Département
Éducation
et Technologie

• Codage et traitement formel

Etienne Vandeput

5.52

Septembre 1995

Centre pour la Formation à
l'Informatique dans le Secondaire

CODAGE ET TRAITEMENT FORMEL

Codes et algorithmes associés

Qui ne s'est pas adonné, dans sa jeunesse, au cours d'un jeu d'aventure, au codage et au décodage de messages secrets. Messages, en principe illisibles pour l'ennemi et parfaitement compréhensibles des correspondants s'ils disposent d'un dictionnaire des symboles employés ou d'algorithmes de traduction. Ce type d'activité nous amène à considérer qu'**un code est une correspondance (bi)univoque entre deux séries de symboles**. Cette correspondance permet, à deux ou plusieurs interlocuteurs de communiquer via des messages transmis sous une forme qui leur est conventionnelle. Un exemple: nous décidons de nous communiquer des messages secrets. Nous choisissons comme convention que chaque lettre (majuscule) de l'alphabet est codée par son numéro d'ordre (*A* est codé *1*, *B* est codé *2* etc.). Pour transmettre nos messages, il faudra prendre des précautions supplémentaires. Ainsi, pour le récepteur, *12* se traduit-il par les deux symboles *AB* ou par le symbole *L*?

Question: connaissez-vous des moyens d'éviter ces situations de doute?

Voici une suggestion. Nous pouvons établir un dictionnaire de correspondance des symboles qui peut être utilisé, tant par le codeur que par le décodeur. En voici un exemple.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
00	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26

Quelques remarques.

Le choix d'un tel code oblige le décodeur à travailler par groupe de deux symboles. On parle de **code de longueur fixe**. Dans ce cas, des délimiteurs ne sont pas nécessaires. Le message suivant peut être traduit sans ambiguïté.

020905142205142105000100120100061518130120091514000405190016051819151414051900180519191521180305 19

Un délimiteur intervient toutefois au niveau sémantique (il est inutile au codeur, au messager et au décodeur), c'est l'espace. Il est seulement utile à celui qui interprète le message, qui lui donne du sens. Par souci d'économie, ce qui sera le leitmotiv dans la partie concernant la compression des données, on pourrait imaginer s'en passer. Les Romains le faisaient et utilisaient le contexte pour rétablir le texte. Mais on devine les problèmes que cela peut poser.

Un **code de longueur variable** impose au codeur (et forcément au décodeur) l'emploi d'un séparateur formel ou d'un fanion pour indiquer le nombre de symboles à prendre en compte. Un exemple très connu de code de longueur variable est le code Morse utilisant de un à quatre symboles (barres et points) pour la codification des lettres de l'alphabet. Un message écrit en Morse nécessite l'emploi de délimiteurs pour que le décodeur puisse distinguer les groupes de symboles qui sont de longueur variable. Au niveau de la transmission, ce délimiteur se traduit par un "silence".

Ce message est impossible à traduire

s ! ! ! ! ! ! s ! ! ! ! s ! s ! ! ! s !

sans délimiteurs

s ! ! ! / ! ! / ! /s ! / ! ! ! s/ ! /s ! / ! ! s/ !

Alphabet Morse

A	B	C	D	E	F	G	H	I	J	K	L	M
! S	**S ! ! !**	**S ! S !**	**S ! !**	**!**	**! ! S !**	**S S !**	**! ! ! !**	**! !**	**! S S S**	**S ! S**	**! S ! !**	**S S**
N	O	P	Q	R	S	T	U	V	W	X	Y	Z
S !	**S S S**	**! S S !**	**S S ! S**	**! S !**	**! ! !**	**S**	**! ! S**	**! ! ! S**	**! S S**	**S ! ! S**	**S ! S S**	**S S ! !**

Dans un cas comme le précédent, la correspondance fait nécessairement l'objet d'un dictionnaire. On peut éviter l'emploi d'un dictionnaire en fournissant deux algorithmes, l'un permettant de coder et l'autre de décoder le message. En voici un exemple trivial: chaque lettre de l'alphabet est remplacée par la lettre située deux rangs en arrière dans l'ordre alphabétique (E est remplacé par C, M par K etc.). On convient que la suite des lettres est circulaire (Z est suivi de A) pour trouver un remplaçant à A et B (respectivement Y et Z).

ZGCLTCLSC

De tels algorithmes sont assez simples à découvrir en observant la fréquence d'apparition de certaines lettres par exemples. On peut intuitivement en deviner quelques-unes et faire jouer la sémantique pour découvrir le reste des mots. Le C apparaît ici comme une lettre dont la fréquence n'est pas négligeable.

Nous pouvons affiner la définition d'**un code** en précisant qu'il **autorise parfois une certaine sécurité dans la transmission des informations**. Dans ce cas, seuls le codeur et le décodeur sont sensés avoir connaissance de l'algorithme ou du dictionnaire de traduction.

D'autres exemples plus complexes peuvent être inventés. La seule règle à respecter est de s'assurer que la correspondance est unique!

Imaginez quelques codages originaux.

Codification des caractères

Le domaine qui nous intéresse est celui du traitement automatique de l'information. La seule technologie actuellement peu coûteuse nous impose une codification finale binaire, pour les raisons que l'on connaît bien. Les premiers problèmes rencontrés par les constructeurs d'ordinateurs ont été ceux du codage des nombres et des caractères. Intéressons nous d'abord au codage des caractères.

De nombreux codes ont vu le jour. Leur existence reste souvent liée à un constructeur ou à la définition d'un standard par un organisme international. Sans vouloir refaire l'histoire du codage des caractères, voici quelques éléments qui vous permettront sans doute de remettre un peu d'ordre dans les connaissances que vous avez à leur propos.

Quelques notions de base

Il va de soi que la plus petite unité d'information est codée au moyen d'un chiffre binaire (en anglais **bi**nary dig**it** ou bit). Avec deux symboles il n'est possible de coder que deux informations différentes sur une position binaire. C'est pourquoi les ordinateurs ont davantage été conçus pour traiter des mots plutôt que des bits. Le mot est l'élément d'information de base qui représente la plus petite unité adressable. C'est souvent l'octet, mais parfois un nombre quelconque de bits, 12, 16, 48...). La taille de cette unité adressable dépend du constructeur.

Un octet permet de coder $2^8 = 256$ caractères différents. L'ajout d'un bit double en effet, à chaque fois, le nombre de combinaisons possibles. Sur 16 bits, le nombre de codes différents possibles est de 65.536. L'octet se divise en deux quartets. Celui de gauche est appelé **quartet de poids fort**, car il contient la partie la plus lourde du nombre qu'il représente. Celui de droite est appelé **quartet de poids faible**.

b7	b6	b5	b4	b3	b2	b1	b0
0	1	0	0	0	0	0	1

b0, b1, b2, b3 sont les bits de poids faible; b4, b5, b6, b7 sont les bits de poids fort.

$(01000001)_2 = 0.2^7 + 1.2^6 + 0.2^5 + 0.2^4 + 0.2^3 + 0.2^2 + 0.2^1 + 1.2^0 = 64 + 1 = (65)_{10}$

La contribution du quartet de gauche est 64, celle du quartet de droite est 1.

L'écriture en binaire est inconcevable. On choisit d'écrire les codes dans un système de base plus élevée, le système hexadécimal (base 16). Les traductions, du binaire à l'hexadécimal et inversement, ne posent pas de problème (en cas de difficultés, voir plus loin le paragraphe sur la codification des nombres).

$(65)_{10} = (01000001)_2 = (41)_{16}$ (on note parfois 41H)

Le code ASCII

L'un des codes les plus connus des utilisateurs de PC est le code ASCII (American Standard Code for Information Interchange). Il a été défini en 1963 aux Etats-Unis puis adopté par d'autres organismes, notamment l'ISO (International Standards Organization) qui en a fait le code ISO-7. Ce code est aussi connu sous le nom d'alphabet international n°5 ou de CCITT n°5 puisque recommandé par l'avis n°5 du Comité Consultatif International Télégraphique et Téléphonique. C'est un **code à 7 bits** autorisant donc le codage de 128 caractères parmi lesquels, tous les caractères alphanumériques utilisés en anglais. Comme la plupart des ordinateurs traitent les bits par paquets de huit ou plus, le huitième bit est, soit inutilisé, soit utilisé comme bit de parité pour le contrôle lors de la transmission (voir plus loin), soit utilisé pour coder un maximum de 128 caractères supplémentaires tels des caractères graphiques et des caractères nationaux (caractères accentués). On parle alors de **code ASCII étendu**.

Le code ASCII de la lettre A est en hexadécimal 41, soit en binaire 01000001 ou encore en décimal 65. Certains logiciels et surtout certains langages exigent la codification en décimal. L'instruction en Basic PRINT CHR$(65) produira l'impression de la lettre A. La codification en hexadécimal est pratique pour ce qui est de la présentation de ces caractères en tableau puisque deux symboles sont nécessaires et suffisants pour leur codage.

Tableau des codes ASCII

Le tableau de la page suivante reprend les caractères du code ASCII non étendu. Les codes 0 à 31sont traditionnellement réservés à des caractères de contrôle dont l'existence est historique. Les plus connus d'entre eux sont certainement le 7 (BELL), le 8 (BACKSPACE) et le 13 (CARRIAGE RETURN).

Le caractère *$* est codé 24 en hexadécimal et donc 00100100 en binaire. Son code décimal est 36.
Le caractère *A* est codé 41 en hexadécimal et donc 01000001 en binaire. Son code décimal est 65.
Le caractère *Z* est codé 5A en hexadécimal et donc 01011010 en binaire. Son code décimal est 90.
Le caractère *a* est codé 61 en hexadécimal et donc 01100001 en binaire. Son code décimal est 97.

