# *Chapitre 3*

# Codage de l'information

## 3.1. Vocabulaire

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

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

L'octet (en anglais *byte* ou B majuscule dans les notations) est une unité d'information composée de 8 bits. Il permet par exemple de stocker un caractère comme une lettre ou un chiffre. Une unité d'information composée de 16 bits est généralement appelée mot (en anglais *word*). Une unité d'information de 32 bits de longueur est appelée mot double (en anglais *double word*, *dword*).

Beaucoup d'informaticiens ont appris que 1 kilooctet valait 1024 octets, mais en décembre 1998, l'organisme international *IEC* a statué sur la question[1].

Voici les unités standardisées :

- Un kilooctet (Ko) = $10^3$ octets
- Un mégaoctet (Mo) = $10^6$ octets
- Un gigaoctet (Go) = $10^9$ octets
- Un téraoctet (To) = $10^{12}$ octets
- Un pétaoctet (Po) = $10^{15}$ octets
- Un exaoctet (Eo) = $10^{18}$ octets
- Un zettaoctet (Zo) = $10^{21}$ octets
- Un yottaoctet (Yo) = $10^{24}$ octets
- Un ronnaoctet[2] (Ro) = $10^{27}$ octets
- Un quettaoctet (Qo) = $10^{30}$ octets

### Ordres de grandeur

| | | | |
|---|---|---|---|
| Un fichier texte | 50 Ko | Une disquette | 1.4 Mo |
| Une image pour le web | 30 Ko | Un CD | 700 Mo |
| Une musique (mp3) | 4 Mo | Un DVD | 4.7 Go |
| Une photo | 6 Mo | Un Blu-ray | 25 Go |
| Un film | 700 Mo à 2 Go | Une clé USB | 8 Go à 256 Go |
| | | Un disque dur | 500 Go à 4 To |

1 http://physics.nist.gov/cuu/Units/binary.html
2 Les préfixes ronna- et quetta- ont été ajoutés en 2022

La taille des documents de type traitement de texte et tableur se compte généralement en Ko quand ils ne contiennent pas d'image. Le texte prend très peu de place. Par exemple, une page de texte sans image dans Word 2007 pèsera 16 Ko. Le même document avec 10 pages de texte pèsera quant à lui 24 Ko. Un document de 1000 pages sans image pèsera donc moins de 1 Mo.

La taille des images va dépendre de leur résolution. Pour faire simple, les photos prises avec des téléphones ou des appareils photos numériques pèseront entre 2 Mo et 10 Mo en format compressé. Les photos non compressées pourront atteindre des tailles bien plus conséquentes. Elles sont réservées à une utilisation professionnelle.

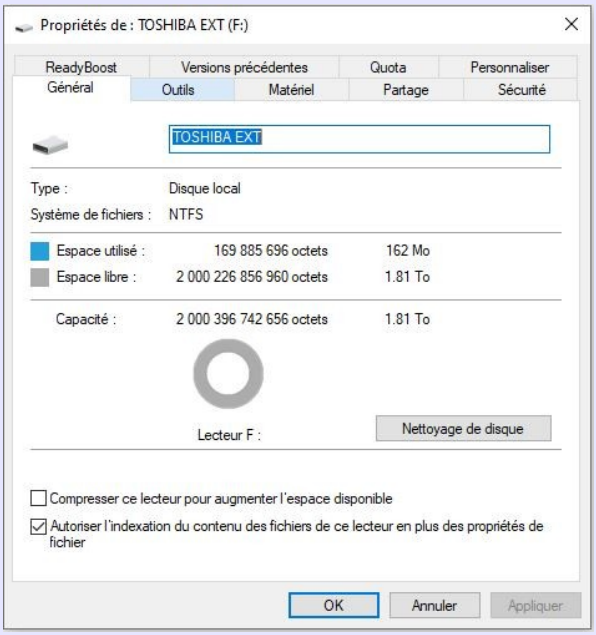
La taille des fichiers de musique compressés au format mp3 est de l'ordre de 1 Mo pour une minute. Il faudra multiplier ce chiffre par 10 si vous souhaitez avoir la version brute du morceau original. Par exemple, si vous souhaitez convertir un CD qui durerait 70 minutes vous obtiendrez des fichiers dont la taille totale sera à peu près 70 Mo. Autant dire que vous pouvez mettre beaucoup de morceaux de musique sur une clé USB…

La taille des vidéos va également dépendre de sa définition, mais aussi de son format d'enregistrement et de sa compression. Difficile de donner une règle car beaucoup de facteurs entrent en jeu.

Pour éviter les confusions des *préfixes binaires*, basés sur les puissances de 2 plutôt que de 10, ont été introduits. Leurs noms sont inspirés des préfixes standards, mais leur seconde syllabe a été remplacée par *bi* pour indiquer leur caractère binaire :

1 kibioctet = $2^{10}$
1 mébioctet = $2^{20}$
1 gibioctet = $2^{30}$
1 tébioctet = $2^{40}$
…

Avez-vous déjà acheté un disque dur et constaté, en l'utilisant pour la première fois, que sa taille réelle était sensiblement plus petite que celle annoncée par le fabricant ?
Lors du développement des premiers ordinateurs, les informaticiens avaient décidé d'utiliser le préfixe « kilo » pour désigner 1024 ($2^{10}$), ce qui est raisonnablement proche de 1000. Cette tendance s'est poursuivie ensuite : un groupe de 1024 kilooctets a été appelé un mégaoctet, un groupe de 1024 mégaoctets a été appelé gigaoctets, et ainsi de suite. Alors que le passage successif entre les préfixes kilo, méga, téra, …, correspond en principe à un facteur 1000, il correspondait donc à un facteur 1024 en informatique. Un mégaoctet devait en principe valoir 1000 x 1000 octets, c'est-à-dire 1'000'000 d'octets, mais il valait 1024 x 1024 octets en informatique, c'est-à-dire 1'048'576 octets... ce qui correspond à une différence de 4.63 % !
Et plus la quantité d'octets augmente, plus la différence est grande. Ainsi, un disque dur de 1 **téra**octet ne peut en réalité contenir que 0,91 **tébi**octet. C'est votre ordinateur qui se trompe en parlant de kilo-, méga-, giga-, téraoctets là où il devrait parler de kibi-, mébi-, gibi-, tébioctets. Malheureusement, ces préfixes binaires ont encore du mal à s'imposer et commencent seulement à être utilisés par certains systèmes d'exploitation.



