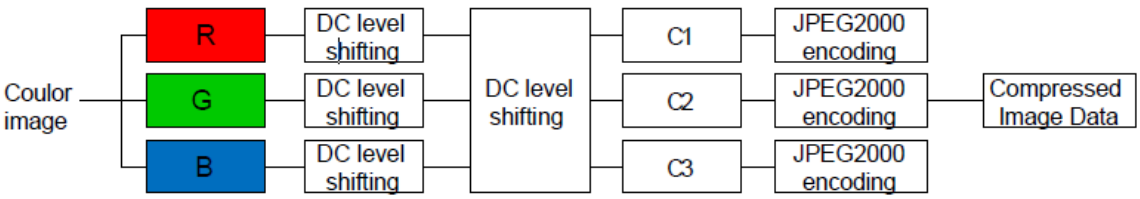
⇨ **JPEG2000 :**



Le tuilage :
Le tuilage permet de sélectionner, de décomposer l'image source en tuile d'une taille prédéfini. Chaque tuile est codée séparément.

Transformation intercomposante :

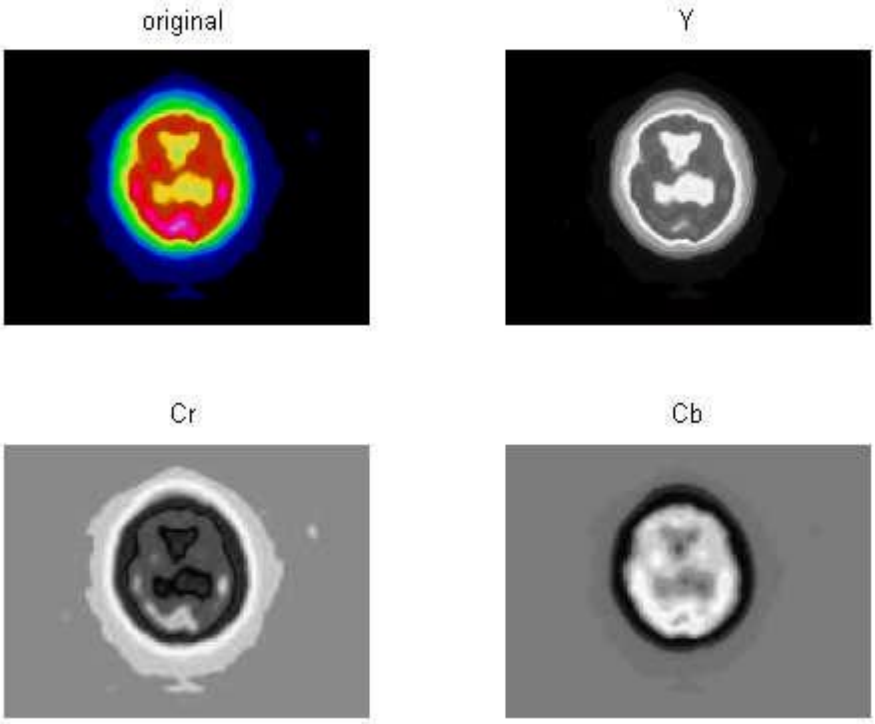
Cette transformation permet de retrouver les différentes composantes couleurs d'une image. Le schéma de principe est de prendre une image, puis de prendre une tuile courante est de la transformé en YCrCb :



On transforme une tuile codé en RGB en nouvelle tuile YCrCb. Selon une matrice de passage définie par la norme de JPEG2000 suivant :

$$\begin{pmatrix} Y \\ C_b \\ C_r \end{pmatrix} = \begin{pmatrix} 0{,}299 & 0{,}587 & 0{,}114 \\ -0{,}16875 & -0{,}33126 & 0{,}5 \\ 0{,}5 & -0{,}41869 & -0{,}08131 \end{pmatrix} \begin{pmatrix} R \\ G \\ B \end{pmatrix}$$

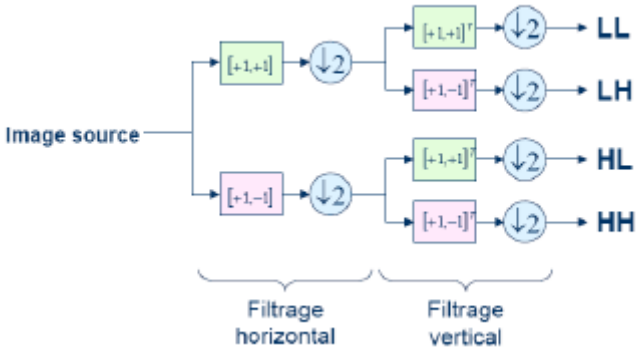
Résultat :



Nous pouvons observer les différentes intensités de contraste pour Y, Cr et Cb. Je conclus que si nous avons une image en noir et blanc, cette étape de sert à rien pour JPEG.

Transformation en ondelette :

La norme de compression d'images fixes JPEG2000 est basée sur la transformée en ondelettes discrète (DWT : *Discrete Wavelet Transform*). La transformée en ondelettes permet l'analyse temps-fréquence d'un signal. En traitement numérique cette transformer est équivalente à un filtre passe-bande. On applique la transformée en ondelettes sur chaque composantes des tuiles.
Nous avons comme principe suivant : On prend une image source ce qui signifie ici prend l'image de la tuile courante.
Puis on la passe dans différents filtres et de sous échantillonnage successive.
Ce qui implique que nous avons en sortie une sous-image correspondant au contenu d'une bande de fréquence particulière.

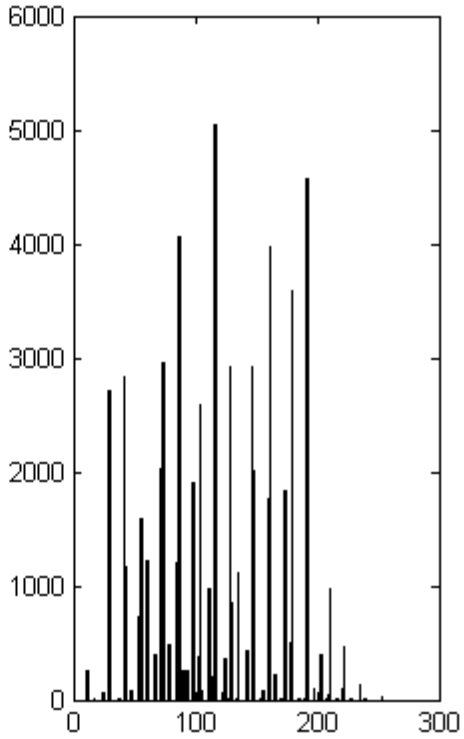


On peut voir que sur la figure précédent différente bande.
Je m'explique :

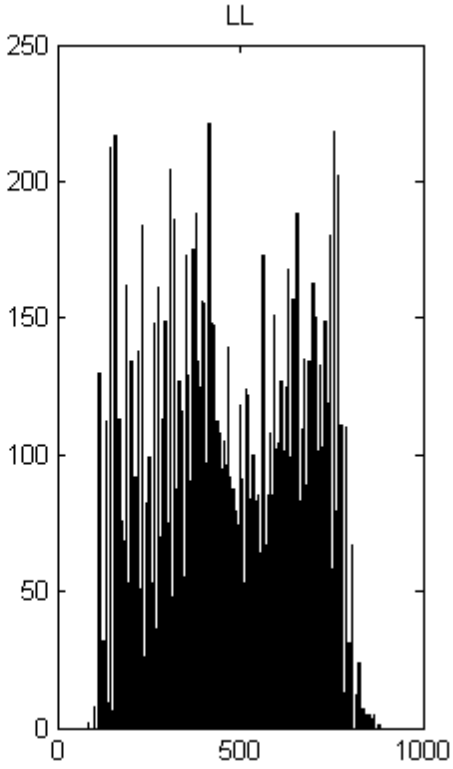
- ⇨ LL : image passe-bas→filtrage passe-bas horizontal et vertical ;
- ⇨ LH : image des contours verticaux → filtrage passe-haut vertical et passe-bas horizontal
- ⇨ HL : image des contours horizontal→ filtrage passe-bas vertical et passe-haut horizontal ;
- ⇨ HH : image des contours diagonaux → filtrage passe-haut vertical et passe-haut horizontal.

On note que LL est la sous-bande BF qui correspond à l'image approximante à la résolution inférieure r_{j-1}. Les trois autres sous-bandes représentent les détails perdus dans cette approximation à la même résolution r_{j-1} : des variations horizontales, verticales et diagonales.

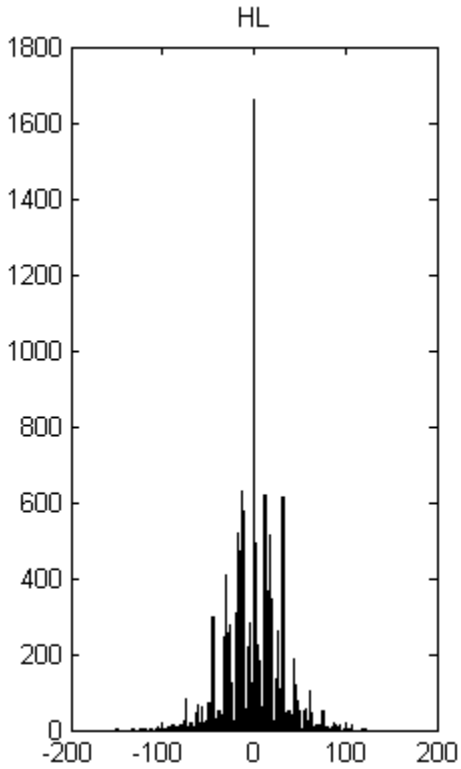
(1) Image source



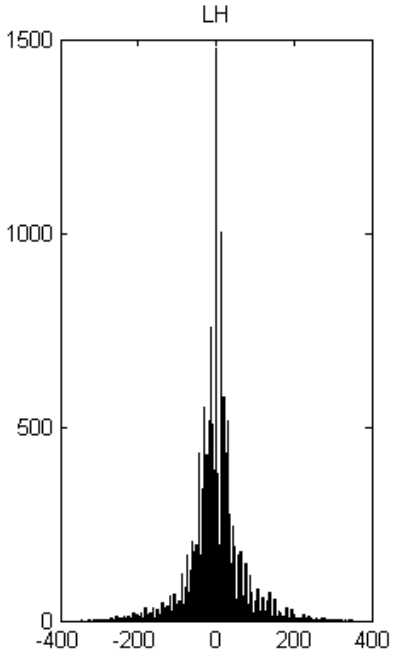
(2) image passe-bas, on observe que l'image est quantifier et l'histogramme étendu.



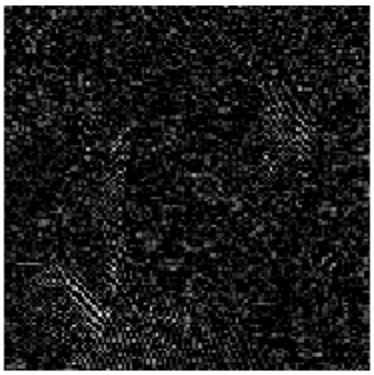
(3) image HL, on a perdu des informations, l'image a un contraste important.

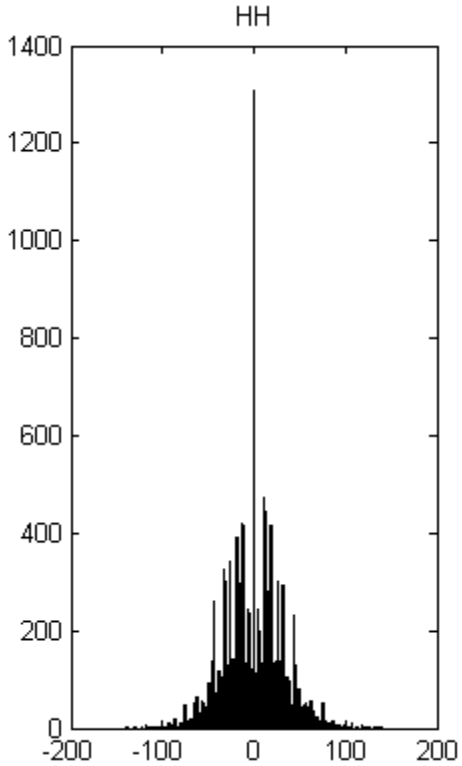


(4) image LH



(5) image HH





Quantification des composantes HF

Après transformée en ondelettes on dispose d'une image constituée d'une sous-image approximation, concentrant l'essentielle de l'énergie du signal de départ, et de douze sous-images détails. La quantification consiste alors à codée la dernière approximation de l'image ainsi que les quelques coefficients de détails situés au-dessus d'un seuil, les autres prenant la valeur de 0.

La quantification a pour but de réduire la précision des coefficients. A chaque sous-bande b issue de la transformée en ondelettes correspond un pas de quantification Δb.