Quelques observations intéressantes

Avec un tel code, les transformations des majuscules en minuscules et inversement sont formellement triviales. Un exécutant formel doit simplement modifier le troisième bit (bit n°5). En français, l'emploi de caractères accentués complique un peu le problème. Une marche à suivre est cependant possible.

	b7	b6	b5	b4	b3	b2	b1	b0
A	0	1	0	0	0	0	0	1

	b7	b6	b5	b4	b3	b2	b1	b0
a	0	1	1	0	0	0	0	1

Autre constatation intéressante, les chiffres sont codés de telle sorte que le quartet de poids faible représente la valeur du chiffre dans le système binaire.

	b7	b6	b5	b4	b3	b2	b1	b0
5	0	0	1	1	0	1	0	1

				0010	0011	0100	0101	0110	0111
				2	3	4	5	6	7
0000	0			SP	0	@	P	`	p
0001	1			!	1	A	Q	a	q
0010	2			“	2	B	R	b	r
0011	3			#	3	C	S	c	s
0100	4			$	4	D	T	d	t
0101	5			%	5	E	U	e	u
0110	6			&	6	F	V	f	v
0111	7			‘	7	G	W	g	w
1000	8			(	8	H	X	h	x
1001	9			)	9	I	Y	i	y
1010	A			*	:	J	Z	j	z
1011	B			+	;	K	[	k	{
1100	C			,	<	L	\	l	|
1101	D			-	=	M	]	m	}
1110	E			.	>	N	^	n	~
1111	F			/	?	O	_	o	DEL

Tableau des principaux codes ASCII

Le code ANSI

C'est un des codes dérivés du code ASCII. Il est le produit de l'**A**merican **N**ational **S**tandards **I**nstitute. Les 128 premiers codes correspondent aux caractères du code ASCII. Les 128 autres sont une extension qui correspond à la **page de code** choisie. Il faut distinguer la *page de code hardware* qui réside dans la ROM de la *page de code préparée* qui peut, à partir d'une unité de disque, être superposée à la page de code hardware. Les pages de code sont numérotées. En voici quelques exemples.

437 Etats-Unis
710 Arabe
850 Multilingue (latin 1)
852 Slave (latin 2)
860 Portugal
862 Hébreu
863 France et Québec
865 Norvège et Danemark

Des commandes du système d'exploitation permettent de préparer les pages de code au niveau du fichier de configuration.

La proximité de certains codes (ASCII et ANSI par exemple) explique les problèmes rencontrés lors de l'exportation et l'importation de fichiers textes. Dans ce cas, des symboles particuliers remplacent notamment les caractères accentués.

Il arrive qu'un système (ordinateur + programme) propose le choix du type de conversion lors de la transformation en fichier texte d'un document. Dans le monde PC, on associe le code ASCII au travail sous DOS et ANSI au travail sous Windows.

A titre d'exercice, vous pouvez tenter les deux expériences suivantes:
- sauver un texte au format ASCII (DOS) et le récupérer avec un éditeur ou traitement de texte fonctionnant sous Windows;
- sauver un texte au format ANSI (Windows) et le récupérer avec un éditeur ou traitement de texte fonctionnant sous DOS.

Observez le résultat.

Exemple: le texte suivant *"Cet été va être tiède, ça c'est sûr!"*
- sauvé en ASCII et récupéré comme ANSI donnera *"Cet ,t, va ˆtre tiŠde, ‡a c'est s–r!"*
- sauvé en ANSI et récupéré comme ASCII donnera *"Cet ¨ t¨ va Ùtre tiÖde, ôa c'est s√r!"*

Le code EBCDIC

L' **E**xtended **B**inary **C**oded **D**ecimal **I**nterchange **C**ode est un code à 8 bits créé par IBM pour ses ordinateurs des séries 360 et 370. L'histoire ne dit pas si la volonté d'IBM était alors d'entretenir l'incompatibilité ou de créer un standard.

Le DCB

Les chiffres sont des caractères comme les autres. En DCB, Décimal Codé Binaire, chacun des chiffres est codé en binaire. Or pour coder un chiffre, un quartet suffit. Dans ce cas, le quartet de gauche est constitué de zéros ou de uns selon les constructeurs. Ainsi, le nombre 125 peut-il être codé 0000 **0001** 0000 **0010** 0000 **0101**. Le Décimal Codé Binaire condensé (BCD packed) est un code autorisant la codification des chiffres sur un seul quartet. Ainsi, le nombre 125 peut-il être codé **0001 0010 0101**. On parle aussi de code 8421 en référence au poids de chacun des bits.

Les codes à quatre bits

La codification 8421 nous amène naturellement à parler d'autres codes possibles pour les chiffres, vu que seulement dix symboles sont à coder pour 16 codes disponibles. Parmi les autres codifications qui existent, citons le code 2421, le code 7421, le code majoré de 3 (XS-3 excess 3), etc.

Exemple: le chiffre 9 sera donc codé 1001 en 8421, 1111 en 2421 et 1100 en code majoré de 3.

Questions et exercices: dressez les tables des codes 2421, 7421, 742-1 et XS-3. Les codes 2421 et XS-3 sont appelés codes auto-complémentaires. Pourquoi? Quel autre avantage possède le code XS-3?

L'intérêt de ces différents codes réside aussi dans le fait qu'à chaque fois, des combinaisons sont interdites, ce qui facilite, par exemple, les contrôles de transmission. Nous en reparlons plus abondamment par la suite. Ainsi, dans le code XS-3, la combinaison 0000 ne correspond a aucun chiffre. Une transmission "nulle" est donc aisément détectable.

D'autres codes

Parmi les codes utilisés pour faciliter le transfert des informations (autrement dit, la détection des erreurs) et l'implémentation, il faut citer le code **deux parmi cinq**. Chaque chiffre est représenté par une combinaison de deux "1" et donc trois "0". Il y en a exactement 10. C'est un code 74210 où 0 est codé 11000.

Le code biquinaire (code 50 43210) est un code à 7 bits utilisant exactement deux "1". L'un dans les deux premiers bits, l'autre dans les cinq derniers.

Le code MBQ est un code à 4 bits dérivé du code biquinaire. Les deux premiers bits sont remplacés par un seul et les cinq derniers par trois bits dans lequel on utilise une codification en BCD.

Le code de Gray montre que la recherche d'un code efficace est parfois liée à des exigences matérielles (en l'occurrence, la conversion de données analogiques continues en données digitales). Dans le code 8421 par exemple, le 7 est codé 0111 et le 8 est codé 1000. Ce qui pose des problèmes pour un système électromécanique qui doit traduire du continu en discret. Gray solutionne ce problème: son code est tel qu'un seul bit change d'une valeur discrète à la suivante. Comme on peut aisément le constater, ce code est un code cyclique.

Chiffre	2 parmi 5	biquinaire	MBQ	Gray
0	11000	01 00001	0 000	0000
1	00011	01 00010	0 001	0001
2	00101	01 00100	0 010	1001
3	00110	01 01000	0 011	1101
4	01001	01 10000	0 100	0101
5	01010	10 00001	1 000	0111
6	01100	10 00010	1 001	1111
7	10001	10 00100	1 010	1011
8	10010	10 01000	1 011	0011
9	10100	10 10000	1 100	0010

Codification des nombres et lien avec les opérations

Nous venons de voir que la codification des nombres se fait comme celle des chaînes de caractères. Toutefois, lorsque ces nombres sont destinés à des calculs, d'autres codifications s'imposent.

Loin de nous l'idée de développer en long et en large de grandes théories sur les systèmes de numération et les principes de conversion. Toutefois, quelques connaissances de base paraissent inévitables. Nous les rappelons pour information.

Les systèmes de numération les plus intéressants pour ce qui nous concerne sont les systèmes: décimal, binaire, octal et hexadécimal. Cela n'a rien d'étonnant, puisque lié à notre propre système de numération et à l'architecture des ordinateurs.

Pour comprendre les systèmes de conversion, il faut se rappeler la signification des chiffres dans un nombre. Ils n'ont pas tous le même poids. Dans le nombre 1031, le chiffre 1 situé le plus à gauche est de poids plus fort que le chiffre 1 situé le plus à droite. En réalité:

$(1031)_{10} = 1 \times 10^3 + 0 \times 10^2 + 3 \times 10^1 + 1 \times 10^0$

Le système est dit "système en base 10" car basé sur l'utilisation de dix symboles différents. A partir de maintenant, nous prendrons l'habitude de renseigner chaque nombre dans sa base (en indice) pour éviter la confusion.

Il en est de même dans les autres systèmes, ce qui donne une méthode générale de conversion d'un système quelconque au système décimal. Par exemple:

$(457)_8 = 4 \times 8^2 + 5 \times 8^1 + 7 \times 8^0 = (303)_{10}$

$(3F)_{16} = 3 \times 16^1 + 15 \times 16^0 = (63)_{10}$

Dans le système hexadécimal, il est nécessaire d'utiliser 16 symboles: les dix chiffres et les six premières lettres de l'alphabet.

$(1011101)_2 = 1 \times 2^6 + 0 \times 2^5 + 1 \times 2^4 + 1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 = (93)_{10}$

Comme on peut le constater, les chiffres ont d'autant plus de poids qu'ils sont situés à gauche. Le poids est la puissance de la base correspondant à sa position (numérotation de 0 à n en commençant par la droite).

b7	b6	b5	b4	b3	b2	b1	b0
0	1	0	0	0	0	0	1

Le poids du deuxième chiffre est 2^6. Celui du dernier, 2^0.

Le passage du système décimal à d'autres systèmes demande de réaliser l'opération inverse, à savoir des divisions successives. Les restes sont pris en compte et constituent les chiffres du nombre converti dont le poids augmente avec le nombre de divisions.

3765 : 10 = 376 reste 5
376 : 10 = 37 reste 6
37 : 10 = 3 reste 7
Le nombre est reconstitué en prenant le dernier résultat (3) suivi de tous les restes (7, 6, 5).

Exercices: convertissez en binaire, octal et hexadécimal $(11)_{10}$, $(512)_{10}$, $(725)_{10}$.

Les conversions entre systèmes dont les bases sont des puissances l'une de l'autre sont excessivement simples.

$(1\ 1010\ 0001\ 1101)_2 = (1A1D)_{16}$
$(110\ 100\ 001\ 101)_2 = (6415)_8$
De la plus petite base vers la plus grande, il suffit de grouper les symboles par paquets en nombre égal à la puissance qui les relie (de 2 à 16, grouper par paquets de 4, de 2 à 8 par paquets de 3).
De la plus grande vers la plus petite, chaque symbole peut être converti directement en utilisant le même nombre de symboles chaque fois (dans les exemples qui suivent, respectivement quatre et deux).
$(A45)_{16} = (1010\ 0100\ 0101)_2$
$(A45)_{16} = (22\ 10\ 11)_4$
Les opérations peuvent se réaliser comme classiquement, avec des reports (NB: 1 + 1 = 10, j'écris 0, je reporte 1):

```
    11011100011
+   11100101110
  111000010001

    A45
+   1C65
  26AA
```

NB: A + C = 16, j'écris 6, je reporte 1.

Les soustractions, multiplications divisions se réalisent selon les mêmes principes que dans le système décimal.

```
   11011100011          10011
-   1100101110      *    1101
-------------        --------
   1110110101           10011
                       10011
                      10011
                     --------
                     11110111
```

Exercice: réalisez en binaire, la division de 11011101 par 111; contrôlez en convertissant tous les nombres en décimal.