# 3.2. Les bases décimale, binaire et hexadécimale

Nous utilisons le système décimal (base 10) dans nos activités quotidiennes. Ce système est basé sur dix symboles, de 0 à 9, avec une unité supérieure (dizaine, centaine, etc.) à chaque fois que dix unités sont comptabilisées. C'est un système positionnel, c'est-à-dire que l'endroit où se trouve le symbole définit sa valeur. Ainsi, le 2 de 523 n'a pas la même valeur que le 2 de 132. En fait, 523 est « l'abréviation » de $5 \cdot 10^2 + 2 \cdot 10^1 + 3 \cdot 10^0$. On peut selon ce principe imaginer une infinité de systèmes numériques fondés sur des bases différentes.

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

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

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

| Décimal | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Binaire | 0 | 1 | 10 | 11 | 100 | 101 | 110 | 111 | 1000 | 1001 | 1010 | 1011 | 1100 | 1101 | 1110 | 1111 |
| Hexadécimal | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F |

En base 2, les quatre opérations de base s'effectuent de la même façon qu'en base 10.

Calculez :

110 + 11
110 – 11
110 x 11
110 ÷ 11

## 3.2.1. Conversion décimal - binaire

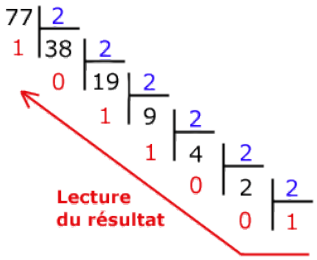
Convertissons 01001101 en décimal à l'aide du schéma ci-dessous :

| $2^7$ | $2^6$ | $2^5$ | $2^4$ | $2^3$ | $2^2$ | $2^1$ | $2^0$ |
|---|---|---|---|---|---|---|---|
| 0 | 1 | 0 | 0 | 1 | 1 | 0 | 1 |

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

Allons maintenant dans l'autre sens et écrivons 77 en base 2. Il s'agit de faire une suite de divisions euclidiennes par 2. Le résultat sera la juxtaposition des restes.

Le schéma ci-contre explique la méthode mieux qu'un long discours. On s'arrête quand on obtient un quotient inférieur à 2.




77 s'écrit donc en base 2 : 1001101.

Canne de hockey

## 3.2.2. Conversion hexadécimal - binaire

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

| Binaire | 0100 | 1101 |
|---|---|---|
| Pseudo-décimal | 4 | 13 |
| Hexadécimal | 4 | D |

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

### Exercice 3.1

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

**Exercice 3.2**

Complétez les lignes du tableau ci-dessous.

| Bases | | |
|---|---|---|
| 2 | 10 | 16 |
| 1001010110 | | |
| | 2002 | |
| | | A1C4 |

**Exercice 3.3***

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

# 3.3. Représentation des nombres entiers

## 3.3.1. Représentation d'un entier naturel

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

**Exemples :** $9 = 00001001_2$, $128 = 10000000_2$, etc.

## 3.3.2. Représentation d'un entier relatif

Un entier relatif est un entier pouvant être négatif. Il faut donc coder le nombre de telle façon que l'on puisse savoir s'il s'agit d'un nombre positif ou d'un nombre négatif, et il faut de plus que les règles d'addition soient conservées.

### Première approche naïve (SPOILER : cela ne marche pas!)

Une représentation naïve pourrait utiliser le bit de poids fort comme marqueur du signe, les autres bits donnant une valeur absolue.

Dans les exemples ci-après, le bit de signe est représenté en bleu ciel.

| Notation naïve | Décimal |
|---|---|
| 00000010 | + 2 |
| 10000010 | − 2 |

Cette représentation possède deux inconvénients. Le premier (mineur) est que le nombre zéro possède deux représentations : 00000000 et 10000000 sont respectivement égaux à +0 et −0.

L'autre inconvénient (majeur) est que cette représentation imposerait de modifier l'algorithme d'addition, car si un des nombres est négatif, l'addition binaire usuelle donne un résultat incorrect. Ainsi :

En base 2 :

0 + 0 = 0
0 + 1 = 1
1 + 0 = 1
1 + 1 = 10

| Addition en notation naïve | Décimal |
|---|---|
| 00000011 | 3 |
| +10000110 | + (−6) |
| =10001001 | = −9 au lieu de −3 |

## La bonne idée

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

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

John **von Neumann** a suggéré l'utilisation de la représentation binaire par complément à deux dans son premier projet de rapport sur la proposition EDVAC de 1945 d'un ordinateur numérique électronique à programme enregistré.
Le premier mini-ordinateur, le PDP-8 introduit en 1965, utilise l'arithmétique du complément à deux, tout comme le Data General Nova de 1969, le PDP-11 de 1970 et presque tous les mini-ordinateurs et micro-ordinateurs ultérieurs.

- **Un entier relatif négatif** sera représenté grâce au codage en complément à deux. Cette représentation permet d'effectuer les opérations arithmétiques usuelles naturellement.

## Principe du complément à deux

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

## Exemple

On désire coder la valeur −19 sur 8 bits. Il faut donc :
1. écrire 19 en binaire : 00010011
2. écrire son complément à 1 : 11101100
3. ajouter 1 : 11101101

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

On construit cette table en partant de la dernière ligne : 10000000. La ligne au-dessus est obtenue en ajoutant 1. On continue ainsi jusqu'à 01111111.

| bit de signe | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | = | 127 |
| 0 | | | | … | | | | = | … |
| 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | = | 2 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | = | 1 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | = | 0 |
| 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | = | −1 |
| 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | = | −2 |
| 1 | | | | … | | | | = | … |
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | = | −127 |
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | = | −128 |

Représentation en complément à deux sur 8 bits

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

Vérifions que l'addition fonctionne correctement :

| Addition en complément à 2 | Décimal | |
|---|---|---|
| 00000011 | 3 | |
| +11111010 | + (−6) | |
| =11111101 | = −3 | ça marche ! |

## Astuce

En appliquant une deuxième fois cette astuce, on retrouve le nombre de départ. Cette observation sera utile pour l'exercice 3.4 b.