On note deux formules pour la norme JPEG2000 :

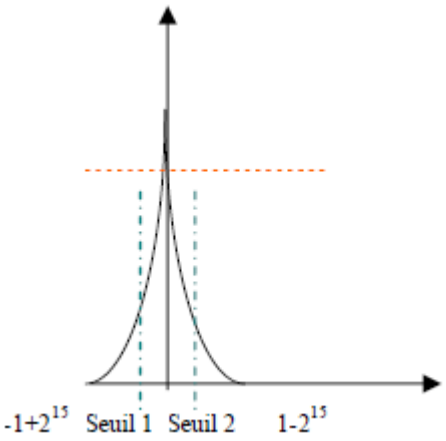
$$Q_b(x,y)sign(A_b(x,y)]\left[\frac{|A_b(x,y)|}{\Delta b}\right]$$

Avec Q_0 : coefficient quantifié, A_b : coefficient, Δb : pas de quantification

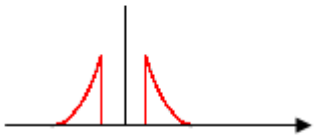
$$\Delta b = \left(1+\frac{\mu_b}{2^{11}}\right) * 2^{R_b-\varepsilon_b}$$

Avec Rb = dépend du nombre de bits utilisait pour représenter une image originale et sur le choix de « the wavelet transform ». On note que µb = 0 pour compression réversible au codé. Rb est sur 15 bits et εb = coefficient pourcentage sous 5 bits.

Il faut trouver le meilleur coefficient de pourcentage pour avoir la meilleure quantification.
On va rapidement expliquer le fonctionnement de la quantification.
On prend une courbe suivante :

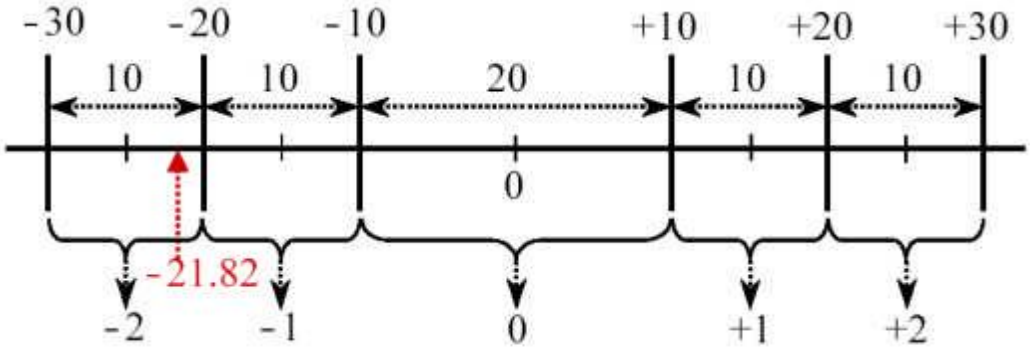


Le principe de la quantification est de trouvé que le seuil choisi doit être inférieur supérieur Qb. Alors Qb est à zéro.
Ce qui implique que la courbe précédente devient la suivante :



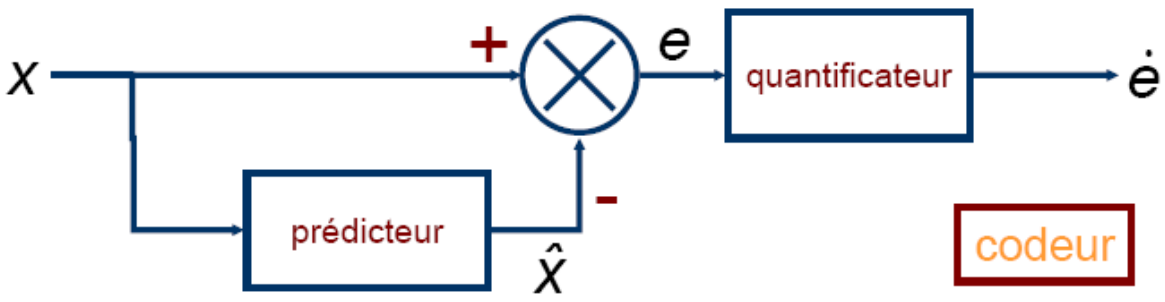
On peut voir que le principe est complexe et doit être plus précis pour permettre une meilleure quantification.
On finit par cet exemple classique :

On recherche la valeur 21,82. On doit faire tout le parcourt de la courbe jusqu'à dépasser le seuil choisi. En dessous du seuil, nous mettons tous les bits de la matrice en zéro.
Ce qui donne le schéma suivant :



Codage DPCM:

Ce codage est de la famille du codage prédictif. Le DPCM signifie Differentiel Pulse Code Modulation. Ce codage permet de faire la corrélation spatiale d'une image. Il prédit la valeur de l'amplitude du point courant par la seule connaissance de son voisinage déjà transmis.
La transmission de l'erreur de prédiction e(m,n) est la différence entre l'amplitude du pixel courant S(m,n) et une
prédiction s(m,n) de celle-ci.
En absence de quantification, le codage est réversible.
On a comme de fonctionnement suivant :

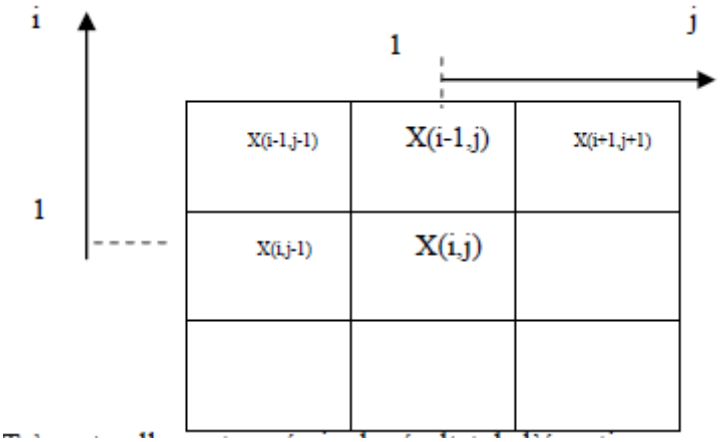


On a l'entrée une image source X de variable double.
X(chapeau) = une somme de valeur autour d'un ensemble pointer par une image choisi. Nous avons voir comment fonctionne ce type de codage sur une image pixel. On représenter l'image décomposer (FIG.14) et ce qui nous donnes la formule suivante :

X(chap) = (a +b+c+d)/4 (3)

a	b	c
d	X	

Je fixe un repère i, j et je choisi une matrice 4.



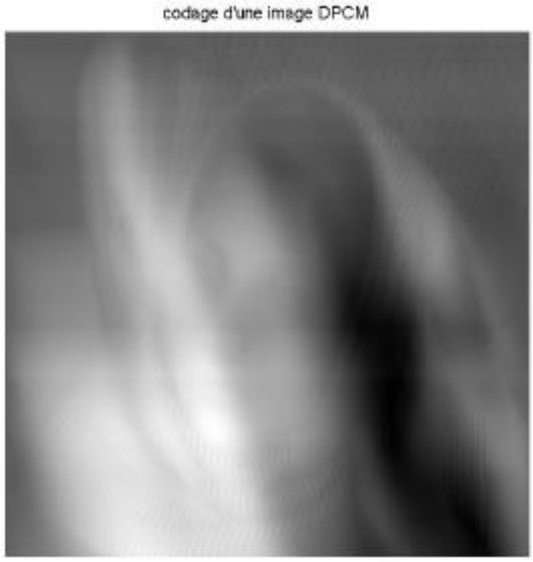
Très naturellement, on écrire le résultat de l'équation :

$$\hat{X} = \frac{1}{4}[X(i-1,j-1) + X(i,j-1) + X(i-1,j-1) + X(i-1,j)]$$

Si on généralise la formule :

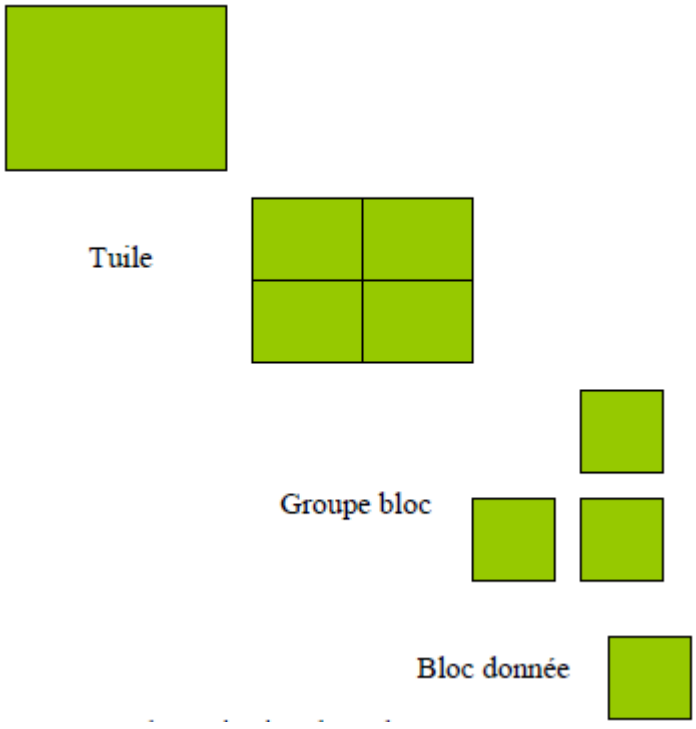
$$\hat{X} = c_1X(i-1,j-1) + c_2X(i,j-1) + c_3X(i-1,j-1) + c_4X(i-1,j)$$

L'erreur de quantification est notée :
E = Xx(image origine)-X(point)

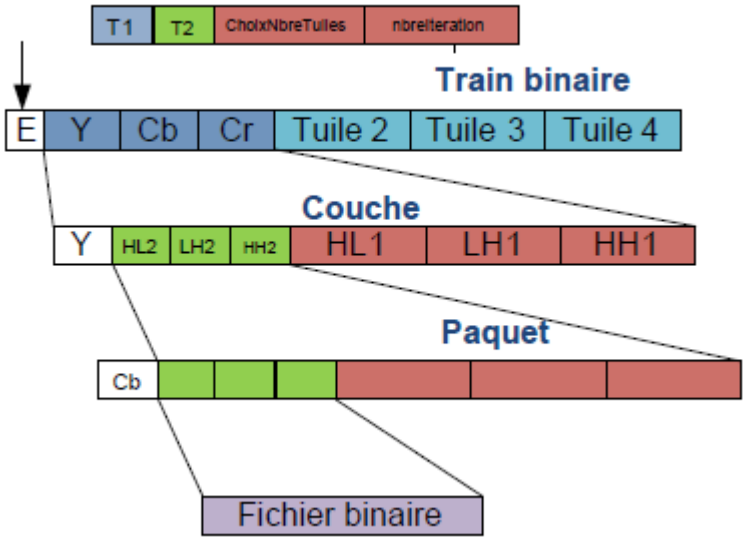


Le train binaire- codage:

JPEG2000 utilise le codage T2. Ce codage est structuré comme ceci.

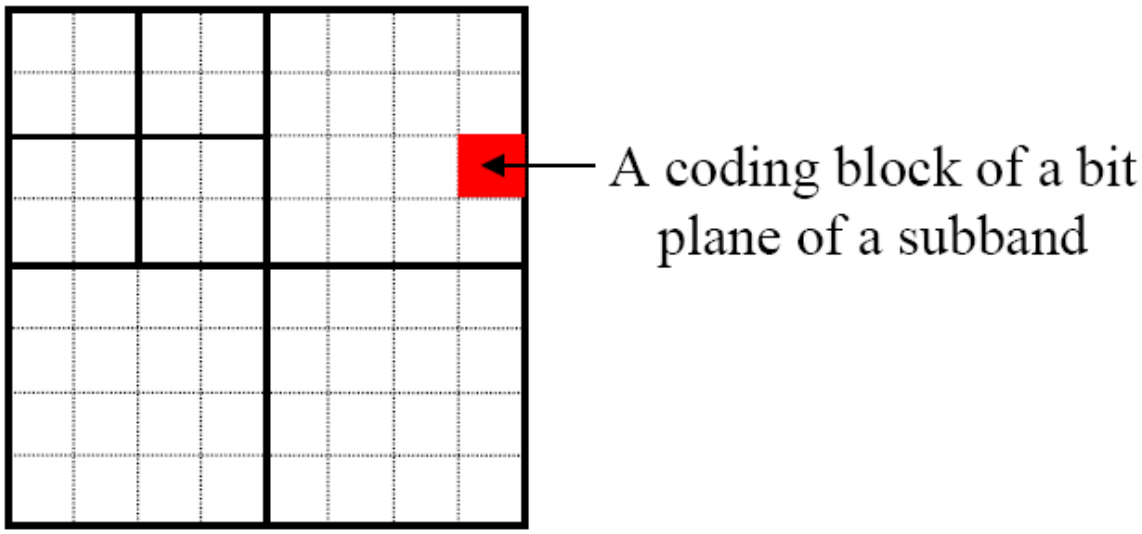


Le train binaire permet la création des paquets en unité de base pour la compression de JPEG2000, paquet associés à une Tuile, une composante, un niveau de résolution, un groupe de blocs de données et une couche, codage des entêtes des paquets.



Codage:

Le codage permet de finir un train d'information en un ensemble binaire.
Dans la norme JPEG2000, on appelle Entropy Coder.
Le codage permet ordonner les différents trains binaires des blocs donnés.