Si les modes de réalisation de ces opérations ne sont pas nouveaux (seul le nombre de symboles est réduit), il faut trouver des représentations qui permettent d'implémenter les opérations.

Le problème du signe

Il est peut-être bon de se rappeler qu'une soustraction est l'addition d'un nombre et de l'opposé de l'autre: 548 - 489 = 548 + (-489)

La soustraction est implémentée si une représentation correcte du signe est trouvée. Une manière de procéder consiste à utiliser le premier bit (celui qui est le plus à gauche) pour coder le signe (0 = positif, 1 = négatif).

18 serait codé 0.10010 alors que -21 serait codé 1.10101 (le point n'existe pas en réalité).

Dans ce cas, les additions et les soustractions doivent être réalisées différemment (emploi de circuits électroniques différents). Un autre élément entre en ligne de compte, la représentation des nombres se fait sur un nombre limité de bits. C'est une limite importante à la technique habituelle d'addition. Pour contourner ces deux difficultés, on utilise une autre représentation faisant appel à la technique du complément.

Le complément restreint et le complément vrai (ou complément à 2)

Pour bien comprendre le mode de fonctionnement des opérations, rien de tel qu'un exemple choisi dans notre système favori, le système décimal. Lorsque nous faisons du calcul écrit, nous pouvons remplacer une opération de soustraction par une opération d'addition en procédant de la sorte:

```
   525          525          525
  -236         +763         +764
-------      -------      -------
   289       (1)288       (1)289
```

Que constatons-nous? Si nous remplaçons le nombre 236 par le nombre obtenu en remplaçant les chiffres par leur **complément à 9**, soit 763, la soustraction se transforme en addition avec un report final qui doit être ajouté au chiffre le plus à droite (288 + 1 = 289).

Ceci s'explique simplement par le fait que 525 - 236 = 525 + (999 - 236) - 1000 + 1.

Si nous remplaçons le nombre 236 par son **complément à 10** (nombre obtenu en remplaçant les chiffres par leur complément à 9 et le chiffre le plus à droite par son complément à 10), soit 764, la soustraction se transforme en addition avec un report final qui doit être négligé.

Ceci s'explique simplement par le fait que 525 - 236 = 525 + (1000 - 236) - 1000.

L'avantage de ces techniques c'est qu'elles fonctionnent aussi lorsque le nombre soustrait est supérieur, en valeur absolue au premier nombre.

```
  236      236      236
 -525     +474     +475
------   ------   ------
 -289      710      711
           -290     -289
```

Lorsqu'il n'y a pas de report, le résultat est négatif et doit être "décomplémenté".

236 - 525 = 236 + (999 - 525) - 1000 + 1
236 - 525 = 236 + (1000 - 525) - 1000

Les opérations - 1000 +1 et - 1000 servent à décomplémenter le résultat.

Les mêmes techniques sont utilisées dans le système binaire. On parle de **complément restreint** (complément à 1) et de **complément vrai** (complément à 2).

Le complément restreint d'un nombre binaire s'obtient très facilement: il suffit d'inverser la valeur de tous les bits. Le complément vrai s'obtient en ajoutant 1 au complément restreint.

Soit à réaliser l'opération 15 - 27 en complément restreint. L'opération est réalisée de la manière suivante: le nombre 15 est additionné au complément restreint de 27. S'il n'y a pas de report au dernier chiffre, le nombre est négatif et la réponse obtenue est son complément restreint, sinon, le nombre est positif et la réponse doit être augmentée du report.

```
15  00001111  00001111
27  00011011  11100100  (complément restreint de 27)
              11110011  (pas de report, donc nombre négatif)
12            00001100  (la réponse est -12)
```

Soit à réaliser l'opération 112 -58.

112	01110000	01110000	
58	00111010	11000101	(complément restreint de 58)
	1	00110101	(report final, donc nombre positif)
54		00110110	(prise en compte du report)

Au niveau du fonctionnement de la machine, c'est le contenu d'un bit du registre d'état qui mémorise l'éventuel report déterminant ce qu'il y a lieu de faire (recherche du complément ou addition du report).

Reprenons les deux soustractions précédentes en complément vrai.

15 -27 = ...

15	00001111	00001111	
27	00011011	11100101	(complément vrai de 27)
		11110100	(pas de report, donc nombre négatif)
12		00001100	(on décomplémente, la réponse est -12)

112 - 58 = ...

112	01110000	01110000	
58	00111010	11000110	(complément vrai de 56)
54		00110110	(report final, donc nombre positif; c'est la réponse)

Les nombres fractionnaires

Comment coder en binaire, des nombres non entiers? Le principe de décomposition en puissances de la base reste identique, sauf qu'il s'agit de puissances négatives. Ainsi, $2^{-1} = 0,5$; $2^{-2} = 0,25$; ... La partie décimale d'un nombre peut donc s'exprimer comme une somme de puissances négatives de deux.

$4,75 = 4 + 0,5 + 0,25 = 2^2 + 2^{-1} + 2^{-2}$

Sa représentation binaire sera donc donc 100,11. Il est possible, il est même probable, que des nombres à représentation décimale très limitée aient une représentation binaire très longue, voire illimitée. En voici un exemple:

12,4 = 8 + 4 + 0,25 + 0,125 + 0,0625 + ...

Sa représentation binaire sera donc donc 1100,0111...

Pour trouver le correspondant binaire de la partie fractionnaire d'un nombre on peut pratiquer de la sorte: multiplier par deux la partie décimale; si le résultat est supérieur à 1, le chiffre suivant est 1, sinon c'est 0 et on poursuit, avec l'excédant de 1 où le nombre obtenu s'il n'excède pas 1.

0,4	x 2	0,8	0,0
0,8	x 2	1,6	0,01
0,6	x 2	1,2	0,011
0,2	x 2	0,4	0,0110
0,4	x 2	0,8	0,01100
0,8	x 2	1,6	0,011001

On devine que 0,4 aura comme représentation binaire 0,01100100110011...

Les nombres en virgule flottante

La façon la plus compacte de représenter un nombre (quel que soit le système) est probablement la représentation en virgule flottante, pourvu qu'on utilise une forme normalisée. La notation en virgule flottante se compose d'une mantisse comprenant les chiffres significatifs et d'une caractéristique comprenant l'exposant de la base auquel il faut élever cette mantisse.

$15{,}3\ 10^{5}$ $-214\ 10^{-2}$ $5\ 10^{9}$ $0{,}001\ 10^{-3}$ sont des représentations en virgule flottante. Une représentation normalisée, plus économique, fera appel a un minimum de symboles. La partie entière de la mantisse sera nulle et le premier chiffre suivant la virgule sera significatif.

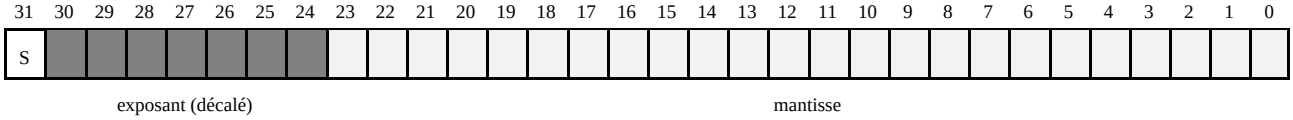
.153 E7 -.214 E1 .5 E10 .1 E-5 sont des représentations normalisées des nombres déjà cités.

Question: comment donner à 0, une représentation en virgule flottante normalisée?

Selon les constructeurs, les normes de représentation des nombres en virgule flottante varieront, autorisant des niveaux de précision assez différents.

Exemple: la norme IBM où les nombres sont codés sur deux mots de 16 bits

En simple précision, 3 octets sont utilisés pour coder la mantisse. L'octet restant sert à coder la caractéristique (7 bits) et le signe (1 bit).



Le décalage de l'exposant (ici 64) permet de coder des exposants négatifs et positifs. L'exposant réel est augmenté de 64 avant d'être codé. Ainsi, 5 est codé 1000101 et -5 est codé 0111011. Les valeurs possibles de l'exposant s'étalent donc ici de -64 à +63. Elles sont toutes codées positivement.

Dans un format (8,23) au lieu de (7,24), la fourchette des exposants possibles est de -128 à +127.

Exercice: représentez, en utilisant cette norme, les nombres 0,15625 et 57,8125.

Pour une plus grande précision, on peut employer quatre mots au lieu de deux. On parle alors de nombres au format **double précision**. Des formats tels que (7,56), (8,55), (11,52) existent, de même que des représentations sur 80 bits (15,64) et 128 bits (15,112).

Exercice: une version de Turbo Pascal utilise un type réel de base nécessitant six octets pour le codage (dont un pour l'exposant et le signe). Avec quelle portée pour les exposants et avec combien de chiffres significatifs peut-on espérer pouvoir travailler?

Les nombres en virgule fixe

Dans cette technique, les nombres sont traités commes des entiers et c'est au programmeur de gérer le problème du placement de la virgule. Les nombres sont enregistrés sous forme d'un nombre entier et d'un signe ou sous forme BCD.

Le contrôle des erreurs

Les sytèmes de codage dont nous venons de parler font parfois mention d'une nécessité de contrôler les données, notamment lors de transmissions. Examinons simplement comment certains de ces contrôles peuvent avoir lieu.

Le contrôle de parité

Il ressemble, en beaucoup plus simple, au type de contrôle effectué sur un numéro de compte en banque grâce aux deux derniers chiffres. Selon le nombre de bits dont la valeur est 1, le premier bit, appelé dans ce cas, bit de parité, reçoit la valeur 0 ou 1. Pour compliquer un peu les choses, on parle de parités paire et impaire.

En parité paire (b7 est le bit de parité), le premier bit reçoit une valeur telle que le nombre total de bits à 1 soit pair.

b7	b6	b5	b4	b3	b2	b1	b0
0	1	0	1	1	0	0	1

En parité impaire, le nombre total de bits à 1 doit être impair.

b7	b6	b5	b4	b3	b2	b1	b0
1	1	0	1	1	0	0	1

L'efficacité de ce code n'est réelle que dans la mesure où il n'y a pas plus d'une erreur produite, par mot, lors de la transmission.

Exemple: si à la réception (parité impaire), l'octet ci-dessus est dans l'état suivant:

b7	b6	b5	b4	b3	b2	b1	b0
1	1	0	1	0	0	0	1

une erreur est détectée. Malheureusement, elle ne peut être corrigée. Si l'octet censé transmettre la même information est dans l'état suivant:

b7	b6	b5	b4	b3	b2	b1	b0
1	1	0	0	0	0	0	1

les erreurs ne seront même pas détectées.