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

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

**Cas particuliers :** $\underbrace{100...00}_{n\,bits} = -2^{n-1}$ avec $n$ = 8, 16, 32 ou 64.

L'astuce ne marche pas avec les cas particuliers...

Par exemple : 10000000 = −128.

## Exercice 3.4

**a.** Codez en complément à deux les entiers relatifs suivants, sur 8 bits ou 16 si nécessaire :
456, −1, −56, −5642.

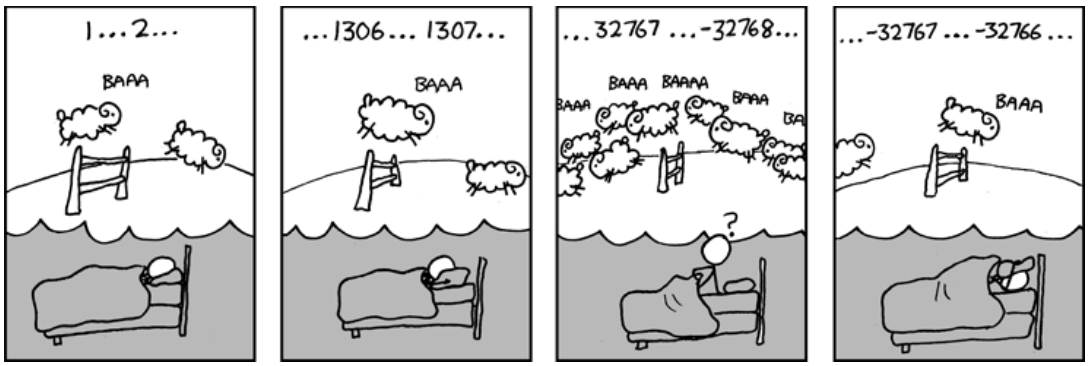
**b.** Traduisez en base dix ces trois entiers relatifs codés en complément à deux :
01101100
11101101
1010101010101010

Pour en savoir plus, lire sur Wikipédia l'article « Vol 501 d'Ariane 5 ».

> Le 4 juin 1996, une fusée Ariane 5 a explosé 37 secondes après le décollage, lors de son vol inaugural. La fusée et son chargement avaient coûté la bagatelle de 500 millions de dollars. Ce désastre vient d'une seule petite variable : celle allouée à l'accélération horizontale. En effet, l'accélération horizontale maximum produite par Ariane 4 donnait une valeur décimale d'environ 64. La valeur d'accélération horizontale de la fusée était codée sur 8 bits, un nombre suffisant pour coder la valeur 64. Mais Ariane 5 était bien plus puissante et brutale : son accélération pouvait atteindre la valeur 300, qui donne 100101100 en binaire et nécessite donc 9 bits. Ainsi, la variable codée sur 8 bits a connu un dépassement de capacité. Il en a résulté une valeur absurde et, par effet domino, le système d'autodestruction préventive de la fusée fut enclenché.

## Exercice 3.5

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



**Exercice 3.6**

Certains logiciels utilisent la représentation POSIX du temps, dans laquelle le temps est représenté comme un nombre de secondes écoulées depuis le 1er janvier 1970 à minuit (0 heure). Sur les ordinateurs 32 bits, la plupart des systèmes d'exploitation représentent ce nombre comme un **nombre entier signé de 32 bits**.

a) Quel est le nombre de secondes maximum que l'on peut représenter ?
b) À quelle date cela correspond-il (jour, mois, année, heures, minutes, secondes).
*Indication :*
Afin de tenir compte des années bissextiles, comptez par cycles de 4 ans composés de $4 \cdot 365 + 1 = 1461$ jours ;
c) Que se passera-t-il une seconde plus tard ? Quel sera le nombre de secondes affiché (en base 10) ? À quelle date cela correspond-il ?

L'an 2000 est une année bissextile.

# 3.4. Représentation des nombres réels

Rappel de math :

$$a^{-n} = \frac{1}{a^n}$$

En base 10, l'expression 652,375 est une manière abrégée d'écrire :

$$6 \cdot 10^2 + 5 \cdot 10^1 + 2 \cdot 10^0 + 3 \cdot 10^{-1} + 7 \cdot 10^{-2} + 5 \cdot 10^{-3}.$$

Il en va de même pour la base 2. Ainsi, l'expression 110,101 signifie :

$$1 \cdot 2^2 + 1 \cdot 2^1 + 0 \cdot 2^0 + 1 \cdot 2^{-1} + 0 \cdot 2^{-2} + 1 \cdot 2^{-3}.$$

## 3.4.1. Conversion de binaire en décimal

On peut ainsi facilement convertir un nombre réel de la base 2 vers la base 10. Par exemple :

$$110{,}101_2 = 1 \cdot 2^2 + 1 \cdot 2^1 + 0 \cdot 2^0 + 1 \cdot 2^{-1} + 0 \cdot 2^{-2} + 1 \cdot 2^{-3} = 4 + 2 + 0{,}5 + 0{,}125 = 6{,}625_{10}.$$

**Exercice 3.7**

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

## 3.4.2. Conversion de décimal en binaire

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

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

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

On continue ainsi jusqu'à la précision désirée...
**Attention !** Un nombre à développement décimal fini en base 10 ne l'est pas forcément en base 2.
Cela peut engendrer de mauvaises surprises. Prenons par exemple ce programme Python :

```
i=0.0
while i<1:
    print(i)
    i+=0.1
```

Le problème vient du fait que l'on n'arrive pas à représenter exactement 0.3 en binaire (voir l'exercice 3.8 a)

| Résultat attendu | Résultat réel |
|---|---|
| 0.0 | 0.0 |
| 0.1 | 0.1 |
| 0.2 | 0.2 |
| 0.3 | 0.30000000000000004 |
| 0.4 | 0.4 |
| 0.5 | 0.5 |
| 0.6 | 0.6 |
| 0.7 | 0.7 |
| 0.8 | 0.7999999999999999 |
| 0.9 | 0.8999999999999999 |
| | 0.9999999999999990 |

**Exercice 3.8a**

Transformez en base 2 : 0,3 ; 0,5625 ; –0,15 ; 12,9.