On utilise codage de HUFFMAN.
Le codage Huffman fait partie des méthodes de codage statistique.
On a comme principe de cette méthode :

-Classement des symboles par ordre de probabilité décroissante ;

-Regroupement des symboles de probabilités les plus faibles en un nouveau message de probabilité égale à la somme des probabilités des messages sources ;

-Reclassement des probabilités du nouvel ensemble de message par ordre décroissant ;

-Réitération de la procédure pour tous les messages à coder ;

-Attribution arbitraire des valeurs binaires 0 et 1 aux deux branches issues d'un même noeud ;

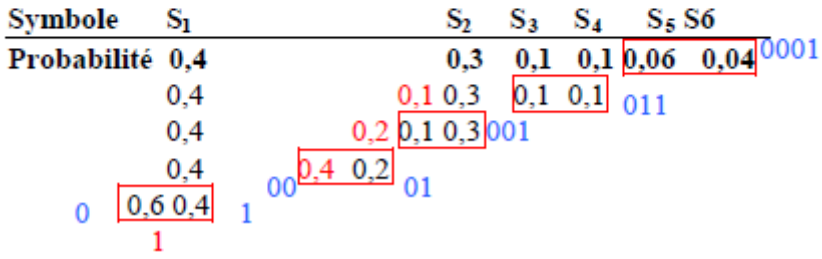
-Les codes d'Huffman sont formés pour chaque symbole en remontant l'arbre.

Pour Huffman, il faut définir le mot entropie qui signifie la mesure de la quantité moyenne d'information apportée par chaque niveau dans l'image.

$$Entropie = -\sum_{n=0}^{N} Pn(n)Q(n)$$

Ce qui signifie que l'entropie quantifie l'information dans le signal. On peut prendre l'exemple suivant sur 3 bits pour compresser une image.
On effectue l'arbre de représentation suivant :

Symbole	Proba	Code
S1	0,40	1
S2	0,30	001
S3	0,10	010
S4	0,10	011
S5	0,06	0000
S6	0,04	0001



On peut calculer aussi la longueur moyenne suivante:

Nb = 1*0,4+2*0,3+3*0,1+4*0,1+5*0,06+5*0,04

Nb = 2,2 bits/pixel (donc <3 bits/pixel recommander par la norme)

On peut déterminer l'entropie H de l'image :

Le principe est une prise en compte de la fréquence d'apparition d'un symbole dans l'ensemble du message.
On peut faire signifier que s'est le codage sur un nombre de bits réduit les symboles les plus probables, et sur des structures binaires plus longues les symboles les moins fréquentes

⇨ *MPEG* :

La norme MPEG (Motion Picture Expert Group) constitue une norme internationale ISO (International Standard Organisation) proposant des méthodes de compression de l'image animée (vidéo) et du son associé (audio). La norme MPEG1 a été adoptée en 1992. Elle vise des applications de compression audio et vidéo synchronisées à des débits de 1,5 Mbit/s. La partie audio vise à la fois à réduire le débit et à atteindre la qualité du disque compact audio.
Les fréquences d'échantillonnage du signal d'entrée peuvent être égales à 32 kHz (NICAM), 44.1 (Compact disc) et 48 kHz (enregistrement studio, DAT). Quatre modes de transmission sont prévus :

(1) Stéréo : codage des deux voies gauche et droite de manière indépendante.
2) Joint stéréo : on exploite la redondance entre les deux voies gauche et droite pour réduire le débit.
3) Dual_channel : deux voies sons indépendantes (par exemple, son bilingue).
4) Mono : une seule voie son.

Selon l'application, différentes couches du système de codage, de complexités et de performances croissantes, peuvent être utilisées :

• La couche I. Elle utilise l'algorithme PASC (Precision Adaptative Subband Coding) développé par Philips pour la cassette audio numérique (DCC). Elle utilise un débit fixe compris entre 32 et 448 kbit/s. La qualité subjective de type compact disc nécessite 192 kbit/s par voie audio, soit 384 kbit/s en stéréo. Elle a pour principal avantage une relative simplicité d'implémentation du codeur et du décodeur, mais elle n'est quasiment plus utilisée aujourd'hui.

• La couche II. Son algorithme est connu sous le nom de MUSICAM, standard retenu pour le DAB (radio numérique). Le débit fixe peut être choisi entre 32 et 192 kbit/s et la qualité compact disc nécessite 128 kbit/s par voie audio, soit 256 kbit/s en stéréo. La complexité du codeur et du décodeur est plus élevée qu'avec la couche I, mais le débit est réduit de 30 à 50 %. C'est cette couche qui est utilisée pour la télévision numérique par satellite en Europe (système DVB).

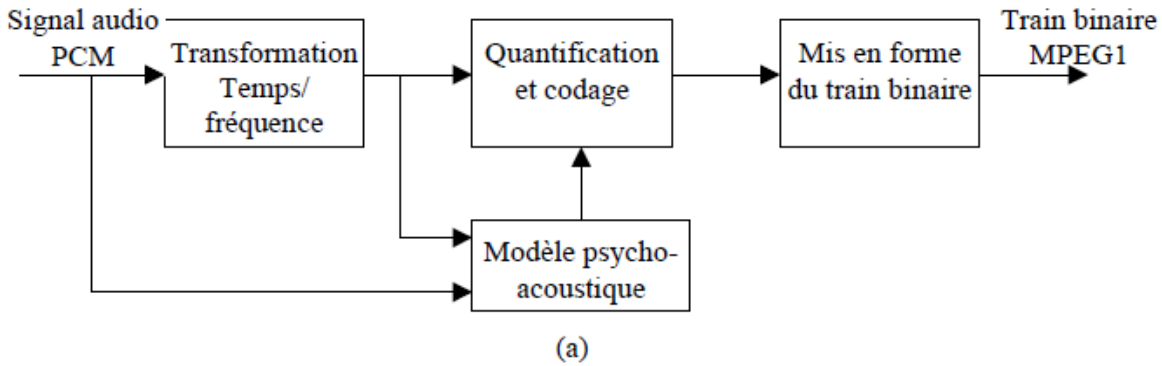
• La couche III. L'algorithme ASPEC (Advanced SPectre Entropy Coding) est un développement plus récent que les autres algorithmes. Beaucoup plus complexe à implémenter, il utilise un débit variable et la qualité compact disc est obtenue avec 64 kbit/s par voie audio, soit 128 kbit/s en stéréo, c'est à dire un taux de compression environ deux fois plus élevé que la couche II. Il s'agit de la norme utilisée dans les fameux fichiers MP3 que l'on trouve un peu partout sur Internet. C'est là sa principale application.
Pour résumé, les taux de compression obtenus pour une qualité subjective de type compact disc sont les suivants :

	débit	Taux de compression
Compact disc	2*44100*16 = 1378 kbit/s	1
MPEG1 couche I	384 kbit/s	3,6
MPEG1 couche II	256 kbit/s	5.4
MPEG1 couche III	128 kbit/s	10,8

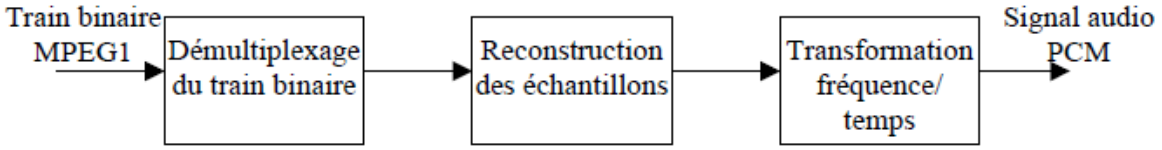
La norme MPEG2, plus récente, a été publiée en 1994. La partie audio, compatible avec MPEG1, apporte :

- le support du multicanal avec 5 voies haute fidélité plus une voie basse fréquence (5.1),
- le support multilinguistique avec 7 voies de commentaires possibles,
- le support des très bas débits jusqu'à 8 kbit/s,
- et l'extension des fréquences d'échantillonnage à 16, 22.05 et 24 kHz.

Il faut encore noter que la société Dolby a mis au point un algorithme concurrent de MPEG2 5.1 qui permet aussi de coder les voies sons (gauche, centre, droite, arrière gauche, arrière droite et basse) dans 384 kbit/s : le dolby digital anciennement appelé dolby AC-3. Les deux systèmes sont utilisés pour le DVD, mais seul MPEG2 5.1 a été retenu pour la télévision numérique en Europe dans le cadre du groupe DVB. La figure suivante représente le synoptique du codeur et du décodeur MPEG-1 audio.



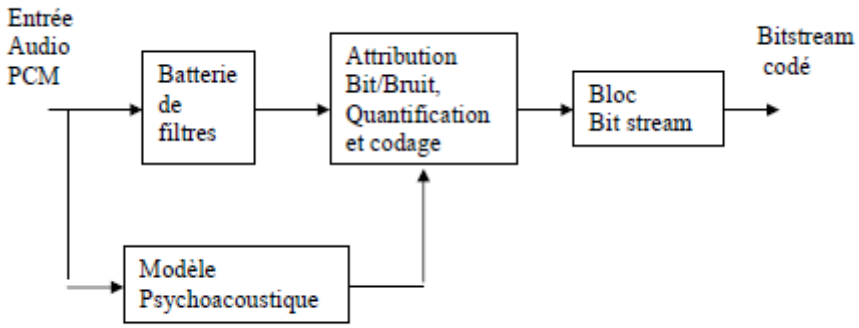
(a)



(b)

MP3 :

Le processus de la compression MP3 peut être décomposé en 3 étapes. D'abord, le jet audio d'entrée traverse une batterie de filtres qui divise le signal en sous-bandes de fréquence. Simultanément, il traverse un modèle psychoacoutique qui utilise le concept de masque d'attribution de bit réduit au minimum l'audibilité du bruit.



La batterie de filtre :

Le but de la batterie de filtres est de diviser le signal audio en 32 sous-bandes de fréquences d'égal largueur. L'oreille humaine a une résolution limité qui peut être exprimée en termes de largueurs de bande critiques moins que 100 Hz et plus de 4kHz. Ainsi la batterie de filtre crée les sous bandes de fréquence d'égal largueur qui se corrèlent avec les largueurs de bandes critiques.

Le modèle psychoacoustique :

La branche de la pscychoacoutique examine le concept de l'auditoire masquant et son effet sur la compression. Dans chaque sous bande où le flou se produit, la présence d'un signal fort, par exemple peut masquer une région des signaux plus faibles.

Attribution de bit :

Par un algorithme itératif, l'attribution de bit emploie l'information du modèle psychoacoustique pour déterminer le nombre de bit de code à assigner à chaque sous bande. Ce processus peut être décrit en utilisant la formule suivante :

MNR(dB) = SNR(dB)-SMR(dB)

Avec

MNR est le rapport de masque à bruit

SNR est le rapport signal/bruit, donné avec la norme audio MPEG

SMR est le rapport de signal à masque, dérivé du modèle psychoacoustique

4 Information structurée

4.1 Notion de type

4.1.1 Le type entier

Le type entier représente partiellement l'ensemble des entiers relatifs $\mathbb{Z}$ des mathématiciens. Alors que l'ensemble $\mathbb{Z}$ est infini, l'ensemble des valeurs défini par le type entier est fini, et limité par les possibilités de chaque ordinateur, en fait, par le nombre de bits utilisés pour sa représentation. Le type entier possède donc un élément minimum et un élément maximum. Chaque entier possède une représentation distincte sur l'ordinateur et la notation des constantes entières est en général classique : une suite de chiffres en base 10.

La cardinalité du type entier dépend de la représentation binaire des nombres. Un mot de n bits permet de représenter 2^n nombres positifs sur l'intervalle $[0,2^n-1]$. Réciproquement, le nombre de bits nécessaires à la représentation d'un entier n est $\log_2 n$. Afin de simplifier les opérations d'addition et de soustraction, les entiers négatifs sont représentés sous la forme complémentée, soit en complément à un, soit en complément à deux. En complément à un, la valeur négative d'un entier x est obtenue en inversant chaque position binaire de sa représentation.

4.1.2 Le type réel

Le type réel sert à définir partiellement ensemble $\mathbb{R}$ des mathématiciens. Les réels ne peuvent être représentés sur l'ordinateur que par des approximations plus ou moins fidèles. Le type réel décrit un nombre fini de représentations d'intervalles du continuum des réels. Si deux objets réels sont dans le même intervalle, ils auront la même représentation et ne pourront pas être distingués. De plus, les réels ne sont pas uniformément répartis sur l'ensemble : plus de la moitié est concentrée sur l'intervalle [-1,1].

4.1.3 Le type booléen

Le type booléen est un type fini qui représente un ensemble composé de deux valeurs logiques, vrai et faux, sur lequel les opérations de disjonction (ou), de disjonction exclusive (xou), de conjonction (et), et de négation (non) peut être appliquées.

4.1.4 Le type caractère

Chaque ordinateur possède son propre jeu de caractères. La plupart des ordinateurs actuels proposent plusieurs jeux de caractères différents et normalisés pour représenter les lettres et des chiffres de diverses langues, des symboles graphiques ou des caractères de contrôle.