C'est la raison pour laquelle des codes sont mis au point pour permettre la détection et même la correction du plus grand nombre possible d'erreurs. On parle de codes autovérificateurs et/ou autocorrecteurs. Les codes de Hamming (ingénieur américain) sont parmi les plus connus.

Les codes de blocs

Principe: l'information est scindée en blocs de taille fixe. A chaque bloc, sont ajoutés des bits de contrôle de manière à former des blocs particuliers. On parle de codes de blocs en ce sens que chacun des blocs obtenus doit faire partie d'un ensemble de combinaisons (mots) acceptables. Si c'est le cas, les blocs sont décomposés à la réception. Sinon, l'erreur est signalée ou mieux, elle est corrigée.
On définit la distance de Hamming comme étant le nombres de bits qui changent entre deux mots successifs du code. Plus la distance est grande, plus le code est sûr. Dans un code normal, la distance de Hamming est de 1, ce qui signifie que toutes les combinaisons sont possibles et les erreurs indétectables.
Des exemples. Nous avons déjà parlé de codes tels le "2 parmi 5" utilisé pour le codage des chiffres dans lequel beaucoup de combinaisons sont non valides. Les codes "i parmi n" sont effet assez efficaces au niveau de la détection des erreurs. Dans un code 4 parmi 8, le nombre de combinaisons acceptables est 70 pour 256 combinaisons possibles. La distance de Hamming de ce code est de 2.

0 0 0 0 1 1 1 1
0 0 0 1 0 1 1 1
0 0 0 1 1 0 1 1
0 0 0 1 1 1 0 1
0 0 0 1 1 1 1 0
0 0 1 0 1 1 1 0
0 0 1 1 0 1 1 0
...

Le nombre de bits différents d'un mot à son suivant est deux. Ce code permet la détection des erreurs simples. Il est possible d'augmenter la distance de Hamming en utilisant une technique comme celle des parités croisées, par exemple.
Soit à transmettre le message *"OK!"*
Il y a trois octets à transmettre. On forme un bloc auquel on ajoute des bits de contrôle de parité verticale et horizontale, ainsi qu'un bit de parité croisée.

	O	K	!	Contrôle de parité horizontale	
Contrôle de parité verticale	1	0	0	1	← Bit de parité croisée
	0	0	0	0	
	1	1	0	0	
	0	0	1	1	
	0	0	0	0	
	1	1	0	0	
	1	0	0	1	
	1	1	0	0	
	1	1	1	1	

Imaginez que lors de la transmission, un bit soit erroné (celui qui est en gras dans le tableau suivant).

			Erreur détectée ↓		
	O	K	!	Contrôle de parité horizontale	
Contrôle de parité verticale	1	0	0	1	← Bit de parité croisée
	0	0	0	0	
	1	1	0	0	
	0	**1**	1	1	← Erreur détectée
	0	0	0	0	
	1	1	0	0	
	1	0	0	1	
	1	1	0	0	
	1	1	1	1	

L'erreur est, non seulement détectée, mais peut être corrigée sans difficulté à la réception. Avec ce type de codage, la distance de Hamming est de 4. La modification d'un seul bit devrait entraîner la modification de trois autres pour que le bloc soit acceptable: celui du contrôle de parité horizontale, celui du contrôle de parité verticale et celui de la parité croisée.

L'erreur est détectée horizontalement et verticalement. Elle est donc localisée.

Question: que se passe-t-il s'il y a deux bits erronés à la réception? Les erreurs seront-elles détectées? corrigées?

Les codes cycliques

Ces codes, encore appelés codes polynomiaux, sont basés sur l'utilisation d'un polynôme générateur. Partons d'un exemple.

On peut associer à l'information 1010111 le polynôme $P(x) = x^6+x^4+x^2+x^1+x^0$ ou encore, $x^6+x^4+x^2+x+1$. Choisissons un polynôme générateur G(x): par exemple, x^3+1. Multiplions le polynôme de départ par G(x)-1, puis divisons le polynôme obtenu par G(x). Cette division, qui donne S(x), admet un reste, R(x). Ce reste ajouté à P(x) donne un polynôme P'(x) qui est divisible par G(x).

Mathématiquement, pour ceux qui auraient un doute,

P(x). (G(x)-1) = S(x).G(x)+R(x) ou encore,
P(x).G(x)-P(x) = S(x).G(x)+R(x) ou encore,
(P(x)-S(x)).G(x) = P(x)+R(x)

Cette dernière équation prouve que P'(x) = P(x)+R(x) est divisible par G(x).

Si donc on transmet l'information correspondant à P'(x) à laquelle on accole l'information correspondant à R(x), le récepteur peut en vérifier la validité en effectuant la division de P'(x) par G(x) et si la division est exacte (transmission correcte), enlever le reste de P'(x) pour reconstituer l'information de départ.

Concrètement ici,

1010111 x 1000 = 1010111000
1010111000 : 1001 = 100110 reste 11
1010111 + 11 = 1011010

L'information transmise est 01011010.0011 (8 bits pour l'information transformée et 4 pour le reste).

Le récepteur effectue l'opération de vérification 1011010 : 1001 ce qui donne 1010 reste 0. Le contrôle est positif; l'information peut être reconstituée 1011010 - 11 = 1010111.

Une autre manière d'expliquer la méthode est de considérer que l'information à transmettre est systématiquement multipliée par un nombre binaire puis divisée par ce nombre binaire augmenté de 1. Le reste de la division est ajouté à l'information qui est transmise avec le reste. Pour vérifier la transmission, le récepteur divise l'information transformée par le nombre binaire augmenté de 1. Si la division est exacte, la transmission s'est bien passée et il reste à déduire le reste de l'information transformée pour reconstituer l'information de départ. Ouf!

Ces opérations peuvent paraître fastidieuses. Il faut les replacer dans le contexte d'un calculateur rapide. Des recherches mathématiques ont conduit à la recherche de polynômes générateurs permettant de détecter un maximum d'erreurs. Ainsi, le polynôme $x^{16}+x^{12}+x^5+1$ est un des plus utilisés. Il permet de détecter toutes les erreurs simples et doubles, toutes les erreurs sur un nombre impair de bits et tous les paquets d'erreurs d'une longueur au moins égale à 16 bits. La redondance est faible (16 bits par paquets de 1000 bits environ).

Coder les images (un exemple): la reconnaissance des caractères

Lorsqu'un caractère est passé au scanner, son image est digitalisée. Le but d'un logiciel d'OCR (Optical Character Recognition: reconnaissance optique des caractères) est de convertir cette image en une représentation beaucoup plus simple pour un programme orienté texte qui désirerait l'utiliser, le code ASCII (ou équivalent) du caractère lu. Comment, à partir de sa lecture optique sur une feuille de papier par exemple, peut-on retrouver ce code dans la mémoire de l'ordinateur. Les techniques, au départ, sont celles qui se réfèrent au codage des images. Elles ne sont donc pas différentes de celles qui sont exposées dans la fiche pédagogiques n°8.1: *Dessin assisté par ordinateur - Les actions de base*.

Lors de la lecture, le texte est en effet considéré comme une image, un dessin, et non comme un ensemble de caractères.

Comparaison des pixels (dessin bitmap)

Le principe le plus simple consiste à comparer, pixel par pixel, la matrice obtenue avec une matrice de correspondance. La lettre reconnue est celle pour laquelle le nombre de différences est le moins élevé. Sur une matrice de 2.500 pixels, les différences se monteront à plusieurs centaines pour les caractères différents du caractère scanné et à seulement quelques pixels pour le caractère à reconnaître.

Question: ce système a cependant une énorme limite. Voyez-vous laquelle?

Comparaison des caractéristiques (dessin vectorisé)

Une façon de faire qui évite cet inconvénient est de travailler, non sur les pixels, mais sur les caractéristiques, les formes du caractères. Un caractère se compose de droites horizontales, verticales, diagonales, hautes, basses, au centre, de courbes fermées, ouvertes, à gauche, à droite. Bref, il est possible de donner du caractère une description plus ou moins complète. La comparaison se fait alors entre l'image scannée et les différents caractères sur base de ces détails.

Toutefois, de nombreux problèmes subsistent, certains traitements n'étant pas aussi formalisables qu'il n'y paraît. Ainsi en est-il de la distinction du O et du 0 qui se fait souvent sur des bases contextuelles. Les caractères sont analysés en fonction de leurs voisins immédiats.

D'autres problèmes doivent être solutionnés tels la distinction entre texte et image, l'ignorance de certains éléments perturbants tels tâches, ratures, lettres liées etc.

La compression des données

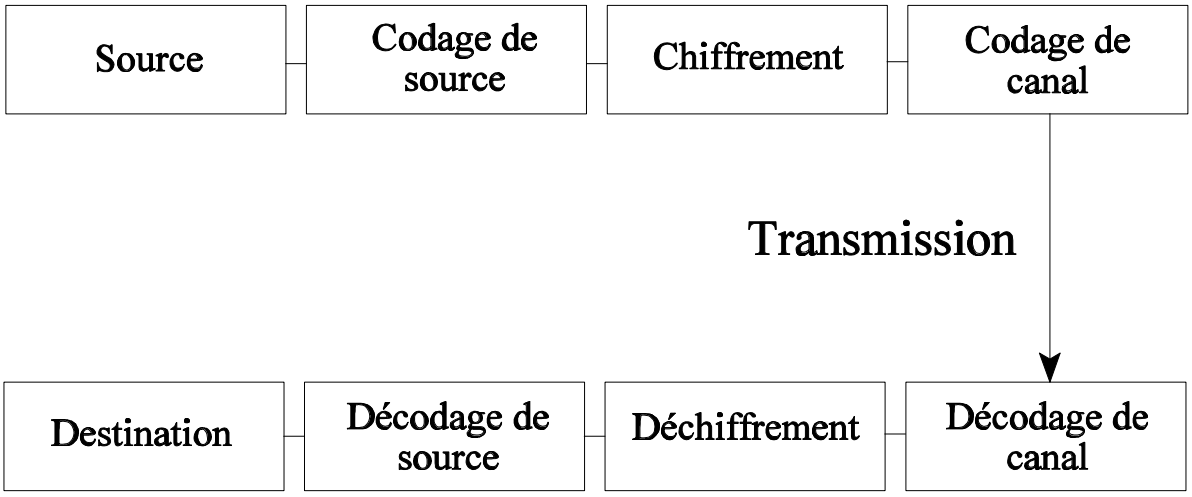
La compression des données se justifie par l'augmentation permanente de leurs volumes et par la multiplication des procédés de transmission dont elles font l'objet. On utilisera donc la compression des données:

- pour augmenter un espace disque,
- pour stocker une image gourmande
- pour accélérer la transmission des données (et donc en minimiser les coûts).