## 3.4.2. La norme IEEE 754 (32 bits)

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

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

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

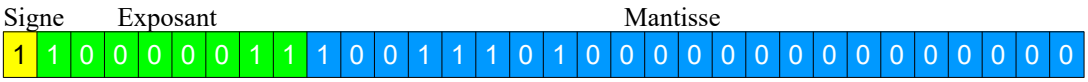
Certaines conditions sont à respecter pour les exposants :

- l'exposant 00000000 est interdit, sauf pour écrire le nombre 0 ;
- l'exposant 11111111 est interdit : on s'en sert toutefois pour signaler des erreurs, on appelle alors cette configuration du nombre NaN, ce qui signifie « Not a number » ;
- il faut ajouter à l'exposant un *biais*, fixé à $2^7–1 = 127$ (01111111).

### Exemple

Traduisons en binaire, en utilisant la norme IEEE 754, le nombre –25,8125.

- Codons d'abord la valeur absolue en binaire : $25{,}8125_{10} = 11001{,}1101_2$
- Mettons ce nombre sous la forme : **1, *mantisse***
  11001,1101 = **1,10011101**$\cdot 2^4$ ($2^4$ décale la virgule de **4** chiffres vers la droite).
- La **mantisse**, étendue sur 23 bits, est **100 1110 1000 0000 0000 0000.**
- **Exposant** = 127 + **4** = $131_{10}$ = $1000\ 0011_2$
- **Signe** = 1, ce qui indique un nombre négatif.




En binaire : 11000001 11001110 10000000 00000000
En hexadécimal : C1 CE 80 00.

Il est à noter que l'on n'écrit pas le 1 devant la virgule, car il existe forcément, sauf pour le nombre 0, qui est un cas particulier. Il y a d'ailleurs deux 0 : un négatif (1 suivi de 31 zéros) et un positif (32 zéros).

Précisons qu'il existe aussi une norme IEEE 754 pour représenter les nombres rationnels sur 64 bits, ce qui permet une plus grande précision. Le principe est le même, mais l'exposant est codé sur 11 bits, le biais est de $2^{10}–1 = 1023$ et la mantisse est codée sur 52 bits.

**Exercice 3.8b**

Écrivez les nombres en base 2 de l'exercice 3.8a sur 32 bits en respectant la norme IEEE 754.

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

# 3.5. Le code ASCII

ASCII : *American Standard Code for Information Interchange*

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

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

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

Le code ASCII a été mis au point pour la langue anglaise, il ne contient donc pas de caractères accentués, ni de caractères spécifiques à une langue. Le code ASCII a donc été étendu à 8 bits pour pouvoir coder plus de caractères (on parle d'ailleurs de code ASCII étendu...).

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

Cette norme s'appelle ISO-8859 et se décline par exemple en ISO-8859-1 lorsqu'elle étend l'ASCII avec les caractères accentués d'Europe occidentale, et qui est souvent appelée Latin-1 ou Europe occidentale.

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

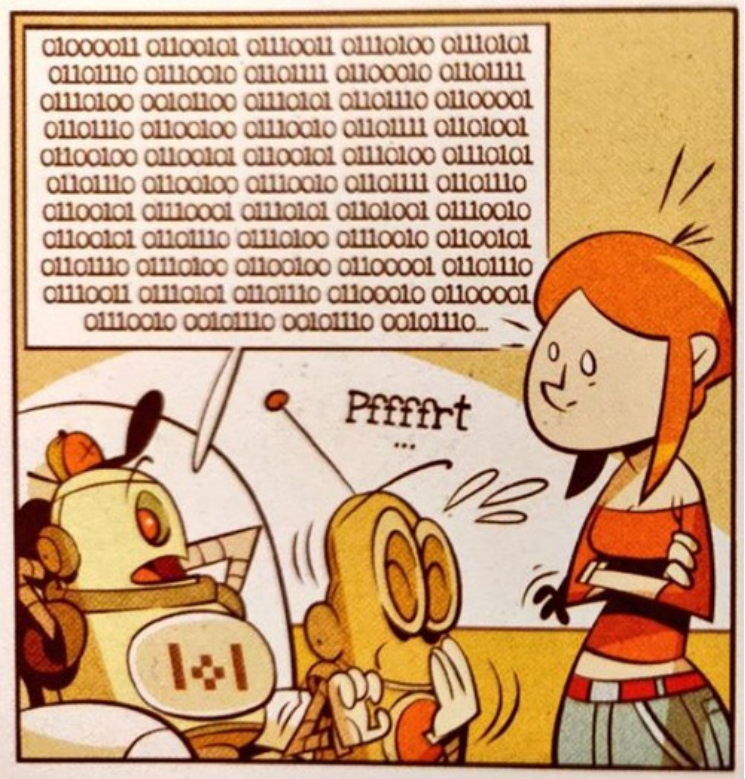
UTF-8 (abréviation de l'anglais *Universal Character Set Transformation Format* 1 - 8 bits) est un codage de caractères informatiques conçu pour coder l'ensemble des caractères du « répertoire universel de caractères codés », aujourd'hui totalement compatible avec le standard Unicode, en restant compatible avec la norme ASCII limitée à l'anglais de base, mais très largement répandue depuis des décennies.

Par sa nature, UTF-8 est d'un usage de plus en plus courant sur Internet, et dans les systèmes devant échanger de l'information. Il s'agit également du codage le plus utilisé dans les systèmes GNU, Linux et compatibles pour gérer le plus simplement possible des textes et leurs traductions dans tous les systèmes d'écritures et tous les alphabets du monde.

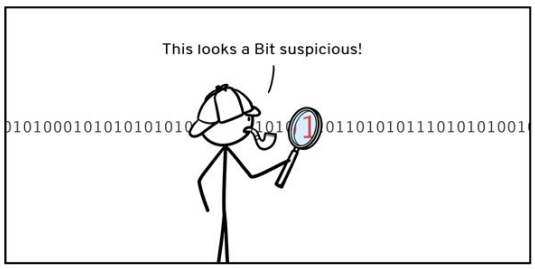
> Toutes ces normes différentes et leurs incompatibilités partielles sont la cause des problèmes que l'on rencontre parfois avec les caractères accentués. C'est pour cette raison qu'il vaut mieux, quand on écrit des courriels à l'étranger, n'utiliser que des caractères non accentués.