Par le passé, seuls deux jeux de caractères américains étaient vraiment disponibles. Le jeu de caractères ASCII[19], codés sur 7 bits, ne permet de représenter que 128 caractères différents. Le jeu EBCDOC[20], spécifique à IBM, code les caractères sur 8 bits et inclut quelques lettres étrangères, comme β ou ü.

Pour satisfaire les usagers non anglophones, la norme ISO-8859 propose plusieurs jeux de 256 caractères codés sur 8 bits. Les 128 premiers caractères sont ceux du jeu ASCII et les 128 suivant correspondent à des variantes nationales. La norme ISO 88591-1, appelée Latin-1, correspond à la variante des pays de l'Europe de l'Ouest. Elle inclut des symboles graphiques ainsi que des caractères à signes diacritiques comme é, à, ç.

Depuis le début des années 1990, le Consortium Unicode développe une norme UNICODE pour définir un système de codage universel pour tous les systèmes d'écritures. UNICODE est un sur-ensemble de tous les jeux de caractères existants, en particulier de la norme ISO/CEI 10646. UNICODE propose trois représentations des caractères : UTF32, UTF16, et UTF8, respectivement codées sur 32, 16 et 8 bits.

4.2 Structures de données de base

4.2.1 Structures finies :

⇨ *Constantes* : Vous pouvez décider de donner une valeur à une variable et que cette valeur ne doit pas changer : elle doit rester fixe dans le temps et inaltérable, pour toute la durée du programme. Sa valeur doit rester constante. D'où son nom.

Une constante est une valeur, représentée tout comme une variable par une valeur, qui ne peut pas être modifiée après son initialisation. Elle est immuable. Un exemple de constante pourrait être la valeur de PI.

⇨ *Variables :* est un mot qui permet temporairement utiliser la mémoire.

4.2.2 Structures indexées :

⇨ *Tableaux :* est un agrégat de composants, objets élémentaires ou non, de même type et dont l'accès à ses composants se fait par indice calculé.

⇨ *Tableaux multidimensionnels* : les composants d'un tableau peuvent être de type quelconque et en particulier de type tableau. Les tableaux de tableaux sont souvent appelés tableaux à plusieurs dimensions.

⇨ *Tableaux associatifs* : Chaque clef est associée à une valeur : un tableau associatif correspond donc à une application en mathématiques. Du point de vue du programmeur, le tableau associatif peut être vu comme une généralisation du tableau : alors que le tableau traditionnel associe des entiers consécutifs à des valeurs d'un certain type, le tableau associatif associe des valeurs d'un type *arbitraire* à des valeurs d'un autre type.

Les opérations usuellement fournies par un tableau associatif sont :

- **ajout** : association d'une nouvelle valeur à une nouvelle clef ;
- **modification** : association d'une nouvelle valeur à une ancienne clef ;
- **suppression** : suppression d'une clef ;
- **recherche** : détermination de la valeur associée à une clef, si elle existe.

⇨ *Vecteurs* : un **vecteur** désigne un conteneur d'éléments ordonnés et accessibles par des indices, dont la taille est dynamique : elle est mise à jour automatiquement lors d'ajouts ou de suppressions d'éléments. On retrouve les vecteurs dans de nombreux langages de

[19] American Standard Code For Information Interchange
[20] Extended Binary Coded Decimal

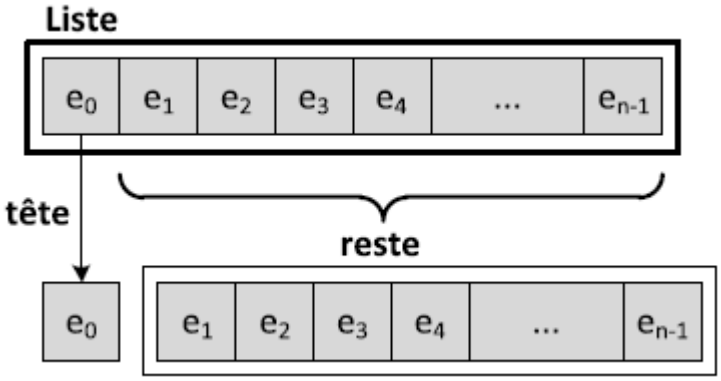
programmation, notamment le C++ et le Java. Ils sont alors inclus dans des librairies et l'utilisateur n'a pas besoin d'en programmer un.

- Vecteur = collection ordonnée d'éléments accessibles par un indice
- Indice = entier compris entre 0 et la taille courante -1
- Taille variable dans le temps (insertions / retraits) ≤ tailleMax

4.2.3 Structures récursives :

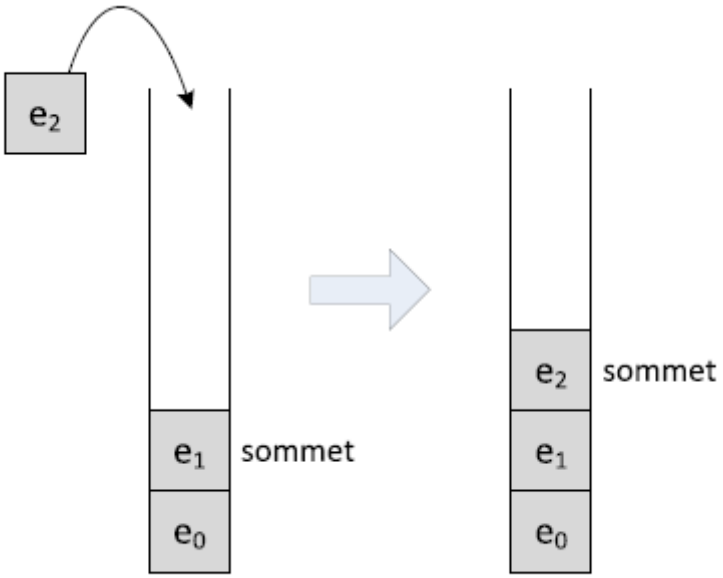
⇨ *Listes* : définit une forme générale de séquence. Une liste est une séquence finie d'éléments repérés selon leur rang. S'il n'y a pas de relation d'ordre sur l'ensemble des éléments de la séquence, il en existe une sur le rang. Le rang du premier élément est 1, le rang du second est 2, et ainsi de suite. L'ajout et la suppression d'un élément peut se faire à n'importe quel rang valide de la liste.

- Liste = collection ordonnée d'éléments (comme pour les vecteurs) décomposée en:
  - une tête qui définit l'élément en tête de la liste,
  - et un reste qui définit la liste restante.

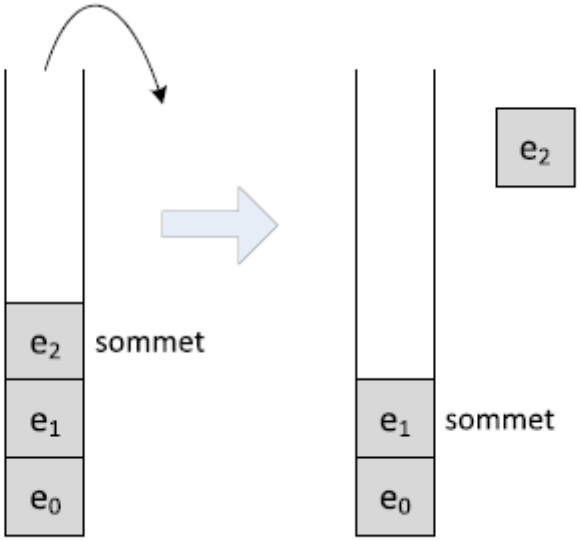


⇨ *Piles* : est une séquence d'éléments accessibles par une seule extrémité appelée sommet. Toutes opérations définies sur les piles s'appliquent à cette extrémité. L'élément situé au sommet s'appelle le sommet de pile.

(1) On empile



(2) On dépile

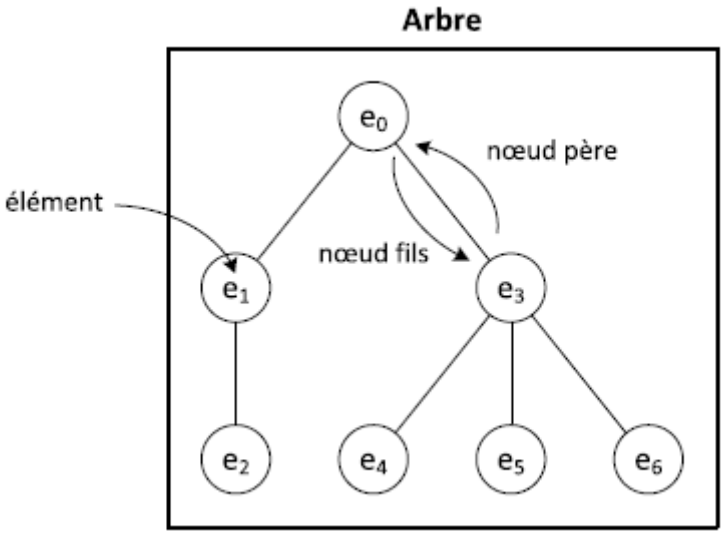


L'ajout et la suppression d'éléments en sommet de pile suivent le modèle dernier entré premier sorti (LIFO[21]). Les piles sont des structures fondamentales, et leurs emploi dans les programmes informatiques est très fréquent.

⇨ *Les files* : définissent le modèle premier entré-premier sorti (FIFO). Les éléments sont insérés dans la séquence par une des extrémités et se sont extraits par l'autre. Ce modèle correspond à la file d'attente que l'on rencontre bien souvent face à un guichet dans les bureaux de poste, ou à une caisse de supermarché la veille d'un week-end.

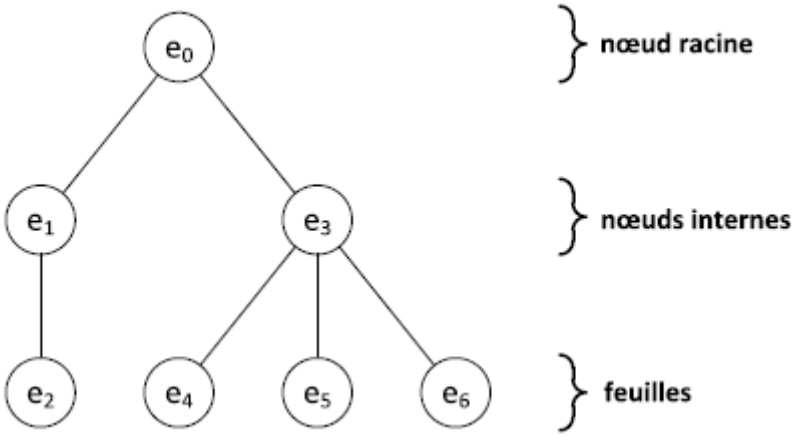
[21] Last-In First-Out

- ⇨ *Dèques* : Un déque possède à la fois les propriétés d'une pile et d'une file. On peut donc ajouter et supprimer un élément à chaque extrémité de la séquence. Les éléments de la séquence sont accessibles par les deux extrémités.
- ⇨ *Arbres* : On appelle arbre un ensemble de nœuds liés par une relation de type père/fils. Un nœud est égal à un élément plus liens vers les nœuds fils.



Du Nœud racine aux feuilles : Nœuds à trois cas de figure

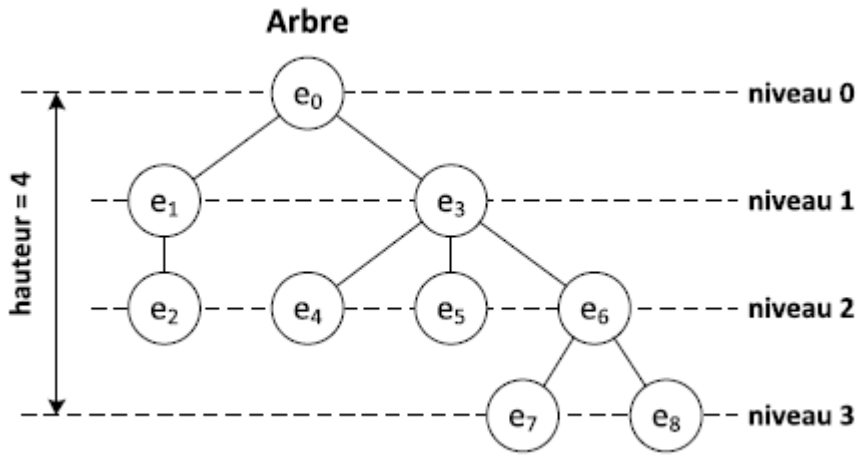
- Pas de parent: noeud racine
- Un parent et au moins un fils: noeud interne
- Un parent et aucun fils: feuille (ou noeud externe)



Conséquence

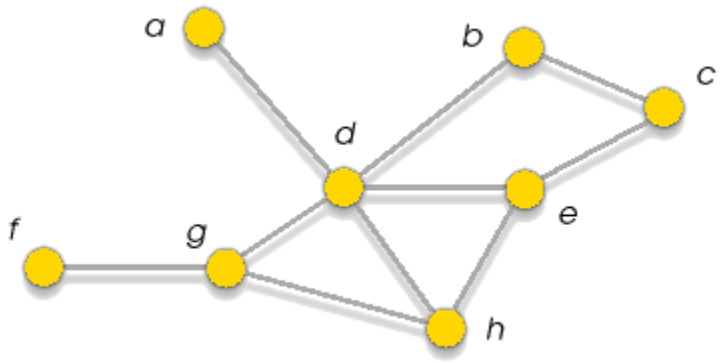
- ❖ Arbre = entièrement défini par son noeud racine
- ❖ Arbre vide = aucun noeud racine

Terminologie



⇨ Graphes : un graphe est un ensemble de sommets reliés par des arcs.