On peut distinguer deux types de compressions: celles dont les techniques sont réversibles (aucune perte d'information) et celles dont les techniques sont irréversibles (chiffres non significatifs, espaces blancs inutiles) et pour lesquelles d'ailleurs on utilise plutôt le terme "compactage". Ces dernières impliquent une perte d'informations qui n'affecte pas (ou très peu) le traitement futur des données. Elles sont par exemple fort présentes dans le domaine du traitement des images et des sons.

Pour des raisons évidentes, certains protocoles, garantissant l'intégrité des données, n'acceptent pas les procédés de compression irréversibles. Ces procédés sont basés sur la redondance de certaines informations, tant lorsque les données sont de types alphanumérique et numérique que lorsqu'il s'agit d'images et de sons.

Le processus le plus complet de transmission de données se compose généralement de plusieurs phases de codage:



- le codage de la source qui réduit considérablement la taille du message (nous en examinerons bientôt les techniques),
- le chiffrement destiné à sécuriser les données (si nécessaire),
- le codage de canal qui augmente un peu la longueur du message et sert à détecter et à corriger des erreurs éventuelles de transmission.

Le schéma qui précède traduit cette description.

A l'autre bout, le décodage se produit dans l'ordre inverse:

- le décodage de canal,
- le déchiffrement,
- le décodage de source. Nous nous sommes déjà intéressés au codage de canal. Nous ne parlerons pas du chiffrement mais nous nous intéresserons en détail au codage et au décodage de source. Commençons par examiner les différents types de redondance des informations et les algorithmes qui y sont attachés.

Les types de redondance

La **fréquence** des caractères peut être déterminante lors du codage. En effet, si le code est un code à longueur variable, il y a intérêt à ce que les caractères les plus fréquents soient codés de façon plus

courte. C'est le principe utilisé par l'algorithme de Huffman développé plus loin. Il se base sur la fréquence des caractères dans la source pour définir un code. Dans un code comme le code Morse, par exemple, le E et le T, les deux lettres les plus fréquentes en anglais ont les codes les plus courts (respectivement: un point, une barre). Signalons encore que la fréquence des lettres dans les textes français est bien connue, mais qu'il existe aussi des contextes locaux dans lesquels ces paramètres peuvent changer.

L'apparition régulière, voire l'emploi limité du nombre de mots, conduisent à des codages par dictionnaire. A titre indicatif, sur 16 bits on peut coder 65.536 mots différents, ce qui n'est pas si mal pour un texte. Si la longueur moyenne d'un mot est de cinq caractères, l'économie est de plus de 50%.

Un troisième type de redondance est constitué par la répétition des caractères ou leur corrélation. Si la répétition concerne davantage les images dans lesquelles les suites de bits de valeurs identiques sont courantes, les corrélations sont nombreuses entre les caractères d'un texte et peuvent être exploitées: point suivi d'un espace et d'une majuscule, *c* suivi d'un *h*, etc.

De la combinaison de ces trois catégories de redondances, va dépendre l'efficacité des algorithmes de compression qui seront mis au point. Par exemple, on peut combiner un codage par dictionnaire avec le choix de codes plus courts pour les mots les plus fréquents.

Donnons maintenant des exemples précis des techniques employées.

Codage des répétitions

Cette méthode est connue sous le nom de *run-length encoding*. Imaginons une suite de caractères dont la plupart se répètent assez souvent:

a a a a d d d d d d d a a a a a a a b b b c c c d d d d

Une seule instance peut être codée, de même que le nombre de répétitions. Un fanion doit cependant être utilisé car pour des caractères isolés ou peu répétés, le procédé peut s'avérer très lourd.

Fanion	Caractère	Nombre

Bien sûr, le fanion ne peut prêter à confusion. On voit que le procédé est rentable lorsque le caractère est répété plus de trois fois. Il est aussi possible de partir du fait que tous les caractères sont répétés, même 0 fois ou encore de doubler le caractère. La méthode peut encore être améliorée. Lorsqu'on travaille avec le code ASCII à 7 bits, le bit de parité peut être utilisé comme fanion. Il est alors utilisé en viol de la règle de parité pour signaler le caractère à répéter qui est immédiatement suivi du nombre de répétitions.

Viol de la règle de parité →		Caractère ASCII	Nombre de répétitions

Dans ce cas on utilise seulement deux octets. Le procédé est rentable à partir de trois répétitions. Le nombre de répétitions possible est 256. En réalité il est de 258, si on admet que 0 signifie 3 répétitions.

Si on travaille avec le code EBCDIC on utilise une autre technique car il s'agit d'un code à 8 bits. On sait toutefois, que les codes commençant par 1011 sont inemployés. Voici donc comment on procède:

1011	Nombre de répétitions	Caractère EBCDIC

Ici le nombre de répétitions est limité à 18 car il n'y a que quatre bits pour coder ce nombre. Ce procédé est employé dans certains protocoles pour l'élimination des espaces.

La suppression des répétitions est aussi utile pour la compression des images en noir et blanc. Comme ces images ne sont que des séquences de 0 et de 1, seules les positions des changements doivent être enregistrées.

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
0	0	0	0	0	0	0	1	1	1	1	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	0	0	0

La suite de nombres suivante permet la reconstitution de la séquence: 0, 8, 12, 22, 30. Cette technique, alliée à celle de la corrélation entre les lignes, est la base de la compression des images de télécopie.

Codage par représentation topographique

C'est une technique qui s'avère intéressante lorsqu'un seul caractère est très fréquent. C'est par exemple le cas de l'espace.

En voici un exemple:

la chaîne de caractère C X X X O X D X X A G E X X peut être comprimée de la façon suivante:

10001010011100	CODAGE

La représentation topographique précède la chaîne comprimée. Elle comprend des 1 à la place des caractères qui sont restés et des 0 aux endroits où le caractère supprimé doit réapparaître. Pour un message dont la longueur moyenne des mots est de six caractères, codés sur sept bits, chaque mot monopolisera en moyenne 49 bits. Si on choisit une représentation topographique de sept bits, il n'y a aucune économie. Toutefois, certains messages ont une proportion d'espaces beaucoup plus importante.

Cette technique peut aussi être employée dans les messages comprenant un nombre conséquent de chiffres. Ceux-ci peuvent être codés sur quatre bits au lieu de huit. La représentation topographique sert à localiser les chiffres dans le message.

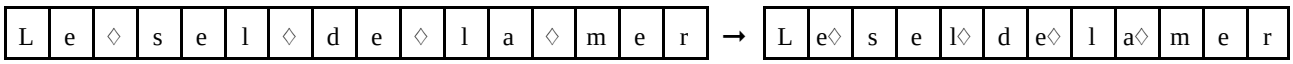
10000100	A1256.89

Dans ce cas précis, l'économie est de deux octets (soit 6 x 4 - 8 bits).

Codage par groupes de caractères

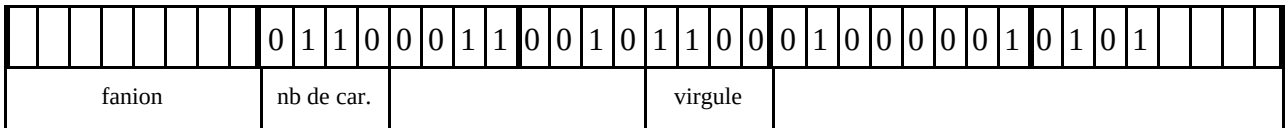
Dans les fichiers source des programmes, des mots-clés sont utilisés en nombre limité. Leur codage peut s'avérer efficace pour la compression. Dans un texte normal, les espaces étant nombreux, les paires de caractères constituées d'une lettre suivie de l'espace sont fréquentes. Leur codage (possible

en utilisant les combinaisons orphelines de l'EBCDIC ou les caractères étendus de l'ASCII) permet l'économie d'un octet à chaque fois.



Compression des nombres en demi-octet

Nous avons déjà signalé que les chiffres pouvaient être codés sur un demi-octet. Pour que cette compression puisse avoir lieu, l'emploi d'un fanion pour chaque nombre est nécessaire. Ce fanion doit être suivi du nombre de caractères concernés par la compression.



Le gain est de 3 octets pour un coût de 1,5 octet (fanion + nombre de caractères). Le nombre de chiffres maximum est de 20 (16 valeurs possibles à partir de 5 pour en tirer un bénéfice). De la sorte, un nombre de 20 chiffres est codé sur 12 octets (10 + 1,5) au lieu de 20.

La compression est simple à implémenter car, tant en ASCII qu'en EBCDIC, les bits de poids fort de tous les chiffres sont identiques (0011 en ASCII, 1111 en EBCDIC), et les bits de poids faible correspondent à la valeur du chiffre. Ils sont donc aisément repérables.

Notez au passage, le bel exercice d'algorithmique qui est ici proposé. Un programme de compression de chaîne peut être réalisé sur base de l'introduction de cette chaîne en binaire et du choix d'un fanion.

Notez aussi que si le fichier à compresser contient des nombres très grands, on peut utiliser un compteur de chiffres à huit bits. C'est le cas de certains fichiers scientifiques et financiers. On peut également inclure un certain nombre de symboles particuliers (+, $, ...) pour augmenter la longueur des chaînes comprimables. La codification des chiffres (de 0 à 9) laisse libres les codes de 10 à 15 pour ceux-ci. Ils sont aussi aisément détectables dans la mesure où les bits de poids fort sont différents pour ces symboles et pour les chiffres (exemple: 1011 pour + en ASCII et 1110 en EBCDIC).

Codage par dictionnaire

Employer un dictionnaire, c'est aussi accorder de l'importance à la sémantique qui permet de distinguer les messages valides des non valides. Le dictionnaire contient les messages valides et leur code. La compression sera d'autant plus importante que le nombre de messages valides est peu élevé. En voici un exemple. L'emploi d'un matricule codé sur 16 bits permet d'identifier plusieurs milliers d'individus. Si on admet que chacun des noms de ces individus fait 7 caractères en moyenne, le gain est de 5 octets. De plus, l'emploi d'un dictionnaire facilite le tri des informations.

Cette manière de procéder a un inconvénient majeur. L'utilisation d'une table unique n'est généralement pas possible, vu les dimensions excessives qu'elle peut avoir. On procède souvent par fractionnement en plusieurs tables. On peut même utiliser un algorithme de conversion. Un bon exemple en est fourni par la conversion des dates. Considérons l'information *24 décembre 1995*. Le

codage le moins recherché de cette date nécessite 16 octets soit 128 bits. Une première compression est possible en *24/12/1995* ou encore en *24121995* si on s'accorde à reconnaître que les deux premiers chiffres concernent le numéro du jour, les deux suivants celui du mois et les quatre derniers ceux de l'année. Si de plus on admet que l'on ne travaille que sur le siècle en cours, on peut encore comprimer en *241295*. On réduit ainsi le codage à 48 bits. Il est encore possible de l'améliorer. La compression en demi-octet permet d'arriver à 40 bits.