## Exercice 3.9

Traduisez cette blague (*Rob Niveau 3*, James, Boris Miroir, éd. Delcourt, p. 56)



# 3.6. Codes détecteurs/correcteurs d'erreurs




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

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

### Pourquoi ces codes ?

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

Richard **Hamming**
(1915-1998)

## 3.6.1. La distance de Hamming

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

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

Il est souhaitable d'avoir une certaine distance entre les mots envoyés, afin de détecter s'il y a eu une erreur de transmission.

Par exemple, si l'on a trois messages de trois bits à transmettre, il vaut mieux choisir les codages qui sont à distance 2 les uns des autres, par exemple 000, 110 et 101. En effet, si un seul bit est altéré, on recevra un message inconnu, ce qui indiquera une erreur de transmission.

Par contre, en utilisant 000, 001 et 010, une erreur pourrait passer inaperçue. En effet, 001 pourrait être en fait le message 000 dont le dernier bit a été altéré...

## 3.6.2. Redondance

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

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

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

### Principe général

- Chaque suite de bits à transmettre est augmentée par une autre suite de bits dite « de redondance » ou « de contrôle ».

- Pour chaque suite de $k$ bits transmise, on ajoute $r$ bits. On dit alors que l'on utilise un code $C(n, k)$ avec $n = k + r$ .
- À la réception, les bits ajoutés permettent d'effectuer des contrôles.

### Bit de parité

Transmettons sept bits auxquels viendra s'ajouter un bit de parité. On peut définir le bit de parité comme étant égal à zéro si le nombre de 1 est pair et à un dans le cas contraire.

Si on reçoit un octe avec un nombre impair de 1, c'est qu'il y a eu une erreur de transmission.

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

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

## 3.6.3. Somme de contrôle

La somme de contrôle (en anglais « checksum ») est un cas particulier de contrôle par redondance. Elle permet de détecter les erreurs, mais pas forcément de les corriger.

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

### 3.6.3.1. Le code ISBN

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

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

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

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

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

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

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

## Exemple

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

**La question qui tue :** pourquoi est-il indispensable de donner une pondération à chaque position ?

| Code ISBN | 2 | 9 | 4 | 0 | 0 | 4 | 3 | 4 | 1 |
|---|---|---|---|---|---|---|---|---|---|
| Pondérations | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 |
| Produits | 20 | 81 | 32 | 0 | 0 | 20 | 12 | 12 | 2 |

La somme des produits est **179**, dont le reste de la division euclidienne par 11 est **3**. La clé de contrôle est donc 11 − 3 = **8**. L'ISBN au complet est : 2-940043-41-**8**.

La vérification de la clé complète à 10 chiffres donne la somme pondérée 179 + 8 = 187, qui est bien un multiple de 11.

## Depuis le 1er janvier 2007, les ISBN sont passés à 13 chiffres

**Pourquoi une réforme du système ISBN ?**

Les codes **EAN-13** (European Article Numbering à 13 chiffres) sont les codes-barres utilisés dans le monde entier sur l'ensemble des produits de grande consommation.

- Pour augmenter la capacité de numérotation du système ISBN qui était devenu insuffisant, notamment en raison de l'augmentation du nombre de publications électroniques.
- Pour rendre l'ISBN complètement compatible avec l'EAN-13 qui sert à la génération des codes-barres :

**Qu'est-ce qui a changé ?**

978 (ou 979) indique que l'objet est un livre.

- On fait précéder l'ISBN du préfixe « 978 » et on recalcule la clé de contrôle.
- L'ISBN à 13 chiffres est identique à l'EAN-13 servant à la génération du code-barres et figurant sous celui-ci.
- Le préfixe « 979 » sera introduit quand les segments ISBN existants seront épuisés.
- Les segments identifiant les éditeurs ne demeureront pas les mêmes lorsque le préfixe passera en 979.
- Depuis le 1er janvier 2007, l'ISBN à 13 chiffres est utilisé pour toutes les commandes et toutes les transactions commerciales, manuelles ou électroniques.
- Depuis le 1er janvier 2007, les codes-barres sont surmontés de l'ISBN à 13 chiffres segmenté (avec des tirets) alors que l'EAN-13 correspondant est indiqué en dessous des codes à barres (sans tiret ni espace).

## Algorithme permettant de générer l'EAN Bookland à partir de l'ISBN

1. Garder les 9 premiers chiffres de l'ISBN.
2. Ajouter le préfixe Bookland de l'EAN.
3. Entrer/lire les facteurs de pondération constants associés à chaque position de l'EAN (1 3 1 3 1 3 1 3 1 3 1 3).
4. Multiplier chaque chiffre par le facteur de pondération qui lui est associé.
5. Diviser la somme des produits par 10 et garder le reste.
6. Si le reste vaut 0, le numéro de contrôle est aussi 0. Sinon, le numéro de contrôle est 10 moins le reste trouvé. Ajouter ce suffixe.

Convertissons l'ISBN 0-8436-1072-7 :

| ISBN (9 premiers chiffres) | | | | 0 | 8 | 4 | 3 | 6 | 1 | 0 | 7 | 2 | ~~7~~ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Ajout du préfixe 978 | 9 | 7 | 8 | 0 | 8 | 4 | 3 | 6 | 1 | 0 | 7 | 2 | |
| Facteurs | 1 | 3 | 1 | 3 | 1 | 3 | 1 | 3 | 1 | 3 | 1 | 3 | |
| Produits | 9 | 21 | 8 | 0 | 8 | 12 | 3 | 18 | 1 | 0 | 7 | 6 | |

Somme des produits : 9 + 21 + 8 + 0 + 8 + 12 + 3 + 18 + 1 + 7 + 6 = 93.
Le reste de la division de 93 par 10 est 3. Le chiffre de contrôle est donc 10 – 3 = **7**.
Le code EAN-ISBN est donc : 978-0-8436-1072-**7**.

## Exercice 3.10

Quel est le numéro de contrôle du code ISBN à 9 chiffres 2-35288-041 ?
Donnez le code EAN Bookland à partir de l'ISBN ci-dessus.