Sur la figure ci-dessous montre un graphe. Je reviendrais sur la théorie du graphe dans un prochain cours sur la théorie des graphes.



4.3 Format de fichier

Un fichier numérique est constitué de 0 et de 1. Mais pas n'importe comment. Chaque logiciel à sa manière propre d'encoder les informations. Certaines manières d'encodage ne sont lisibles que par logiciel lui-même. D'autres sont lus par d'autres logiciels du même type, puis quelques formats de fichiers sont lus par tous les logiciels de la famille. Par exemple le format word document (.doc, .docx) est produit et lu par Word mais aussi par d'autres logiciels de traitement de texte, mais pas par tous. Alors que le format texte Mis en forme ou RTF (.rtf) est lu et produit par tous les traitements de textes ou presque.

Catégories	**Extensions**	**Significations**
Traitement de texte	.docx, .odt	Word document, open office writer
Présentation	.pptx, .odp	PowerPoint, open office presentation
Tableur	.xls, .ods	Excel, open office tableur
Images	.jpg, .tiff, .bmp, .gif,.png	Jpeg, tiff,bmp, gif, PNG
Vidéos	.mov, .ogg, .mp4,	Quicktime, ogg, mpeg4,
Son	.wav, .mp3,	Media player, mpeg3
Texte mis en forme	.rtf	

5 Cryptologie

5.1) Définitions :

- ⇨ **La cryptographie** : étude et conception des procédés de chiffrement des informations.
- ⇨ **La cryptanalyse** : analyse des textes chiffrés pour retrouver les informations dissimulées.

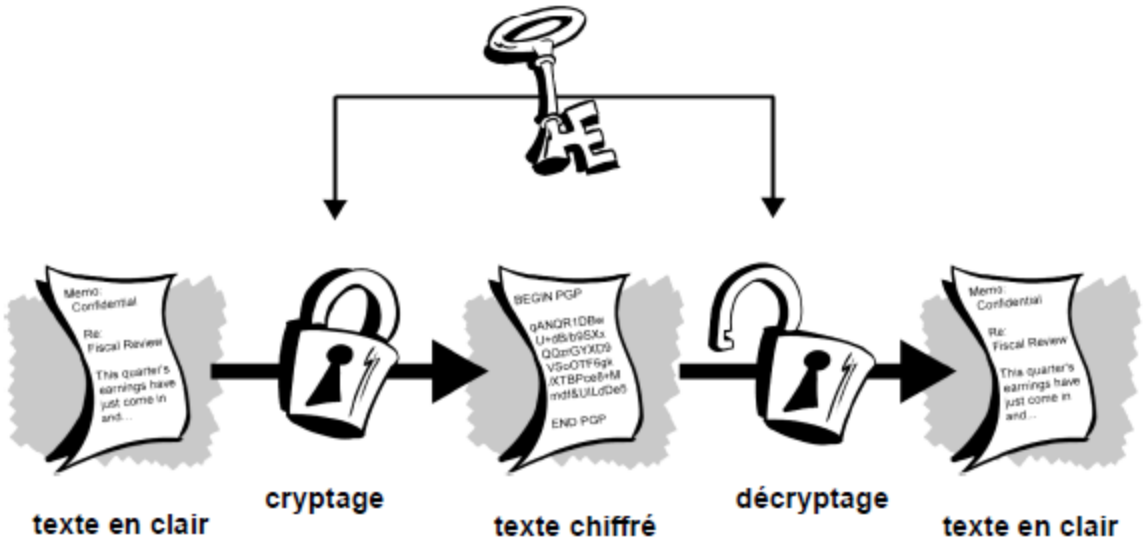
Bien distinguer cryptographie/ stéganographie :

- Cryptographie : transforme un message clair en cryptogramme
- Stéganographie : dissimule l'existence même de l'information secrète (encre sympathique etc…)

- ⇨ **Chiffrement** : le chiffrement consiste à transformer une donnée (texte, message,…) afin de la rendre incompréhensible par une personne autre que celui qui a créé le message et celui qui en est le destinataire. La fonction permettant de retrouver le texte clair à partir du texte chiffré porte le nom de déchiffrement.
- ⇨ **Texte chiffré** : Appelé également cryptogramme, le texte chiffré est le résultat de l'application d'un chiffrement à un texte clair.
- ⇨ **Clef** : Il s'agir du paramètre impliqué et autorisant des opérations de chiffrement et/ ou déchiffrement. Dans le cas d'un algorithme symétrique, la clef est identique lors des deux opérations. Dans le cas d'algorithmes asymétriques, elle diffère pour les deux opérations.
- ⇨ **Cryptanalyse** : opposée à la cryptologie, elle a pour but de retrouver le texte clair à partir de textes chiffrés en déterminant les failles des algorithmes utilisés.
- ⇨ **Cryptosystème** : Il est défini comme l'ensemble des clés possibles (espace de clés), des textes clairs et chiffrés possibles associés à un algorithme donné.

Cryptographie conventionnelle

En cryptographie conventionnelle, également appelé cryptage de clé secrète ou de clé symétrique, une seule clé suffit pour le cryptage et le décryptage. La norme de cryptage de données (DES) est un exemple de système de cryptographie conventionnelle utilisé par le gouvernement américain.



5.2) Principe de Kerckhoff :

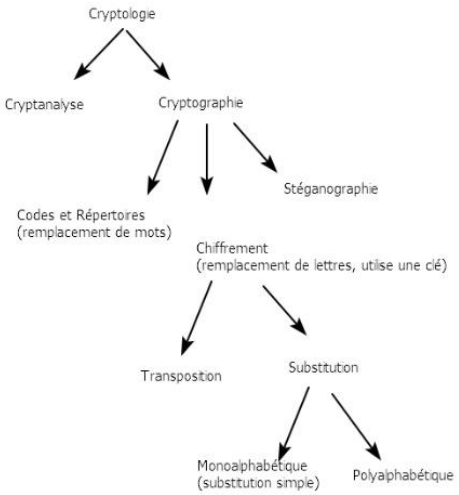
Kerckhoff a définit la phrase suivant :

La sécurité du chiffre ne doit pas dépendre de ce qui ne peut pas être facilement changé.

5.3) La cryptographie classique :

Dans ce schéma ci-dessous figurent les différentes branches de la cryptographie classique.

5.3) La cryptographie classique :



5.4) Chiffrement de César :

Le chiffrement de substitution est un exemple extrêmement simple de cryptographie conventionnelle. Il substitue une information par une autre. Cette opération s'effectue généralement en décalant les lettres de l'alphabet. Le code secret de Jules César est à la base de la cryptographie conventionnelle. Dans ce cas, l'algorithme constitue à décaler les lettres de l'alphabet et la clé correspond au nombre de caractères de décalage.

Par exemple, si vous codez le mot « SECRET » à l'aide de la valeur 3 de la clé de César, l'alphabet est décalé de manière à commencer à la lettre D.
Ainsi, l'alphabet

ABCDEFGHIJKLMNOPQRSTUVWXYZ

Si vous décalez le début de 3 lettres, vous obtenez

DEFGHIJKLMNOPQRSTUVWXYZABC

Où D = A, E = B, F = C, etc.

Avec ce procédé, le texte en clair « SECRET » est crypté en « VHFUHW ».
Pour autoriser un autre utilisateur à lire le texte chiffré, indiquez-lui que la valeur de la clé est égale à 3.
Évidemment, ceci est considéré comme une cryptographie extrêmement vulnérable de par les standards actuels. Mais, cette méthode convenait à César et illustre le mode de fonctionnement de la cryptographie conventionnelle.

DEUXIEME PARTIE

1 Les fondements de l'informatique

Un ordinateur est un ensemble de circuits électroniques permettant de manipuler des informations qu'on appelle des données et capable de faire « tourner » des programmes. On appelle programme une suite ou séquence d'instructions programmées à l'avance et qu'il va dérouler du début à la fin dans le but d'obtenir des résultats.

C'est **Von Neumann** qui a défini en 1944 l'architecture des ordinateurs modernes encore largement utilisée aujourd'hui. L'architecture de Von Neumann décompose l'ordinateur en quatre parties distinctes :

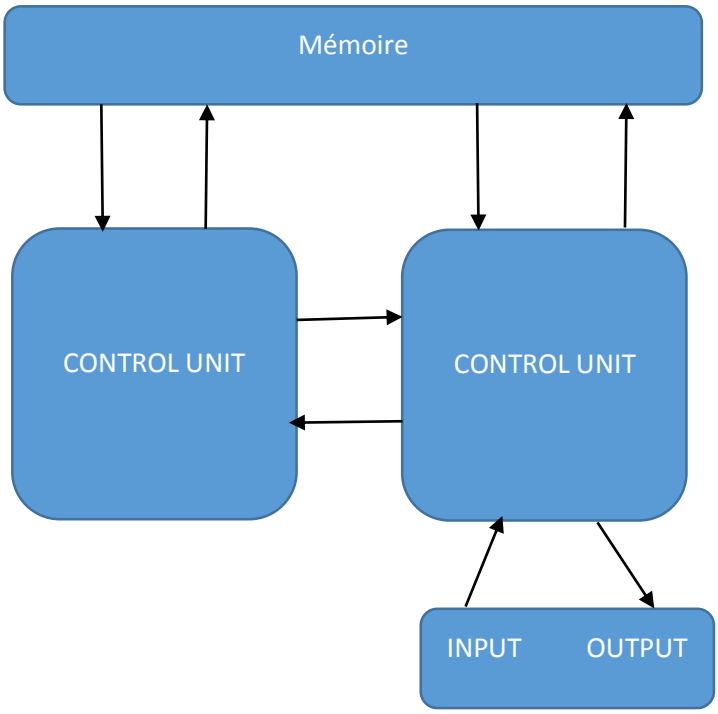
1. L'***Unité Arithmétique et logique UAL*** (ALU en anglais) est l'organe de l'ordinateur qui exécute les calculs : additions, soustractions, multiplications, divisions, modulos, gestion des décalages de valeurs. Il existe des UAL spécialisées dont les nombres à virgule flottante, d'autres dans des traitements complexes comme les logarithmes, les inversions, les racines, les vecteurs, les calculs trigonométriques, etc. Certaines documentations lui rajoutent quelques registres[22] et lui donnent le nom de ***processeur*** (CPU).
2. L'**Unité de Contrôle UC** (CU en anglais), à ne pas confondre avec l'Unité Centrale, contrôle le séquençage des opérations, autrement dit le déroulement du programme. Elle prend ses instructions dans la mémoire et donne ses ordres à l'UAL. Les résultats retournés peuvent influer sur le séquençage. L'UC passe alors à l'instruction suivante ou à une autre instruction telle que le programme lui ordonne effectuer.
3. La **mémoire** peut être décrite comme une suite de petites cases numérotées, chaque case pouvant contenir une petite information (la taille de chaque case[23]a une taille fixe[24]). Cette information peut-être une instruction ou un morceau d'instruction du programme ou une donnée. C'est l'UC qui a comme rôle central de contrôler l'accès à la mémoire pour le programme et les données. Chaque numéro de case est appelé une adresse. Pour accéder à la mémoire, il suffit de connaître son adresse. Les instructions du programme pour l'UC et les données pour l'UAL sont placées dans des zones différentes de la même mémoire physique.
4. **Les entrées/ sorties** E/S (I/O en anglais) permettent de communiquer avec le monde extérieur et donc cela peut être un clavier, une souris, une clé usb, un écran, une imprimante, un haut-parleur, un casque.

Les instructions du programme sont présentes dans la mémoire. L'unité de contrôle va prendre la première instruction du programme et l'exécuter. Si l'instruction est par exemple d'additionner deux nombres, elle va demander à l'UAL de prendre ces deux nombres en mémoire et de les additionner et éventuellement de placer le résultat dans une nouvelle case. Puis l'UC passe à l'instruction suivante. Si elle consiste à afficher ce résultat, alors l'UC va lire le contenu de la mémoire à l'adresse où est placé le résultat via le composant d'E/S adéquat. Et ainsi de suite. Au final le déroulement d'un programme au sein de l'ordinateur est le suivant :

- L'UC extrait une instruction de la mémoire ;
- Analyse l'instruction ;
- Recherche en mémoire les données concernées par l'instruction ;
- Déclenche l'opération adéquate sur l'ALU ou l'E/S

[22] Termes utilisés en microprocesseur pour désigner des petites cases de mémoires intégrées à l'UAL.
[23] Dite les registres
[24] Certains langage informatique comme C et C++ utilise les registres et idem pour les langages proches de la machine dite langage assembleur.