Si on choisit de coder, non les chiffres, mais les nombres, on constate que le numéro du jour peut être codé sur 5 bits (31 jours au plus) et le mois sur 4 bits. Deux octets laissent donc libres 7 bits pour le codage de l'année ce qui permet de travailler sur un intervalle de 127 ans. Le taux de compression est donc au total de 1 pour 8.

Compression par suppression

Ce n'est pas toujours le choix des codes qui va entraîner des économies. La codification des dates nous conduit naturellement vers les champs des enregistrements d'une base de données. Certaines informations n'occupent pas toujours toute la place qui leur est octroyée. Cet excès de place peut s'avérer conséquent pour une base de données de taille normale (plusieurs milliers d'enregistrements). Dans de tels cas, il est intéressant de mettre en place un système qui élimine les blancs indésirables. Un champ contenant une adresse à laquelle on a décidé d'allouer 40 caractères peut très bien être rempli en moyenne à 50%. Pour un fichier de cinq mille enregistrements, la taille de la mémoire inutilement occupée est donc de 100Ko.

Le codage relatif ou différentiel

Nous avons déjà fait allusion à cette méthode. Elle consiste à n'encoder qu'un message initial et les différences entre messages successifs. Elle est particulièrement efficace lorsque des données successives n'ont pas une forme simple et diffèrent peu ou pas du tout (des séries de mesures très précises, par exemple).

Comme nous l'avons déjà signalé, cette technique est utilisée en compression d'images, et en particulier en télécopie. Lorsqu'il s'agit d'une image balayée, les corrélations entre lignes successives sont loin d'être négligeables. Une ligne de points peut être enregistrée par la technique des répétitions et les suivantes par cette technique. Dans le domaine de la télécopie, la compression est une nécessité. En effet, un document de 500 caractères en transmission normale demande le transfert d'autant d'octets soit 4.000 bits. Si la vitesse de transmission est de 14.400 bps, le temps de transfert est nettement inférieur à la seconde. En utilisant le codage fax, le nombre de bits à transmettre est proche du million. Une transmission sans compression prendrait donc plus d'une minute!

L'algorithme de Huffman

Voici bien un algorithme qui tire profit de la fréquence des différents caractères dans un message. Prenons un exemple simple pour l'illustrer. Supposons que dans le fichier à compresser, seules les six premières lettres de l'alphabet apparaissent dans les fréquences suivantes:

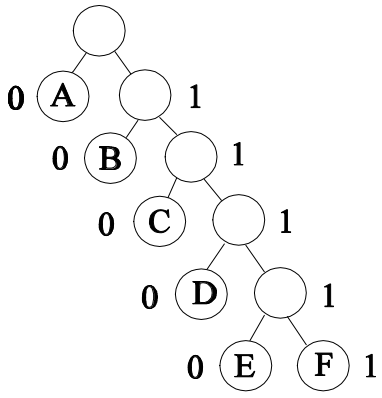
A	40%
B	29%
C	20%
D	8%
E	2%
F	1%

A	0
B	10
C	110
D	1110
E	11110
F	11111

Un arbre est créé sur base des caractères les moins fréquents, ici E et F et en incluant au niveau supérieur, la fréquence immédiatement supérieure. Chaque noeud de gauche reçoit la valeur 0 et chaque noeud de droite la valeur 1. Les caractères sont donc codés de la manière suivante.

Cette codification est intéressante dans la mesure où chaque caractère est codé sur moins d'un octet. De plus, les lettres les plus fréquentes ont les codes les plus courts. Par exemple, le mot *EFFACE* sera codé *111101111111111011011110*. Il occupera 24 bits soit trois octets au lieu de six.

Comment les lettres seront-elles isolées lors de la décompression? L'algorithme s'intéressera aux "0" qui marquent la fin du codage d'une lettre et aux séquences de cinq "un" consécutifs. Formellement, la séparation des groupes ne pose aucun problème. Notez au passage, cet exercice d'algorithmique qui, placé dans son contexte, peut s'avérer intéressant.



Inutile de préciser que l'arbre créé par l'algorithme est stocké avec le fichier compressé. Pour un fichier de petite taille, le stockage de l'arbre peut faire perdre davantage de place que celle qui est économisée par la compression.

La codification suivant l'algorithme de Huffman conduit à ce que l'on appelle un code à longueur variable. Parmi ces codes à longueur variable, vous connaissez le code Morse qui, à l'instar de l'algorithme de Huffman, attribue les codes les plus courts aux lettres (statistiquement) les plus courantes (en anglais): un point pour le E et une barre pour le T. Les autres caractères ont un code de deux à quatre symboles.

Codes à variation restreinte

A	00
B	01
C	1100
D	
E	1101
F	1110
G	1111

Nous l'avons déjà signalé, les ordinateurs sont orientés "mots"ce qui est un inconvénient pour un algorithme comme celui d'Huffman. On a donc mis au point des codes à variation restreinte dont voici un exemple: le code 2/4.
Il permet de coder sept caractères. On pourrait s'étonner de ne pas en voir un huitième codé *11*. C'est qu'il faut pouvoir décompresser les données. Ainsi, le mot *EFFACE* sera codé *11011110111000101101*.

Pour décompresser cette information, l'algorithme procédera de la sorte: si le début de la séquence est *11* (pas de correspondance), il s'agit d'un code 4, sinon c'est un code 2.

En examinant la séquence ci-dessus, vous pouvez sans peine, retrouver de manière formelle les lettres qui constituent l'information.

De la même façon, le code 5/10 permet de coder 63 caractères. Les 31 codes à 5 bits sont utilisés pour les caractères les plus fréquents.

Escamotage des champs vides d'une BD

Un peu à la façon des représentations topographiques, des "bits de présence" peuvent signaler à quels champs correspondent les données qui les suivent. Voici un exemple pour une base de données dont le nombre de champs des enregistrements est inférieur ou égal à 16.

0010001100010000	Information zone 3	Information zone 7	Information zone 8	Information zone 12

Un traitement formel permet de reconstituer la BD dans un état plus "classique". Avec le premier "mot" il est possible de déterminer combien de champs sont non vides et surtout lesquels. Les informations qui le suivent correspondent au contenu de ces champs.

Exploitation de l'ordre des données

Lorsqu'on dispose d'une base de données triée, il est possible d'utiliser cette caractéristique pour compresser. Imaginez un répertoire (annuaire) comprenant un nombre important de noms triés par ordre alphabétique. Le plus souvent, un nom reprend quelques caractères du nom précédent. Il n'est donc pas nécessaire de mémoriser une nouvelle fois ces caractères, mais seulement leur nombre. Voyez l'illustration qui suit.

Nom	Nom comprimé	
Adam	0	Adam
Alain	1	lain
Alardo	3	rdo
Albart	2	bart
Albert	3	ert
Albertini	6	ini

La reconstitution des informations est de nouveau purement “mécanique”. Cet exemple peut donner des idées d’exercices sur le traitement formel.

Compactage de données

Imaginons un fichier volumineux trié en utilisant une clé dont les valeurs ressemblent à celles qui suivent:

11101001011000
11101010110111
11101010111111
...

On constate que la distinction d’une valeur à la suivante se fait à partir d’un endroit précis. Ce qui suit est ce qu’il est permis d’appeler une “arrière-chaîne redondante” (RRS: rear redundant string). Son enregistrement n’est pas du tout nécessaire.

1110100
11101010110
11101010111
...

Pour des raisons déjà évoquées auparavant (lecture par “mots”), on a intérêt à choisir une longueur fixe. Ce peut être la longueur de la chaîne significative ayant la plus courte RRS.

11101001011
11101010110
11101010111
...

Les différentes techniques qui sont développées ici sont souvent combinées pour une même série de données, ce qui augmente encore le niveau de compression. Ainsi par exemple, on peut utiliser la technique de suppression des caractères répétitifs avant de mettre en oeuvre l’algorithme de Huffman.

L’algorithme de Lempel Ziv

Cet algorithme, encore appelé LZW (car modifié par Welch) est basé sur un système d’extension de l’alphabet. Il crée une table dont les 256 premières cellules contiennent les caractères de l’ASCII étendu. Cette table va grandir au fur et à mesure du parcours du texte, les cellules suivantes étant remplies par les chaînes de caractères non encore rencontrées auparavant. Chacune de ces chaînes est donc codée, ce qui engendre une belle économie d’octets sur des fichiers de données volumineux. Donnons un exemple. Soit à compresser le texte:

DERRIERE LE MUR, UN MURMURE...

Au départ, la table est dans l’état suivant:

253	254	255	256	257	258	259	260	261	262	263	264	265	266	267	268	269	270	271	272	273		
*	*	*																				

Les étoiles indiquent la présence de symboles correspondant aux codes 253, 254 et 255 et qui sont utilisés diversement (ASCII étendu) par les constructeurs. La table est vide à partir de la cellule 256. Le compresseur examine d'abord les deux premiers caractères. La chaîne DE ne figure pas dans la table. Elle est stockée dans la cellule 256. Le caractère D est envoyé dans le fichier compressé. Le caractère E restant est associé au caractère suivant pour former la chaîne ER. Celle-ci ne figure pas dans la table. Elle est stockée dans la cellule 257. Le caractère E est envoyé dans le fichier compressé. Le caractère R restant est associé au caractère suivant pour former la chaîne RR. Celle-ci ne figure pas...

Après un certain nombre d'opérations, voici l'état de la table:

253	254	255	256	257	258	259	260	261	262	263	264	265	266	267	268	269	270	271	272	273		
*	*	*	DE	ER	RR	RI	IE															

Les codes contenus dans le fichier compressé sont (en décimal) 68 69 82 82 73
Reprenons les opérations à ce moment. Le caractère restant E est associé au caractère suivant pour former la chaîne ER. Celle-ci figure **déjà** dans la table. Elle est associée au caractère suivant pour former la chaîne ERE. Celle-ci ne figure pas dans la table. Elle est stockée dans la cellule 261. La chaîne ER est envoyée dans le fichier compressé.

Les codes contenus dans le fichier compressé sont (en décimal) 68 69 82 82 73 **257**

Le caractère restant E est associé...