### 3.6.3.2. Formule de Luhn

La formule de Luhn, aussi connue comme l'algorithme « modulo 10 », est une simple formule de somme de contrôle utilisée pour valider une variété de numéros de comptes, comme les numéros de cartes bancaires, les numéros d'assurance sociale canadiens, les numéros IMEI des téléphones portables, etc. Elle est aussi utilisé dans l'immatriculation du matériel ferroviaire.

Hans Peter **Luhn** (1896-1964)

Elle fut développée dans les années 1960 par un ingénieur allemand chez IBM, Hans Peter **Luhn**, comme méthode de validation d'identification de nombres. Sa notoriété provient principalement de son adoption par les compagnies de cartes de crédit peu après sa création.

L'algorithme fait partie du domaine public et est largement répandu aujourd'hui. Cette formule génère un chiffre de vérification, qui est généralement annexé à un numéro d'identité partiel pour générer un identifiant complet.

Cet identifiant (numéro partiel + chiffre de vérification) est soumis à l'algorithme suivant pour vérifier sa validité :

1. On alterne les facteurs de pondération 1 et 2 sous les chiffres du code, en les écrivant de droite à gauche.
2. Chaque chiffre du code est multiplié par son facteur de pondération. Si un chiffre multiplié par 2 est plus grand que 9 (comme c'est le cas par exemple pour 8 qui devient 16), alors il faut le ramener à un chiffre entre 1 et 9 en soustrayant 9.
3. On effectue la somme de tous les produits obtenus.
4. Si cette somme est un multiple de 10, alors le nombre original est valide.

### Exemple

Considérons l'identification du nombre 972-487-086. La première étape consiste à doubler un chiffre sur deux en partant de l'avant-dernier jusqu'au début, et de faire la somme de tous les chiffres, doublés ou non (si un chiffre est supérieur à 9, on retranche 9, d'où la 4ème ligne). Le tableau suivant montre cette étape :

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| Code | 9 | 7 | 2 | 4 | 8 | 7 | 0 | 8 | 6 | |
| Facteurs | 1 | 2 | 1 | 2 | 1 | 2 | 1 | 2 | 1 | |
| Produits | 9 | **14** | 2 | 8 | 8 | **14** | 0 | **16** | 6 | |
| Chiffres entre 0 et 9 | 9 | 5 | 2 | 8 | 8 | 5 | 0 | 7 | 6 | Somme=50 |

La somme, égale à 50, est un multiple de 10, donc le nombre est valide.

## Exercice 3.11

Le code IMEI est un numéro de série de 15 à 17 chiffres permettant d'identifier de manière unique tout appareil mobile.

Vous trouverez ce numéro sur la facture d'achat de l'appareil ou sur sa boîte d'emballage. Si l'appareil est un smartphone, vous pouvez également composer le *#06# afin de l'obtenir.

Vérifiez que le numéro IMEI de votre smartphone satisfait la condition de la formule de Luhn.

## Exercice 3.12

Chaque véhicule ferroviaire dispose d'un numéro d'identification unique le distinguant de tout autre véhicule ferroviaire. La numérotation des « plaques d'immatriculation » des trains a été uniformisée par **l'Union internationale des chemins de fer (UIC) :** chaque locomotive, chaque automotrice et chaque voiture de voyageurs est identifiée par un numéro à douze chiffres.

Le numéro écrit sur la voiture de voyageurs ci-dessus indique (voir [6]) :

- le type de véhicule : 50 = service commercial national
- le pays : 85 = Suisse
- le type de voiture : 3- = voiture mixte 1ère – 2e classe, -9 = neuf compartiments
- la vitesse maximale : 43 = vitesse maximale comprise entre 121 et 140 km/h
- le numéro de série : 829
- le chiffre 3 situé après le tiret est une clé de contrôle pour la vérification informatique.

Vérifiez que ce numéro d'immatriculation satisfait la condition de la formule de Luhn.

## Exercice 3.13

Les numéros figurant sur ces cartes de crédit sont-ils valides ?

### 3.6.3.3. Modulo 10 récursif

Le Modulo 10 récursif est un algorithme utilisé principalement par les banques et la Poste suisse (pour le traitement des bulletins de versement et des QR-factures) pour calculer la clé de contrôle des numéros de compte.

CCP signifie Compte Courant Postal. Pour vérifier qu'un numéro de CCP est valide, on va utiliser le tableau ci-dessous (voir [9]) :

| retenue | **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | contrôle |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 0 | 9 | 4 | 6 | 8 | 2 | 7 | 1 | 3 | 5 | **0** |
| **1** | 9 | 4 | 6 | 8 | 2 | 7 | 1 | 3 | 5 | 0 | **9** |
| **2** | 4 | 6 | 8 | 2 | 7 | 1 | 3 | 5 | 0 | 9 | **8** |
| **3** | 6 | 8 | 2 | 7 | 1 | 3 | 5 | 0 | 9 | 4 | **7** |
| **4** | 8 | 2 | 7 | 1 | 3 | 5 | 0 | 9 | 4 | 6 | **6** |
| **5** | 2 | 7 | 1 | 3 | 5 | 0 | 9 | 4 | 6 | 8 | **5** |
| **6** | 7 | 1 | 3 | 5 | 0 | 9 | 4 | 6 | 8 | 2 | **4** |
| **7** | 1 | 3 | 5 | 0 | 9 | 4 | 6 | 8 | 2 | 7 | **3** |
| **8** | 3 | 5 | 0 | 9 | 4 | 6 | 8 | 2 | 7 | 1 | **2** |
| **9** | 5 | 0 | 9 | 4 | 6 | 8 | 2 | 7 | 1 | 3 | **1** |

La structure mathématique du tableau garantit la détection de 100 % des erreurs de frappe uniques (un faux chiffre) et de la quasi-totalité des inversions de deux chiffres adjacents (ex: saisir *45* au lieu de *54*), qui sont les erreurs humaines les plus fréquentes.

Vérifions le numéro de CCP 17-603303-5.