- Range le résultat dans la mémoire



2. La machine de Turing

Avant même l'apparition des premiers vrais ordinateurs programmables, Alan Turing avait défini en 1936 ce qu'on appelle la **Machine de Turing**.

Une machine de Turing n'étant pas une vraie machine, il suffit pour s'en servir soit de se servir de sa tête (réflexion et mémoire), soit d'un crayon qui fera office de **tête de lecture**, d'une longue bande de papier décomposée en cases qu'on appelle **ruban**, et d'une table de symboles et de procédures liée à l'état de la case à respecter quand on tombe sur une case contenant un symbole donné. On se place sur la première case, on vérifie son symbole et son état associée, on exécute la procédure associée et on continue à dérouler ce « programme » jusqu'à ce que la procédure vérifiant qu'on a obtenu le résultat final soit vérifiée. On vient de dérouler un programme et l'ensemble symboles/procédure décrit ce programme. C'est l'ancêtre de l'algorithme.

Tables de vérité pour les lois De Morgan :

$\overline{a+b} = \overline{a}.\overline{b}$

a	b	$a+b$	$\overline{a+b}$	$\overline{a}$	$\overline{b}$	$\overline{a}.\overline{b}$
0	0	0	1	1	1	1
0	1	1	0	1	0	0
1	0	1	0	0	1	0
1	1	1	0	0	0	0

$\overline{a.b} = \overline{a} + \overline{b}$

a	b	$a.b$	$\overline{a.b}$	$\overline{a}$	$\overline{b}$	$\overline{a}+\overline{b}$
0	0	0	1	1	1	1
0	1	0	1	1	0	1
1	0	0	1	0	1	1
1	1	1	0	0	0	0

- Fonctions logiques composées
  - OU exclusif : $a \oplus b = \bar{a}.b + a.\bar{b}$
  - Équivalence : $a \Leftrightarrow b = \overline{a \oplus b} = \bar{a}.\bar{b} + a.b$

Ou exclusif

a	b	$a \oplus b$
0	0	0
0	1	1
1	0	1
1	1	0

Equivalence

a	b	$a \Leftrightarrow b$
0	0	1
0	1	0
1	0	0
1	1	1

3. Algèbre de Boole

3.1 Définitions et théorèmes

On appelle Algèbre de Boole tout quadruplet constitué d'un ensemble non vide muni d'au moins deux éléments différents, d'une loi unaire notée et deux lois binaires noté +, et . , satisfaisant les postules d'Huntington définis par :

Postulat 1 : la loi + (appelée OU logique) est une loi de composition interne : $\forall x, y \in B \;\; x + y \;\in B$.

Postulat 2 : la loi . (appelée ET logique) est une loi de composition interne : $\forall x, y \in B \;\; x.y \;\in B$.

Postulat 3 : il existe un élément, noté 0, neutre pour la loi + : $\forall x \in B \;\; x + 0 = x$

Postulat 4 : il existe un élément, noté 1, neutre pour la loi . : $\forall x \in B \;\; x.1 = x$

Postulat 5 : la loi + est commutative : $\forall x, y \in B \;\; x + y = y + x$

Postulat 6 : la loi . est commutative : $\forall x, y \in B \;\; x.y = y\,.x.$

Postulat 7 : la loi + est distributive par rapport à la loi . : $\forall x, y, z \in B \;\; x + (y.z) = (x + y).(x + z)$

Postulat 8 : la loi . est distributive par rapport à la loi + : $\forall x, y, z \in B \;\; x.(y + z) = (x.y)(x.z)$

Postulat 9 : la loi $^-$ (appelée Complément logique ou opérateur NON) vérifie :

$$\forall x \in B \quad \exists \bar{x} \in B \quad \begin{cases} x + \bar{x} = 1 \\ x.\bar{x} = 0 \end{cases}$$

Théorèmes fondamentaux :

Th 1 : éléments absorbants

$$\forall x \in B \quad x + 1 = 1$$

$$\forall x \in B \quad x.0 = 0$$

Th 2 : idempotence

$$\forall x \in B \quad x + x = x$$

$$\forall x \in B \quad x.x = x$$

Th 3 : involution

$$\forall x \in B \quad \bar{\bar{x}} = x$$

Th 4 : absorption

$$\forall x \in B \quad x + x.y = x$$

$$\forall x \in B \quad x.(x + y) = x$$

Th 5 : simplification

$$\forall x, y \in B \quad x + \bar{x}.y = x + y$$

$$\forall x, y \in B \quad x.(\bar{x} + y) = x.y$$

Th 6 : associativité

$$\forall x, y, z \in B \quad x + (y + z) = (x + y) + z = x + y + z$$

$$\forall x, y, z \in B \quad x.(y.z) = (x.y).z = x.y.z$$

Th 7 : consensus

$$\forall x, y, z \in B \quad x.y + \bar{x}.z + y.z = x.y + \bar{x}.z$$

$$\forall x, y, z \in B \quad (x + y).(\bar{x} + z).(y + z) = (x + y).(\bar{x} + z)$$

Th8 : théorème de De Morgan

$$\forall x_0, \dots, x_n \in B \quad \overline{x_0 + \cdots + x_n} = \overline{x_0}. \dots . \overline{x_n}$$

$$\forall x_0, \dots, x_n \in B \quad \overline{x_0 . \dots . x_n} = \overline{x_0} + \dots + \overline{x_n}$$

3.2 Tables de Karnaugh

Une table de Karnaugh est similaire à une table de vérité, à deux différences près :

- Elle n'est pas disposée linéairement en colonne (les valeurs de la fonction étant les unes en dessous des autres) mais dans un tableau à deux dimensions (lignes-colonnes) ;

Par exemple, un modèle de table de Karnaugh à trois variables ressemble à :

yz / *x*	00	01	11	10
0				
1				

Pour quatre variables :

yz / *wx*	00	01	11	10
00				
01				
11				
10				

Remplissage des tables de Karnaugh :

Le remplissage d'une table de Karnaugh se fait de la même manière que le remplissage d'une table de vérité. Chaque monôme correspond à un groupe de cases adjacentes ayant la valeur 1. Si on considère une fonction booléenne $f(x_1,x_2,\ldots,x_n)$ de B^n dans B, alors un monôme comprenant p littéraux consistera en un regroupement de 2^{n-p} cases à 1. Notons que si la fonction est définie sous forme conjonctive, chaque monal comportant p littéraux consistera en un regroupement de 2^{n-p} cases à 0.

Prenons exemple d'une fonction que nous allons remplir dans un table de Karnaugh :

$$f(w,x,y,z) = \overline{w}.\bar{x}.z + \overline{w}.x + \bar{x}.\bar{y}.z + \bar{x}.y.z.$$

Le premier monôme, $\overline{w}.\bar{x}.z$, comporte trois littéraux. Il va donc permettre de remplir deux cases à 1. Ces deux cases sont celles pour lesquelles w=0, x=0 et z=1.

Le deuxième monôme ($\overline{w}.x$) ayant deux littéraux, il va correspondre à un groupement de quatre cases à 1. Ce sont les cases qui correspondent à w=0 et x=1, c'est-à-dire la deuxième ligne.

On procède de la même manière pour les deux monômes suivants.

Le troisième monôme ($\bar{x}.\bar{y}.z$) correspond à deux case : celles pour lesquelles x=0, y=0 et z=1. On note qu'une de ces deux cases est déjà à 1. Le dernier monôme (w=0, x=0 et z=1) correspond lui aussi à deux cases.

Au final, on obtient la table suivante :

yz / wx	00	01	11	10
00		1	1	
01	1	1	1	1
11				
10		1	1	

Détermination d'une forme disjonctive simplifiée (base minimale) :

La détermination d'une forme simplifiée comporte deux étapes : la détermination des monômes les plus élémentaires intervenant dans la fonction, puis la recherche d'une forme simplifiée à partir de cette liste de monômes.

Pour déterminer les monômes les plus élémentaires de la fonction, on réalise des regroupements de cases portant la valeur 1 dans la table de Karnaugh. Ces regroupements doivent être tel que :

- Le nombre de cases dans le regroupement est une puissance de 2 (1, 2, 4, 8 etc). Cela garantit que ce regroupement correspond à un monôme.
- Ce regroupement doit être le plus grand possible. Cela garantit que le monôme correspondant est premier.

Pour déterminer l'expression algébrique du monôme correspondant à un regroupement, il suffit de constituer un monôme ne faisant apparaître que les variables qui gardent une valeur constante pour toutes les cases du regroupement, en complémentant celles qui valent zéro.

À partir de la liste de ces monômes élémentaires, on va déterminer une expression minimale de la fonction. On va utiliser deux étapes pour cela :

- On déterminer d'abord tous les monômes essentiels. Il s'agit des monômes qui correspondent à des regroupements dont une case au moins n'est contenue dans aucun autre regroupement.
- On détermine ensuite les monômes non essentiels qui complètent la liste précédemment établie pour couvrir la fonction, en cherchant à minimiser leur nombre, puis leur taille.

Exemple : On reprend l'exemple suivant :

$$f(w,x,y,z) = \overline{w}.\overline{x}.z + \overline{w}.x + \overline{x}.\overline{y}.z + \overline{x}.y.z.$$

yz / wx	00	01	11	10
00		1	1	
01	1	1	1	1
11				
10		1	1	

La liste des monômes élémentaires est : $\overline{w}.x$ (deuxièe ligne du tableau, w est constant et égal à zéro, x est constant et égal à un, y et z changent de valeur selon la case du regroupement), $\overline{w}.z$ (regroupement à cheval sur les deux premières lignes du tableau) et $\bar{x}.z$ (regroupement à cheval sur la première et la quatrième ligne).

Le monôme $\overline{w}.x$ est obligatoire, car la case située à l'extrême gauche et celle située à l'extrême droite ne figurent qua dans celui-ci. De même, le monôme $\bar{x}.z$ est obligatoire car les deux cases situées sur la dernière ligne n'apparaissent que dans ce monôme. Les monômes obligatoires $\overline{w}.x$ et $\bar{x}.z$ couvrant la fonction, ils constituent une expression minimale. L'expression simplifiée de la fonction est donc :

$f(w,x,y,z) = \overline{w}.x + \bar{x}.z$

Détermination d'une forme conjonctive simplifiée :

L'algèbre de Boole étant duale, on va pouvoir reproduire le principe mis en lumière au paragraphe précédent pour obtenir une forme conjonctive simplifiée. On procédera de la même manière :

- On déterminera les monaux élémentaires de la fonction en constituant des regroupements de cases portant la valeur 0 tels que :
  - Le nombre de cases dans ces regroupements soit une puissance de 2 (1, 2, 4, 8, etc). Cela garantit que ces regroupements correspondent à des monaux.
  - Ces regroupements soient les plus grand possibles. Cela garantit que les monaux correspondant soient les plus simples possibles.

Reprenons l'exemple précédent :

$$f(w,x,y,z) = \overline{w}.\bar{x}.z + \overline{w}.x + \bar{x}.\bar{y}.z + \bar{x}.y.z.$$

yz / wx	00	01	11	10
00	0			0
01				
11	0	0	0	0
10	0			0

On peut procéder aux regroupements pour faire apparaître les monaux élémentaires. On a :

yz / wx	00	01	11	10
00	0			0
01				
11	0	0	0	0
10	0			0

Les monaux élémentaires sont : $\overline{w} + \bar{x}$ (troisième ligne du tableau, w est constant et égal à un, x est constant et égal à un, y et z changent de valeur selon la case du regroupement), x+z (les quatre coins du tableau, qui sont adjacents !) et $\overline{w} + z$ (le regroupement à cheval sur la première et la quatrième colonne).

Les deux premiers monaux sont obligatoires : $\overline{w} + \bar{x}$ car il est le seul à couvrir les deux cases du centre de la troisième ligne, et x+z car il est le seul à couvrir la première et la dernière case de la première ligne.

Ces deux monaux obligatoires couvrant la fonction, une forme simplifiée conjonctive est donc :

$f(w, x, y, z) = (\overline{w} + \bar{x}).(x + z)$

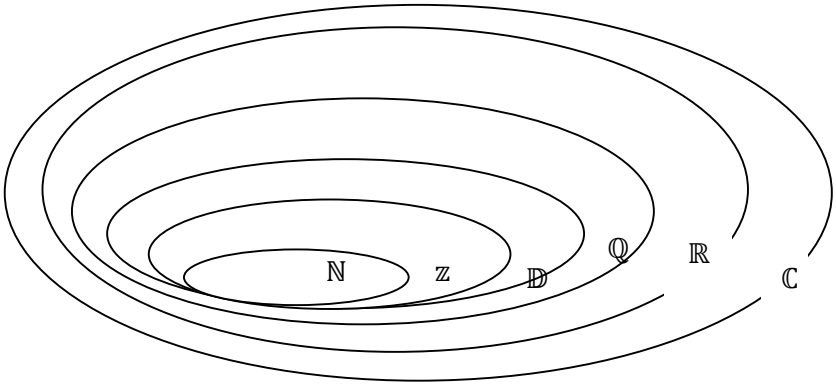
ANNEXE

Les notations mathématiques

Ensembles de nombres

On définit un ensemble s de nombre par le symbole Ω :

Voici l'ordre de rangement de l'ensemble de nombre:



Symbole	Appellation
$\mathbb{N}$	Ensemble des entiers naturels
$\mathbb{z}$	Ensemble des entier relatifs
$\mathbb{D}$	Ensemble des décimaux
$\mathbb{Q}$	Ensemble des rationnels
$\mathbb{R}$	Ensemble des réels
$\mathbb{C}$	Ensemble des complexes

Soit $\Omega \in \mathbb{N} \subset \mathbb{z} \subset \mathbb{D} \subset \mathbb{Q} \subset \mathbb{R} \subset \mathbb{C}$.