Après un certain nombre d'opérations, voici l'état de la table:

253	254	255	256	257	258	259	260	261	262	263	264	265	266	267	268	269	270	271	272	273		
*	*	*	DE	ER	RR	RI	IE	ERE	E_	_L	LE	E_M	MU	UR	R,	,_	_U	UN	N_	_M		

Les codes contenus dans le fichier compressé sont (en décimal) 68 69 82 82 73 **257** 69 32 76 69 **262** 77 85 82 44 32 85 78 32

Reprenons les opérations à ce moment. Le caractère restant M est associé au caractère suivant pour former la chaîne MU. Celle-ci figure déjà dans la table. Elle est associée au caractère suivant pour former la chaîne MUR. Celle-ci ne figure pas dans la table. Elle est stockée dans la cellule 274. La chaîne MU est envoyée dans le fichier compressé. Le caractère R restant est associé au caractère suivant pour former la chaîne RM. Celle-ci ne figure pas dans la table. Elle est stockée dans la cellule 275. Le caractère R est envoyé dans le fichier compressé. Le caractère restant M est associé au caractère suivant pour former la chaîne MU. Celle-ci figure déjà dans la table. Elle est associée au caractère suivant pour former la chaîne MUR. Celle-ci figure déjà dans la table. Elle est associée au caractère suivant pour former la chaîne MURE. Celle-ci ne figure pas dans la table. Elle est stockée dans la cellule 276.

A ce stade, voici l'état de la table:

256	257	258	259	260	261	262	263	264	265	266	267	268	269	270	271	272	273	274	275	276	277	278
DE	ER	RR	RI	IE	ERE	E_	_L	LE	E_M	MU	UR	R,	,_	_U	UN	N_	_M	MUR	RM	MURE		

Les codes contenus dans le fichier compressé sont (en décimal) 68 69 82 82 73 **257** 69 32 76 69 **262** 77 85 82 44 32 85 78 32 **266** 82 **274**

Plus on avance dans le parcours du texte, plus on retrouve de chaînes déjà mémorisées. De la sorte, la longueur des nouvelles chaînes à stocker augmente, ce qui assure des économies de plus en plus importantes. En admettant que le nombre de chaînes stockées n'excède pas 4096, le codage de la chaîne MURE demande seulement 12 bits contre 4 octets normalement. Le gain n'est pas encore sensible pour un texte aussi court, mais il va s'amplifier considérablement avec l'apparition de chaînes plus longues dans la table.

Le principe de décompression résulte d'une simple consultation de la table. Des améliorations sont possibles, ce que prévoit l'algorithme: élimination des chaînes courtes (principalement les lettres) et des chaînes qui apparaissent tout le temps sous des formes plus longues déjà codées (par exemple, suppression de la chaîne AB si elle apparaît toujours dans les chaînes ABA et ABE qui sont déjà codées).

Un petit mot de la compression des données graphiques et sonores

Les images sont généralement gourmandes en espace mémoire. On s'est donc rapidement intéressé à des techniques permettant d'éliminer les données redondantes d'une image (celles qui n'affectent pas sa qualité pour la tâche envisagée). Ces techniques utilisent des algorithmes spécialisés qui peuvent réduire jusqu'à cent fois le volume initial des données. Des standards de compression existent comme le *JPEG* pour l'image fixe et le *MPEG* pour l'image animée. Ils sont fort utiles dans la création de compositions multimédia, car la capacité d'un disque compact n'est pas illimitée.

Le JPEG

JPEG désigne un groupe, le *Joint Photographic Experts Group*, qui a mis au point des algorithmes de compressions d'images, avec ou sans perte d'information. Il est clair que les algorithmes avec perte autorisent des coefficients de compression plus élevés. On peut atteindre un coefficient de compression de 100. Cependant, l'image est sérieusement dégradée. Un coefficient de 20 donne de très bons résultats (image très faiblement altérée). Un des principes est de prendre en compte toutes les répétitions (des pixels proches ont souvent les mêmes caractéristiques de couleur) ou de mémoriser uniquement les différences d'un pixel à l'autre. De la sorte, l'image peut être reconstituée. Le JPEG est utilisé en vidéo numérique, de même que d'autres techniques de compression telles la réduction du nombre d'images par seconde et la réduction de l'espace écran utilisé (jeux vidéo) mais les exigences de ce média lui feront sans doute préférer le MPEG.

Le MPEG

Le *Moving Pictures Experts Group* a mis au point un algorithme qui, comme le JPEG, élimine les informations redondantes. Une amélioration toutefois: en plus d'une compression intra-image (comme le fait JPEG), il y a également une compression interimage. MPEG n'enregistre que les changements d'une image à l'autre. Dans les jeux vidéo, par exemple, peu d'objets se déplacent; l'arrière-plan varie souvent très peu.

Bibliographie

H. Nussbaumer *Téléinformatique III* Presses Polytechniques Universitaires Romandes Lausanne 1991

A. Strohmeier *Le matériel informatique: concepts et principes* Presses Polytechniques Universitaires Romandes Lausanne 1986

L.Clément *Opérateurs logiques, combinatoires et séquentiels* De Boeck Wesmael 1987

P.-A. Goupille *Technologie des ordinateurs* (2ème édition) Masson Paris1993

E. Holsinger *Le multimédia... Comment ça marche?* Dunod Paris 1994

Les rédacteurs des éditions Time-Life *Le décodeur d'énigmes* Le monde des ordinateurs Time-Life Amsterdam 1989

A. Ralston, E. Reilly *Encyclopedia of Computer Science* (Third edition) Chapman & Hall London 1993

M. Freihof, I. Kurten *MS-DOS 6.2* Micro Application Paris 1993

J. De Schryver *Dictionnaire de la Micro 93* Micro Application Paris 1993

J.-P. Meesters, Virga *Le Nouveau Dictionnaire Marabout de la Micro-informatique* Les Nouvelles Editions Marabout Verviers 1990

P. Morvan *Dictionnaire de l'informatique* Larousse Paris 1981

B. Pfaffenberger *Que's Computer User's Dictionary* Que Corporation Carmel Indiana 1990

M. Rousseau *Soft & Micro n°87* Juillet 1992

H. Lemberg *PC Expert n°17* Septembre 1993

F. Sass, E. Vandeput *Fiche pédagogique n°8.1: DAO, les actions de base* CeFIS, FPE, ICAFOC Namur 1994

Annexe 1

Comment une machine peut-elle additionner?

Le principe du calcul binaire s'applique aux systèmes à deux états. Dans cette optique, 0 et 1 ne représentent que des symboles qui sont eux-mêmes les représentants de l'état binaire d'une situation:

- oui, non
- blanc, noir
- vrai, faux
- courant, pas courant
- allumé, éteint
- signal basse tension, signal haute tension...

En ce qui concerne l'ordinateur, les circuits électroniques qui le composent sont susceptibles d'être soumis à des tensions basses ou hautes. Celles-ci provoquent le passage (ou non) d'un courant dans les circuits. L'entrée d'une information élémentaire correspond donc à l'émission, par un périphérique source, d'un signal de basse ou haute tension et la sortie d'une information élémentaire à la réception d'un tel signal. Ainsi, la question *"Comment une machine peut-elle additionner?"* devient-elle *"Comment peut-on construire des circuits électroniques permettant de réaliser une addition?"*

Le principe consiste à assembler des portes logiques. Celles-ci, constituées de transistors, sont telles qu'elles reçoivent en entrée, un ou deux signaux et en fournissent un en sortie. Les trois portes logiques de base sont la porte ET, la porte OU et la porte NON. Les tableaux ci-dessous montrent quelles réponses chacune de ces portes fournit en regard des informations reçues en entrée.

ET		
Entrée 1	Entrée 2	Sortie
1	1	1
1	0	0
0	1	0
0	0	0

OU		
Entrée 1	Entrée 2	Sortie
1	1	1
1	0	1
0	1	1
0	0	0

NON	
Entrée	Sortie
1	0
0	1

Ces réponses sont directement inspirées du calcul booléen (*George BOOLE*, mathématicien anglais 1815-1864). Ce calcul, dont les règles de base sont synthétisées dans les tableaux qui précèdent, est lié à une logique du tiers exclus (donc binaire). Les propositions énoncées dans une telle logique sont soit vraies, soit fausses. Il n'y a pas d'autres possibilités. Dans le contexte informatique qui nous occupe, ce n'est pas de propositions qu'il s'agit, mais de circuits électroniques n'ayant que deux états

possibles. Examinons comment peut se faire l'assemblage dans le cas simple d'une addition de deux quartets. Soit à réaliser l'addition binaire suivante:

```
  0 1 0 1
+ 0 1 1 1
---------
  1 1 0 0
```

Constatons d'abord que l'addition des deux premiers chiffres binaires, si elle peut provoquer un report, ne doit pas tenir compte d'un report précédent. Elle peut être réalisée grâce à un circuit simple appelé "demi-additionneur".

L'addition des autres chiffres binaires doit tenir compte de ce report éventuel. Le circuit correspondant est plus complexe. On parle d'additionneur complet (voir schémas en annexe). Pour créer un circuit complexe permettant de réaliser l'addition de deux nombres binaires de quatre bits, il suffit de connecter un demi-additionneur et trois additionneurs complets par l'intermédiaire des reports.

Construction du demi-additionneur

Le demi-additionneur doit fournir comme résultat 0 si les deux données sont identiques et 1 si une des deux données seulement est nulle. De plus, un report doit être enregistré. Les tables de vérité de la logique booléenne nous demandent de trouver une expression qui fournira, en résultat, la troisième colonne du tableau suivant:

p	q	?
1	1	0
1	0	1
0	1	1
0	0	0

Cette expression doit mettre en jeu *p*, *q* qui symbolisent deux entrées élémentaires et les opérateurs logiques de base qui sont: *ET*, *OU* (non exclusif) et *NON*, dont on connaît les tables de vérité, et qui représentent les portes logiques disponibles. On voit que le résultat est proche de *p ou q*, mais que le cas où *p* et *q* valent 1 tous les deux est à rejeter. Une expression qui convient est donc:

(p OU q) ET NON(p ET q)

p	q	p OU q	p ET q	NON(p ET q)	(p OU q) ET NON(p ET q)
1	1	1	1	0	0
1	0	1	0	1	1
0	1	1	0	1	1
0	0	0	0	1	0

Il s'agit du *OU* exclusif (l'un ou l'autre, mais pas les deux). Quant au report, il est disponible au niveau de *p ET q*. On réalise donc simplement un demi-additionneur au moyen de deux portes *ET*, une porte *OU* et une porte *NON*.

Construction de l'additionneur complet

L'additionneur complet doit prendre en compte une troisième donnée élémentaire, le report éventuel issu de l'addition précédente. Ici, les trois entrées sont symbolisées par *p*, *q* et *r*.