1. On commence toujours avec une retenue de 0.
2. Le premier chiffre du CCP est 1. Le tableau indique que la nouvelle retenue est 9 (intersection de la ligne « 0 » et de la colonne « 1 »).
3. Avec une retenue de 9 et un deuxième chiffre égal à 7, on trouve que la troisième retenue est 7 (intersection de la ligne « 9 » et de la colonne « 7 »).
4. On continue ainsi jusqu'à l'avant-dernier chiffre du CCP (le dernier chiffre est la clé de contrôle qui se lira dans la colonne « contrôle ») :

Conseil pratique : écrivez d'abord le numéro de CCP (sans le dernier chiffre) dans la seconde colonne.

| **Retenues** | **Chiffres du CCP** |
|---|---|
| 0 | 1 |
| 9 | 7 |
| 7 | 6 |
| 6 | 0 |
| 7 | 3 |
| 0 | 3 |
| 6 | 0 |
| 7 | 3 |
| 0 | 0 |

La dernière retenue est 0. Cela correspond dans la colonne rose à la clé de contrôle 0.

Le dernier chiffre du CCP devrait donc être un 0 et non pas un 5, ce qui signifie que le numéro de CCP est incorrect.

**Exercice 3.14**

Vérifiez le numéro de CCP 10-546323-6.

## 3.6.4. Code de Hamming

Un code de Hamming permet la détection et la correction automatique d'une erreur si elle ne porte que sur un bit du message. Il est basé sur l'idée des bits de parité ; il y en a plusieurs, placés à des endroits précis.

### Structure d'un code de Hamming

- Un code de Hamming est composé de deux parties (mélangées) : le message composé de $m$ bits et les $n$ bits de contrôle de parité. On parle de mot de Hamming.
- La longueur totale d'un mot de Hamming est $2^n-1$. Donc : 7, 15, 31, ...
- La longueur du message est donc $m = (2^n-1) - n$.
- Les bits de contrôle de parité $C_i$ sont en position $2^i$, avec $i = 0, 1, 2, ...$
- Les bits du message $D_j$ occupe le reste du mot.

On parle de code de Hamming $x$–$y$ : $x$ est la longueur totale du mot ($n$+$m$) et $y$ la longueur du message ($m$).

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

*Structure d'un code de Hamming 7–4*

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

### Retrouver l'erreur dans un mot de Hamming

Si les bits de contrôle de parité $C_2$, $C_1$, $C_0$ ont tous la bonne valeur, il n'y a pas d'erreur ; sinon la valeur des bits de contrôle indique la position de l'erreur entre 1 et 7.

Le code de Hamming présenté ici ne permet de retrouver et corriger qu'une erreur.

**Truc :** pour se rappeler quels bits sont vérifiés par chacun des bits de contrôle de parité, on peut construire le tableau ci-dessous :

1 = 001, 2 = 010, 3 = 011, etc.

Attention à l'ordre des bits de contrôle !

| | 7 | 6 | 5 | 4 | 3 | 2 | 1 |
|---|---|---|---|---|---|---|---|
| $C_2$ | 1 | 1 | 1 | 1 | 0 | 0 | 0 |
| $C_1$ | 1 | 1 | 0 | 0 | 1 | 1 | 0 |
| $C_0$ | 1 | 0 | 1 | 0 | 1 | 0 | 1 |

- La première ligne (colorée) indique la position des bits.
- Dans chaque colonne, on écrit le numéro de la position verticalement en binaire.
- Les 1 dans les lignes indiquent alors quels bits sont vérifiés. Ainsi :
  - $C_2$ : la somme des bits 7, 6, 5, 4 doit être paire.
  - $C_1$ : la somme des bits 7, 6, 3, 2 doit être paire.
  - $C_0$ : la somme des bits 7, 5, 3, 1 doit être paire.

## Exemple d'un code de Hamming 7-4

Alice souhaite envoyer à Bob le message 1010. Elle forme le mot de Hamming correspondant :

| 7 | 6 | 5 | 4 | 3 | 2 | 1 |
|---|---|---|---|---|---|---|
| 1 | 0 | 1 | $C_2$ | 0 | $C_1$ | $C_0$ |

$C_2$ vaudra **0** pour rendre pair 1+0+1+**0** (les bits d'indices 7, 6, 5, 4).
$C_1$ vaudra **1** pour rendre pair 1+0+0+**1** (les bits d'indices 7, 6, 3, 2).
$C_0$ vaudra **0** pour rendre pair 1+1+0+**0** (les bits d'indices 7, 5, 3, 1).

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

Le mot de Hamming 7-4 du message 1010 est donc 1010010.

### Détection et correction d'une erreur

Alice envoie ce mot, mais Bob reçoit le 0010010 (le bit de poids fort a été altéré).

$C_2$ a la mauvaise valeur, car 0+0+1+0 est impair. Il y a une erreur en position 7, 6, 5 ou 4.
$C_1$ a la mauvaise valeur, car 0+0+0+1 est impair. Il y a une erreur en position 7, 6, 3 ou 2.
$C_0$ a la mauvaise valeur, car 0+1+0+0 est impair. Il y a une erreur en position 7, 5, 3 ou 1.

C'est donc le bit 7 qui est erroné. Il faudra donc remplacer le 0 par un 1.

**Truc :** Écrivons le nombre binaire $C_2C_1C_0$ où $C_i$ vaut 0 si le bit de contrôle $C_i$ a la bonne valeur et 1 sinon. On obtient ici 111, ce qui correspond à... 7 en binaire. Le nombre binaire $C_2C_1C_0$ correspond au numéro du bit à corriger. Magique !

Que se passe-t-il si c'est un des bits de contrôle qui est altéré ? Imaginons que Bob ait reçu 1010011 (cette fois-ci, c'est le bit de poids faible qui a été altéré).

$C_2$ a la bonne valeur, car 1+0+1+0 est pair. Il n'y a pas d'erreur en position 7, 6, 5 et 4.
$C_1$ a la bonne valeur, car 1+0+0+1 est pair. Il n'y a pas d'erreur en position 7, 6, 3 et 2
$C_0$ a la mauvaise valeur, car 1+1+0+1 est impair. Il y a une erreur en position 7, 5, 3 ou 1.

Ici, $C_2C_1C_0$ vaut 001. Le bit erroné est donc le numéro 1.