Définitions:

1. Un entier naturel est un nombre entier et positif.

Tous les entiers naturels forment un ensemble noté $\mathbb{N} = \{0; 1; 2; \dots\}$.

Ex: $26 \in \mathbb{N}$ *se lit* appartient à N.

2. Un entier relatif est un nombre pouvant être positif ou négatif.

Tous les entiers relatifs forment l'ensemble noté $\mathbb{z} = \{\dots; -2; -1; 0; 1; 2; \dots\}$

3. Un nombre décimal est un quotient d'un nombre entier par une puissance de 10.

Ex: $3{,}2 = \frac{32}{10^1} est\ un\ nombre\ décimal.$

Remarque: *Un nombre décimal est un nombre dont la partie décimale est finie, c'est-à-dire qui n'a qu'un nombre fini de chiffres après la virgule*.

4. Un nombre rationnel est un quotient de deux nombres: $\frac{p}{q}\ tel\ que\ p \in \mathbb{Z}\ et\ q \in \mathbb{N}$

Exemple 2/3 est un nombre rationnel.

5. Un nombre réel est tous les nombres qui appartiennent à l'ensemble des points précédents définis.

6. Un nombre imaginaire est un nombre réel de la racine $i^2 = -1$, donc de l'écriture $a+i^2b$.

Les opérateurs logiques

&	conjonction	... & ...	à la fois ... et...
⋁	disjonction	… ⋁…	soit .... soit, ou... ou...
~	négation	~…	ce n'est pas dans ce cas que
⊃	implication	…⊃...	si… alors…
≡	bi-implication	…≡...	... si et seulement si, identique à
∀	quantifieur universel	$\forall x$ …	pour tous x…
∃	quantifieur existentiel	$\exists x$ …	ils existent x tel que…

→	implique à		
↔	équivalent à		
∅	ensemble vide		
∈	appartient à	∉	n'appartient pas à
∋	contient comme élément	∌	ne contient pas de membre
⊂	est inclus dans	⊄	ce n'est pas inclus
∪	union ou "ou"		
∩	intersection ou "et"		
E^n	produit cartésien de n ensemble de E		
EΔF	différence symétrique des deux ensembles E et F		
{}	une famille d'élément		
|	tel que		
:	tel que		
⊆	sous-ensemble à		
A'	complément de A		

Relations

$x = y\ (mod.\mathcal{R})$ x congru à y modulo $\mathcal{R}$

Lois de composition

$E/\mathcal{R}$ ensemble quotient de l'ensemble E par la relation d'équivalence $\mathcal{R}$

$\leq$ inférieur ou égal à ; < strictement inférieur à

$\geq$ supérieur ou égal à ; > strictement supérieur à

$x\ \mathcal{R}\ y$ x est lié à y par la relation $\mathcal{R}$

$f: x \longmapsto f(x)$ x a pour image f(x) par l'application f

*, T, $\perp$, o étoile, truc, antitruc, rond

$\sum_{i=1}^{n} x_i = x_1 + x_2 + \cdots + x_n$ somme des n éléments x_1, x_2, …x_n

$\prod_{i=1}^{n} x_i = x_1 . x_2 . \ldots x_n$ produit des n éléments x_1, x_2, …x_n

$\frac{df}{dx}$ $ou\ f'(x)$ dérivée de f; f'' dérivée seconde de f; $f^{(n)}$ dérivée nième de f

df différentielle de f; df_{x0} ou $d_{x0}f$ différentielle de f au point x_0;

$\int f(x)dx$ intégrale de f; $\iint$ intégrale double ; $\iiint$ intégrale triple

$|x|$ valeur absolue de x ou module de x; $\|x\|$ norme de x

n! factorielle n

e élément neutre, base exponentielle

f, g, h application, fonction

f^{-1} application réciproque de l'application f

f(x) image de l'élément x par l'application f

I ensemble d'indices

Formules

- **Un polynôme** : est une fonction de x de la forme de $P(x) = a_0x^0 + a_1x^1 + a_2x^2 + \cdots + a_nx^n$. On note P(x) de la forme générale comme $P(x) = \sum_{i=0}^{n} a_i x^i$ avec a_i appartenant au domaine des Réels et imaginaires, décimales.

- **La Transformée de Fourier** : est de la forme de H(u) (traitement des signaux analogiques) suivant:

$$H(u) = \int_{-\infty}^{+\infty} h(x)e^{-2iu\pi x}dx$$

- **Intégration par partie**: On la note IPP.
  Cette intégrale est la forme suivante:
  $V(x) = \int_a^b P(x)Q(x)dx$ avec P(x) et Q(x) sont deux fonctions de x.

  La méthode est la suivant que je vais employer par la suite:

  P(x)= P'(x)=
  Q'(x) = Q(x) =

  Donc la méthode IPP, nous donnes le calcul suivant:

  $V(x) = [P(x)Q(x)]_a^b - \int_a^b P'(x)Q(x)dx$
  Ce qui nous donne maintenant :

$$V(x) = P(x)_bQ(x)_b - P(x)_aQ(x)_a - \int_a^b P'(x)Q(x)dx$$

- **Formule d'Euler**:

$\cos(x) = \frac{e^{ix}+e^{-ix}}{2}$ et $\sin(x) = \frac{e^{ix}-e^{-ix}}{2i}$

- **Sinus cardinale**:

$sinc(\pi u) = \frac{\sin(\pi u)}{\pi u}$

Aspects temporels et fréquentiels de l'échantillonnage

L'obtention d'un signal échantillonné $x_{ech}(k.T_{ech})$ à partir d'un signal analogique x(t) peut être modélisée mathématiquement dans le domaine temporel par la multiplication de x(t) par un peigne de Dirac de période Tech (noté d_{Tech} (t)):

$$x_{ech}(k.T_{ech}) = x(t).\delta_{Tech}(t) = x(t).\sum_{k}\delta(t - kT_{ech})$$

L'échantillonnage est illustré graphiquement dans le domaine temporel aux points (i), (ii) et (iii) (Figure 4).

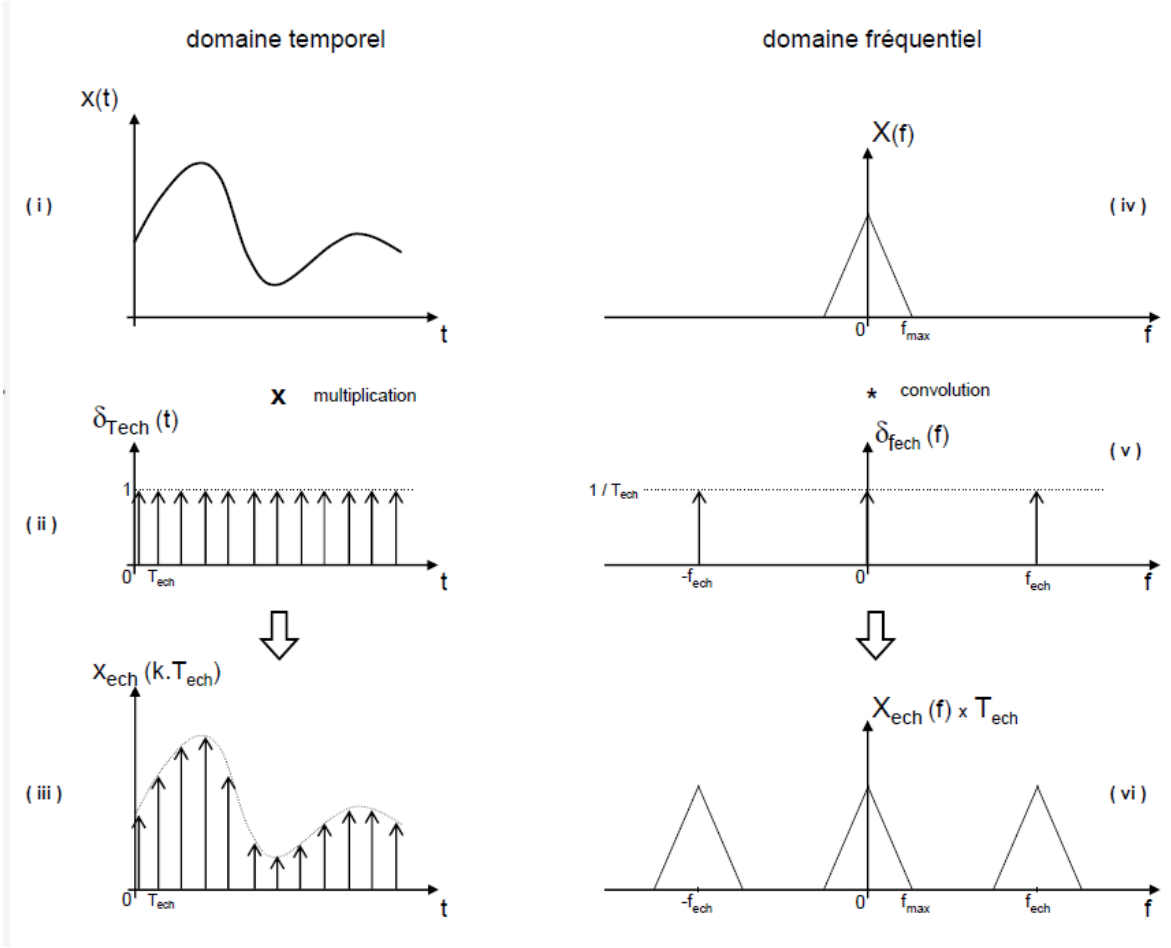


Figure 4: Échantillonnage d'un signal analogique

L'échantillonnage peut également être décris graphiquement dans le domaine fréquentiel.

Au signal analogique x(t), est associé dans le domaine fréquentiel le spectre X(f) (cf. Figure 4.iv) s'étendant sur une bande de fréquence de –fmax à fmax.
L'on rappelle un certain nombre de résultats démontrés en cours d'analyse de Fourier :

- Une multiplication dans le domaine temporel correspond à un produit de convolution dans le domaine spectral (et inversement),

- La transformée de Fourier d'un peigne de Dirac temporel, de période Tech, et d'amplitude 1, est un peigne de Dirac dans le domaine fréquentiel, de période $f_{ech} = 1 / T_{ech}$ et d'amplitude $1 / T_{ech}$.

Ainsi, à la multiplication temporelle x(t). $\delta_{Tech}(f)$ on fait correspondre dans le domaine fréquentiel le produit de convolution $X(f) * \delta_{Tech}(f)$ (δ_{Tech} (f) étant la transformée de Fourier de δ_{Tech} (t), cf. point (v) de la Figure 4). Le résultat de ce produit de convolution (Fig. 4.vi) est la transformée de Fourier du signal échantillonné $x_{ech}(k.T_{ech})$. On obtient le spectre X(f) répété à toutes les fréquences multiples de la fréquence d'échantillonnage (centrés sur les $k.f_{ech}$, k entier), à un facteur multiplicatif près sur l'amplitude Tech introduit par le peigne fréquentiel de Dirac.

Une approche graphique dans le domaine spectrale permet d'illustrer la récupération de l'information contenue dans un signal échantillonné par un filtrage passe bas (cf. Fig 5). En supposant un filtrage passe bas parfait (un tel filtre est impossible à réaliser) sur la bande de fréquence de $-f_{ech}/2$ à $f_{ech}/2$ (appelée **bande de Nyquist**, la fréquence $f_{ech}/2$ étant appelée **fréquence de Nyquist**), on retrouve le spectre X(f) et donc le signal temporel qui y correspond x(t).