Règlons d'abord le problème du report. Une façon de faire est de considérer qu'il y a report si deux des trois données au moins sont non nulles. Ce qui donne par exemple *(r ET p) OU ((r ET q) OU (p ET q))*. C'est la partie inférieure gauche du schéma.

Pour ce qui est de l'addition proprement dite, le résultat est 1 si une seule des trois données est non nulle ou si elles le sont toutes. Cela donne par exemple:

(p ET (NON (p ET r))) OU (r ET (NON(p ET r))) qui pourrait se traduire par *"p sans r ou r sans p"*.

p	q	r	p ET r	NON(p ET r)	(p ET (NON (p ET r)))	(r ET (NON(p ET r)))	(p ET (NON (p ET r))) OU (r ET (NON(p ET r)))
1	1	1	1	0	0	0	0
1	1	0	0	1	1	0	1
1	0	1	1	0	0	0	0
1	0	0	0	1	1	0	1
0	1	1	0	1	0	1	1
0	1	0	0	1	0	0	0
0	0	1	0	1	0	1	1
0	0	0	0	1	0	0	0

Cette même expression combinée de la même manière avec *q* donne:

((p sans r ou r sans p) sans q) ou (q sans (p sans r ou r sans p))

p	q	r	p sans r ou r sans p	(p sans r ou r sans p) sans q	q sans (p sans r ou r sans p)	((p sans r ou r sans p) sans q) ou (q sans (p sans r ou r sans p))
1	1	1	0	0	1	1
1	1	0	1	0	0	0
1	0	1	0	0	0	0
1	0	0	1	1	0	1
0	1	1	1	0	0	0
0	1	0	0	0	1	1
0	0	1	1	1	0	1
0	0	0	0	0	0	0

Le début de la proposition traduit *"une seule des trois données est non nulle"*. La fin de la proposition traduit *"q est non nulle et p et r ont les mêmes valeurs"*, ce qui inclut le cas où p, q et r sont non nulles.

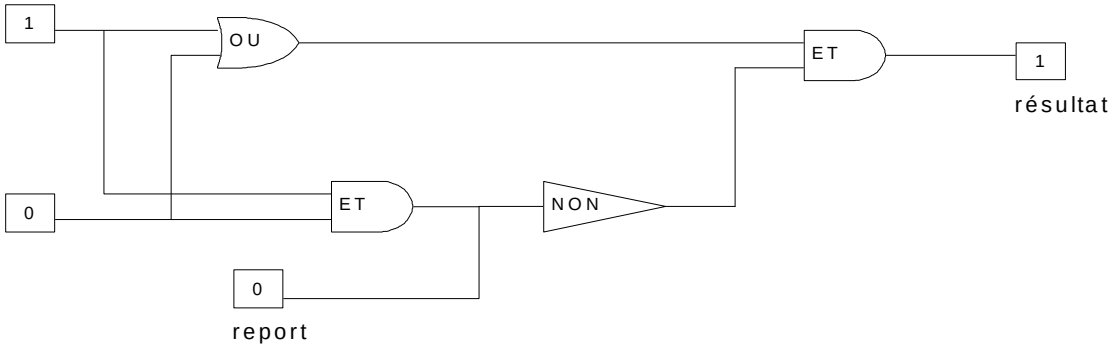
En observant les valeurs de p, q et r, on constate que les résultats obtenus correspondent bien à l'attente. Par exemple, trois "1" donnent "1", deux "1" et un "0" donnent "0" etc.

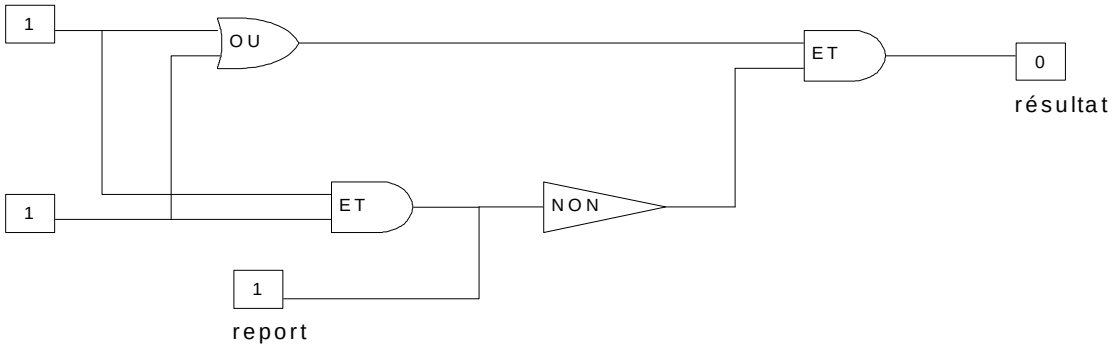
Il est à remarquer que d'autres combinaisons sont possibles. Ainsi, *p ET (NON(p ET r))* est équivalent à *p ET (NON r)*. On pourrait donc imaginer d'autres circuits.

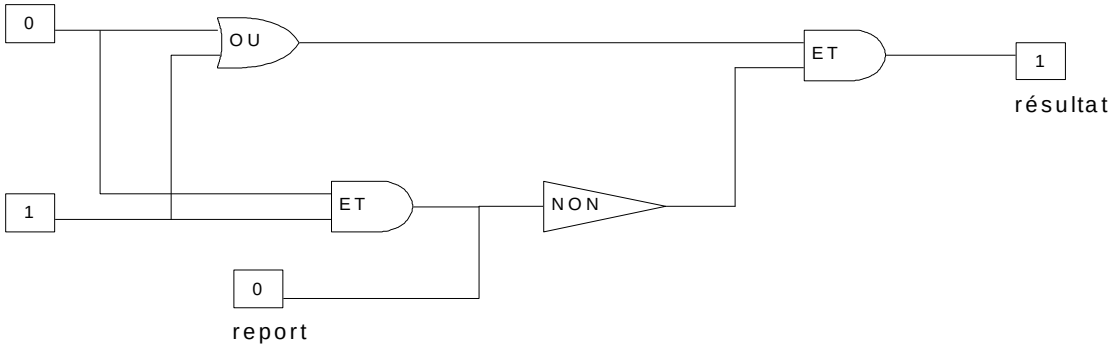
Annexe 2

Demi-additionneur binaire

pas de report au départ,
report possible à l'arrivée

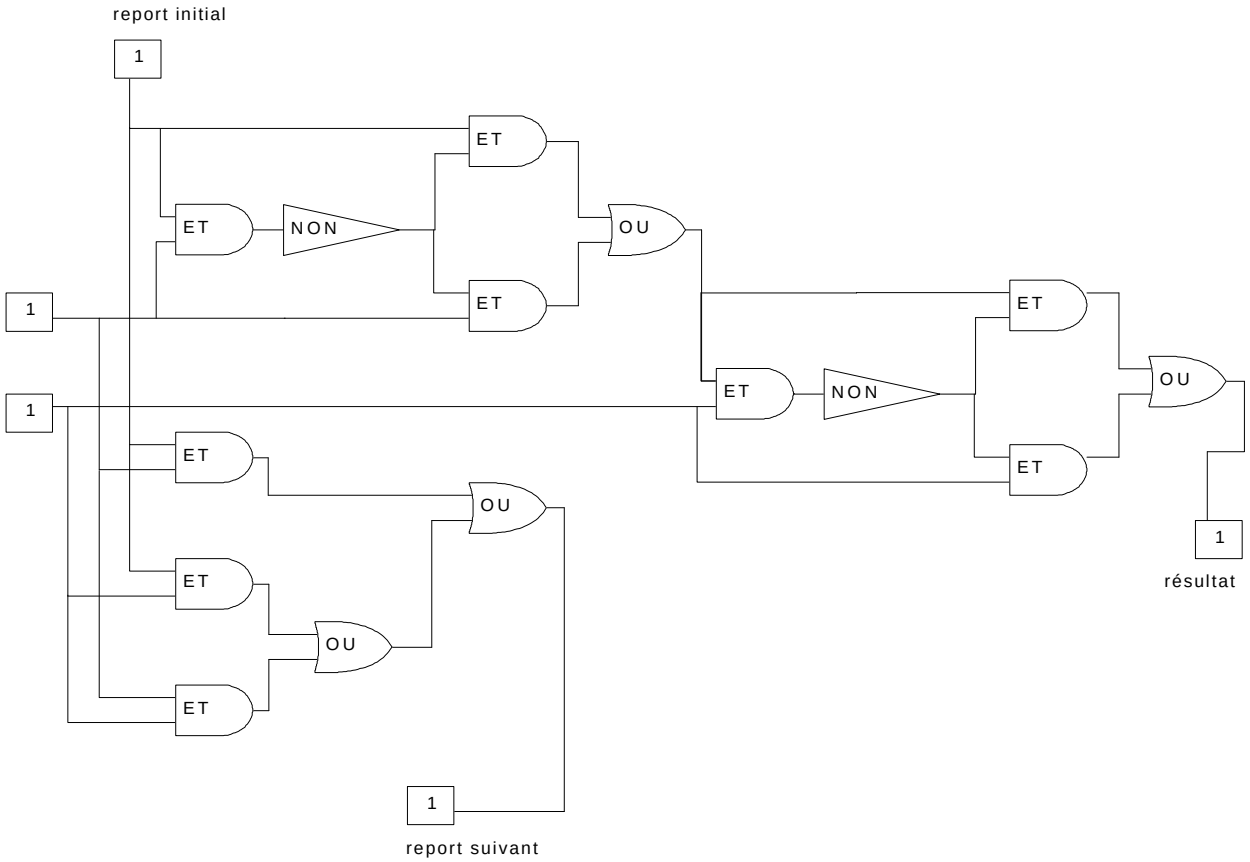






Annexe 3

Additionneur binaire complet



Notre exemple:

```
  1 1
  1
+1
----
  1
```

Annexe 4

Code EBCDIC

		00				01				10				11			
		0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
00	0					SP	&	_									0
	1							/		a	j			A	J		1
	2									b	k	s		B	K	S	2
	3									c	l	t		C	L	T	3
01	4									d	m	u		D	M	U	4
	5									e	n	v		E	N	V	5
	6									f	o	w		F	O	W	6
	7									g	p	x		G	P	X	7
10	8									h	q	y		H	Q	Y	8
	9									i	r	z		I	R	Z	9
	A					¢	!		:								
	B						$	,	#								
11	C					<	*	%	@								
	D					(	)	-	‘								
	E					+	;	>	=								
	F					|	¬	?	·								

La lettre D est codée C4 en hexadécimal soit 11000100 en binaire. Le chiffre 7 est codé F7 en hexadécimal, soit 11110111 en binaire. Les codes commençant par 1011 sont inemployés.