## Exercice 3.15

Alice veut envoyer le message 1011. Quels bits de contrôle devez-vous lui adjoindre et quel mot de Hamming transmettrez-vous alors ?

## Exercice 3.16

Bob reçoit ce mot de Hamming 7–4 : 1101101. Y a-t-il une erreur ?

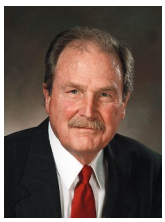
**Exercice 3.17**

Soit un mot de Hamming 15–11 suivant :

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

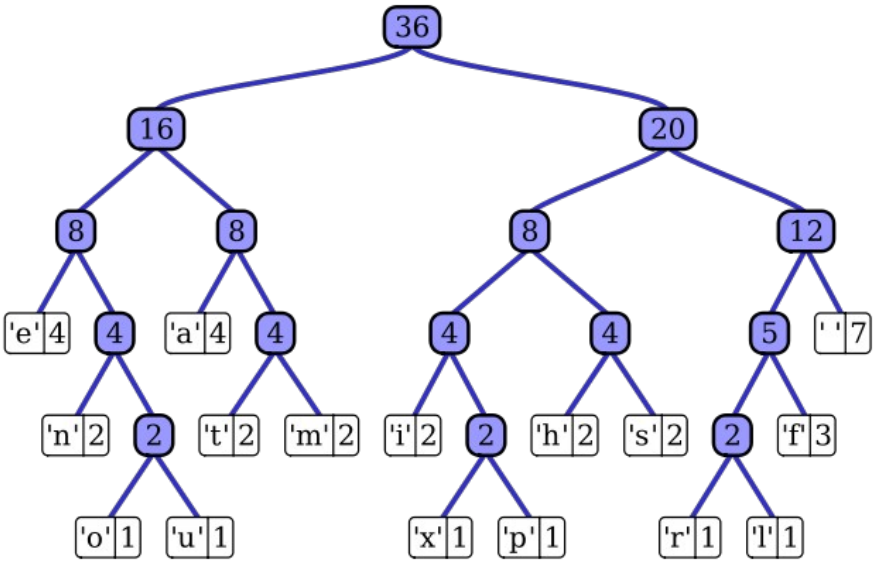
1. Quels sont les bits de contrôle de parité ?
2. Quelles positions contrôlent chacun de ces bits ?
3. Quel est le message reçu ?
4. Est-ce que le message reçu correspond au message transmis ?

# 3.7. Codage de Huffman



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

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




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

## Le principe

Huffman propose de recoder les données qui ont une occurrence très faible sur une longueur binaire supérieure à la moyenne, et recoder les données très fréquentes sur une longueur binaire très courte.

Ainsi, pour les données rares, nous perdons quelques bits regagnés pour les données répétitives. Par exemple, dans un fichier ASCII le « w » apparaissant 10 fois aura un code très long : 010100000100. Ici la perte est de 40 bits (10 x 4 bits), car sans compression, il serait codé sur 8 bits au lieu de 12. Par contre, le caractère le plus fréquent comme le « e » avec 200 apparitions sera codé par 1. Le gain sera de 1400 bits (7 x 200 bits). On comprend l'intérêt d'une telle méthode.

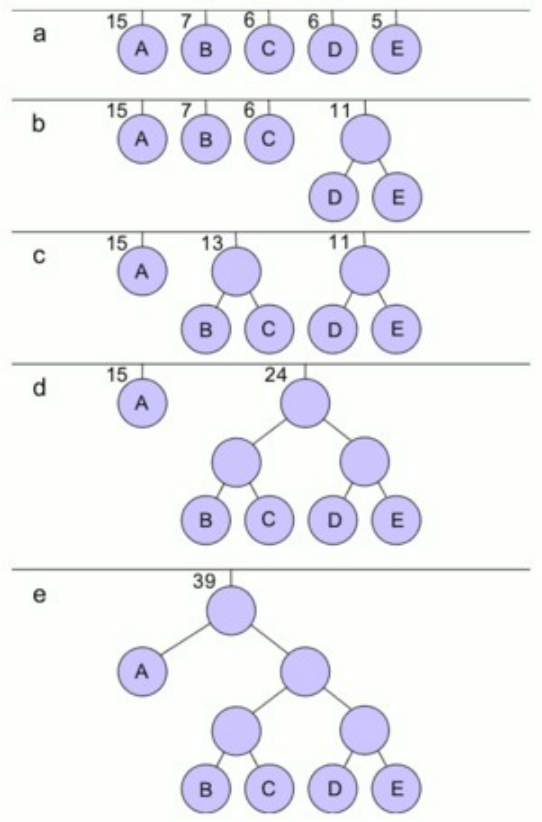
Le codage de Huffman a une propriété de préfixe : une séquence binaire ne peut jamais être à la fois représentative d'un élément codé et constituer le début du code d'un autre élément. Si un

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

## Méthode

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

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




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

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

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

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

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

## Exercice 3.18

Construisez un codage de Huffman du message « CECI EST UN CODAGE DE HUFFMAN. ». Notez qu'il y a plusieurs codages de Huffman possibles, mais la longueur du message codé sera toujours la même. N'oubliez pas les espaces et le point.

Vérifiez la propriété du préfixe.

## Exercice 3.19

De quel roman célèbre est tirée la phrase compressée ci-dessous ?

111110110000110101010110111101000100110101101000111 0

L'arbre de Huffman (malheureusement en partie effacé) est le suivant :

Voici la fréquence d'apparition des lettres :

| a | C | e | h | l (i) | l (L) | m | s | | . |
|---|---|---|---|---|---|---|---|---|---|
| 2 | 1 | 2 | 1 | 1 | 3 | 2 | 1 | 2 | 1 |

# Sources

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

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

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

[4] Wikipédia, « Complément à deux », <https://fr.wikipedia.org/wiki/Complément_à_deux>

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

[6] Wikipédia, « Classification UIC des voitures de chemin de fer » <https://fr.wikipedia.org/wiki/Classification_UIC_des_voitures_de_chemin_de_fer>

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

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

[9] Documentation six-groupe : <https://www.six-group.com/dam/download/banking-services/standardization/qr-bill/ig-qr-bill-v2.2-fr.pdf>