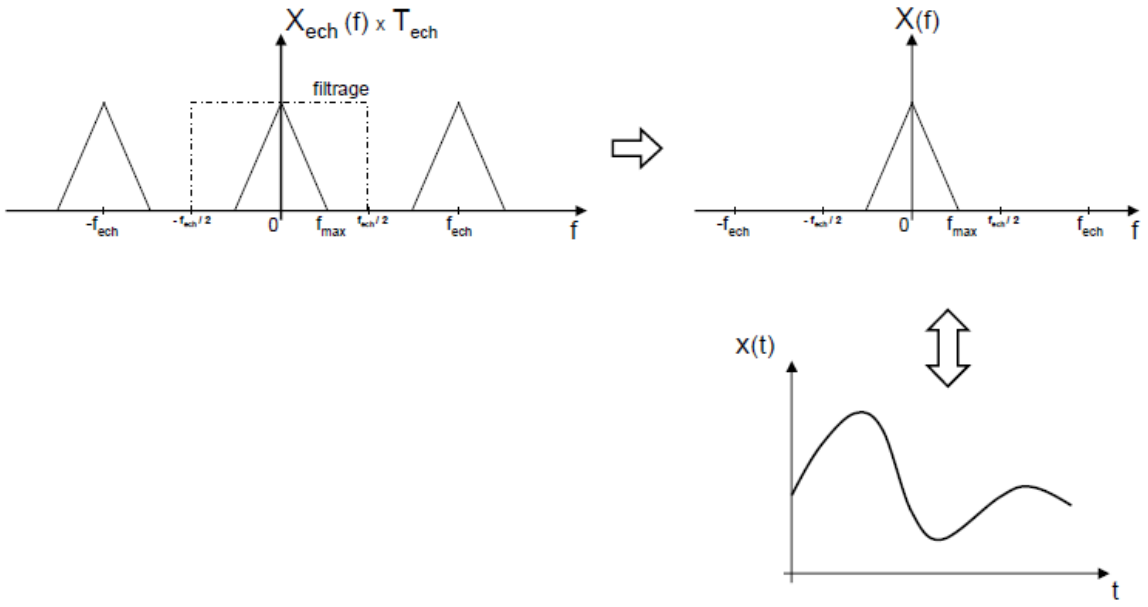


Figure 5 récupération de l'information par filtrage passe bas

Quantifications

A) Quantification scalaire :

Principe:

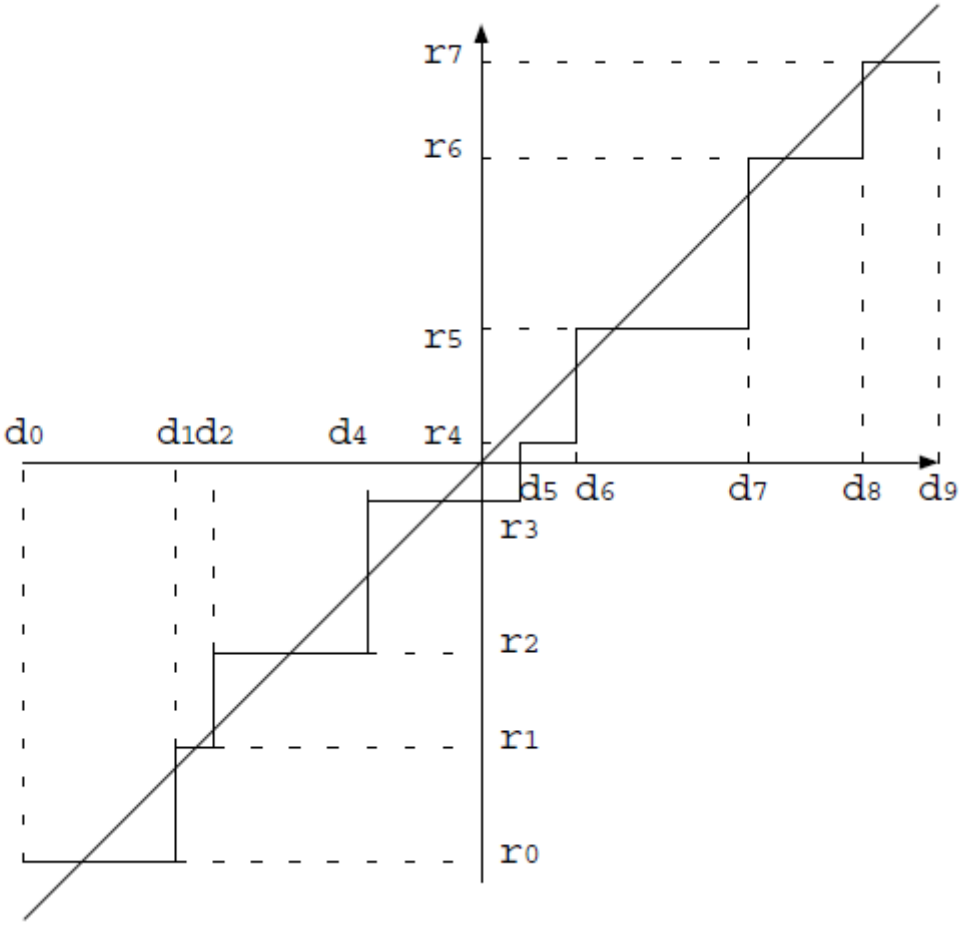
1. Diviser l'intervalle d'entrées [a, b] en N_q sous-intervalles {[d_i , d_{i+1}[}, $i \in$ {0..Nq}. d_i seuils de décision.
2. Associer à chaque sous-intervalle [d_i , d_{i+1}[une valeur r_i (niveau de reconstruction).
3. On code une donnée d de S par r_i si $d \in [d_i, d_{i+1}[$

En pratique, on envoie l'indice de l'intervalle. Le décodeur doit connaitre le niveau de reconstruction.
Le taux de compression possible est de Card(E(S))/N_d .
La quantification peut être uniforme (seuils de décisions équidistants) ou non.

Si on connait la statistique de la source, on peut définir un quantificateur optimal.

Pour cela, on cherche l'ensemble {seuils de décision – niveau de reconstruction} qui minimise l'EQM du signal reconstruit.

Si les symboles en entrées sont uniformément distribués (probabilité uniforme de la fonction de distribution), un résultat classique montre que la quantification optimale peut être obtenue en considérant des intervalles égaux.



B) Quantification vectorielle :

On associe un ensemble de k valeurs d'entrée successives (un vecteur) à un élément d'un livre de codes ou dictionnaire (codebook).

fonction $E(S)^k \mapsto L$, où L = {V_i}, $i \in \{1, 2, \ldots . M\}$

Le taux de compression possible est Card(E(S))^k/Card(L).

Pour des images, on considère plutôt de petites zones carrées (2×2 ou 4×4) dans l'image.

Le problème est de déterminer:

⇨ Le livre de codes permettant de coder efficacement une image.

Cela peut se faire sur un ensemble prédéfini d'apprentissage ou sur l'image à coder. Ce dernier cas donne de meilleurs résultats, mais le livre de code doit être envoyé au décodeur.

⇨ le code le plus ressemblant à un bloc donné

Quelques notions de codage

A) Calcul de l'Efficacité d'un codage

- Taux ou facteur de compression

Mesure la réduction opérée sur le volume des données lors du codage

$$TC = \frac{Nombre\ total\ de\ bits\ avant\ codage}{Nombre\ total\ de\ bits\ après\ codage}$$

- Nombre moyen de bits par pixel après codage
- Élément de théorie de l'information

- Entropie H(S) de la source dans le cas d'un codage statistique (sans perte)

$H(S) = \sum_{i=1}^{n} p(S_i) I(S_i)$ en bits/ symbole

Démonstration de la matrice inverse de JPEG 2000

On a comme matrice suivante :

$$\begin{pmatrix} Y \\ C_b \\ C_r \end{pmatrix} = \begin{pmatrix} 0{,}299 & 0{,}587 & 0{,}114 \\ -0{,}16875 & -0{,}33126 & 0{,}5 \\ 0{,}5 & -0{,}41869 & -0{,}08131 \end{pmatrix} \begin{pmatrix} R \\ G \\ B \end{pmatrix}$$

Matrice inverse (objectif à arriver) :

$$\begin{pmatrix} R \\ G \\ B \end{pmatrix} = \begin{pmatrix} 1{,}0935 & -0{,}000 & 1{,}5331 \\ 0{,}9524 & -034413 & 1{,}5331 \\ 1{,}000 & 1{,}7720 & -0{,}7809 \end{pmatrix} \begin{pmatrix} Y \\ C_b \\ C_r \end{pmatrix}$$

Il y a matrice inverse si le déterminant de la matrice A est différent à 0. (det(A)≠0).

On peut écrire que CA=AC=I avec I = $\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$.

On va démontrer que nous pouvons inverser cette matrice. J'utilise la méthode de SARRUS.

$$\begin{vmatrix} 0{,}299 & 0{,}587 & 0{,}114 \\ -0{,}16875 & -0{,}33126 & 0{,}5 \\ 0{,}5 & -0{,}41869 & -0{,}08131 \end{vmatrix} \begin{matrix} 0{,}299 & 0{,}587 \\ -0{,}16875 & -0{,}33126 \\ 0{,}5 & -0{,}41869 \end{matrix}$$

= + 0,299*-0,33126*0,08131 + 0,587*0,5*0,5+0,114*-0,1687*-0,41869 – 0,114*-0,33126*0,5-0,299*0,5*-0,41869 – 0,587*-0,1687*0,08131 ≠ 0

On vient de justifier que la matrice A est possible de l'inverser.
Pour assouplir l'écriture, je mets des lettres à la place des décimales.
On a comme matrice équivalente :

$$\begin{pmatrix} Y \\ C_b \\ C_r \end{pmatrix} = \begin{pmatrix} 0{,}299 & 0{,}587 & 0{,}114 \\ -0{,}16875 & -0{,}33126 & 0{,}5 \\ 0{,}5 & -0{,}41869 & -0{,}08131 \end{pmatrix} \begin{pmatrix} R \\ G \\ B \end{pmatrix}$$

$$\begin{pmatrix} Y \\ C_b \\ C_r \end{pmatrix} = \begin{pmatrix} a_1 & a_2 & a_3 \\ a_4 & a_5 & a_6 \\ a_7 & a_8 & a_9 \end{pmatrix} \begin{pmatrix} R \\ G \\ B \end{pmatrix}$$

$$\begin{cases} a_1R + a_2G + a_3B = Y & L_1 \\ a_4R + a_5G + a_6B = C_b & L_2 \\ a_7R + a_8G + a_9B = C_r & L_3 \end{cases}$$

$$\begin{cases} a_1R + a_2G + a_3B = Y & L_1 \leftarrow L_1 \\ 0R + (a_1a_5 - a_2a_4)G + (a_1a_6 - a_3a_4)B = -a_4Y + a_1C_b & L_2 \leftarrow a_1L_2 - a_4L_1 \\ 0R + (a_1a_8 - a_2a_7)G + (a_1a_9 - a_3a_7)B = -a_7Y + aC_r & L_3 \leftarrow a_1L_3 - a_7L_1 \end{cases}$$

$$\begin{cases} a_1R + a_2G + a_3B = Y & L_1 \leftarrow L_1 \\ (a_1a_5 - a_2a_4)G + (a_1a_6 - a_3a_4)B = -a_4Y + a_1C_b & L_2 \leftarrow L_2 \\ 0G + [(a_1a_5 - a_2a_4)(a_1a_9 - a_3a_7) - (a_1a_8 - a_2a_7)(a_1a_6 - a_3a_4)]B = & \\ [-a_7(a_1a_5 - a_2a_4) - (a_1a_8 - a_2a_7)(-a_4)]Y - a_1(a_1a_8 - a_2a_7)C_b + a_1(a_1a_5 - a_2a_4)C_r & \\ L_3 \leftarrow (a_1a_5 - a_2a_4)L_3 - (a_1a_8 - a_2a_7)L_2 & \end{cases}$$

On prend L_3 et on recherche B :

$$B = \frac{[-a_7(a_1a_5 - a_2a_4) - (a_1a_8 - a_2a_7)(-a_4)]}{[(a_1a_5 - a_2a_4)(a_1a_9 - a_3a_7) - (a_1a_8 - a_2a_7)(a_1a_6 - a_3a_4)]}Y - \frac{a_1(a_1a_8 - a_2a_7)}{[(a_1a_5 - a_2a_4)(a_1a_9 - a_3a_7) - (a_1a_8 - a_2a_7)(a_1a_6 - a_3a_4)]}C_b + \frac{a_1(a_1a_5 - a_2a_4)}{[(a_1a_5 - a_2a_4)(a_1a_9 - a_3a_7) - (a_1a_8 - a_2a_7)(a_1a_6 - a_3a_4)]}C_r$$

$$G = \frac{-a_4Y + a_1C_b - (a_1a_6 - a_3a_4)B}{(a_1a_5 - a_2a_4)}$$

$$R = \frac{Y - a_3B - a_2G}{a_1R}$$

On remplace les a_i par les valeurs de la matrice.

On écrit :

B = 1Y+1,772C_b-0,7809C_r

G=0,9524Y-0,34413C_b+1,5331C_r

R=1 ,0935Y-0C_b+1,5331C_r

On retrouve bien la matrice du début que nous devons retrouver :

$$\begin{pmatrix} R \\ G \\ B \end{pmatrix} = \begin{pmatrix} 1{,}0935 & -0{,}000 & 1{,}5331 \\ 0{,}9524 & -034413 & 1{,}5331 \\ 1{,}000 & 1{,}7720 & -0{,}7809 \end{pmatrix} \begin{pmatrix} Y \\ C_b \\ C_r \end{pmatrix}$$

(B)

REFERENCES

Chapitre 1

Livres et poly :

Cours d'Algorithme de Sébastien ROHAUT

Cours ESEO de base du binaire

Cours ESEO compression image

Poly de CALEANDRE du conservatoire National des arts et métiers.

Algorithme et programmation en java,Vincnt Granet, 4 édition, 2014, DUNOD

Références norme Unicode : www.unicode.org/versions/Unicode8.0.0/

Exemple traitement d'image en python : http://www.scipy-lectures.org/advanced/image_processing/index.html