LOGIQUE & CALCUL

DES SUITES EXOTIQUES POUR ÉCRIRE LES NOMBRES

Aujourd'hui encore, on perfectionne les systèmes de représentation des nombres entiers, et l'on en découvre même de nouveaux.

L'AUTEUR

JEAN-PAUL DELAHAYE
professeur émérite à l'université de Lille et chercheur au laboratoire Cristal (Centre de recherche en informatique, signal et automatique de Lille)

Il ne fait aucun doute que deux des progrès majeurs de l'histoire des mathématiques sont l'invention du système de numération positionnelle, qui permet de représenter tous les entiers à l'aide de symboles écrits, et la mise au point d'algorithmes généraux pour manipuler ces écritures et effectuer additions, soustractions, multiplications, etc. Aujourd'hui, le système de numération presque universellement adopté est la numération positionnelle dite «standard», en base 10 pour les humains et en base 2 pour les ordinateurs. Cependant, il ne s'agit pas de l'unique possibilité ! Bien d'autres systèmes de numération positionnelle ont été proposés pour représenter les entiers, et méritent qu'on s'y intéresse – ce qui se fait d'ailleurs en informatique. On perfectionne leur maîtrise grâce à de nouveaux algorithmes pour les opérations de base et, plus étonnant encore, on conçoit à l'heure actuelle des systèmes de numération jusqu'alors inconnus.

Au cours du petit voyage que je vous propose dans le monde des systèmes de numérations utilisables par les technologies du numérique, nous évoquerons la mise au point, toute récente, d'algorithmes permettant d'additionner et de multiplier des entiers représentés en «numération de Zeckendorf»; nous reviendrons sur l'amusant système de Gros-Gray et sur le fascinant «ternaire équilibré»; puis nous décrirons un nouveau système de numération positionnelle, mis au point en 2024 par Péter Hajnal, professeur à l'université de Szeged, en Hongrie. Nous nous concentrerons sur la représentation des nombres entiers, mais tout ce qui sera évoqué s'étend sans grande difficulté aux nombres réels.

NUMÉRATION ET ARITHMÉTIQUE

Un système de numération positionnelle repose toujours sur un résultat d'arithmétique, qui garantit l'existence (et, le plus souvent, l'unicité) de la représentation de tout nombre entier dans ce système. C'est ce résultat d'arithmétique qui déterminera les méthodes pour comparer les nombres et mener des opérations de base, comme l'addition ou la multiplication, dans ce système.

Le système de numération binaire, par exemple, repose sur le théorème assurant que tout entier à partir de 1 s'écrit de façon unique comme une somme de puissances de 2, chacune prise au plus une fois. La règle fixant l'écriture est que si 2^i est présent dans la décomposition d'un entier, sa représentation en base 2 comporte un 1 en position $i+1$ à partir de la droite, et un 0 en cette position si 2^i est absent de la décomposition. Ainsi, de la décomposition $13 = 8+4+1 = 2^3+2^2+2^0$, on tire la représentation $13_{\text{base 10}} = 1101_{\text{base 2}}$.

De la même manière, le système de numération en base 10, que tout le monde pratique au quotidien, repose sur le théorème assurant que tout entier n se décompose de façon unique sous la forme: $n = a_k \times 10^k + a_{k-1} \times 10^{k-1} + \ldots + a_1 \times 10^1 + a_0 \times 10^0$, où chaque a_i est un entier entre 0 et 9, et a_k est non nul. L'écriture de n en base 10 est alors $a_k a_{k-1} \ldots a_1 a_{0\ \text{base 10}}$.

Jean-Paul Delahaye a également publié : **Au-delà du Bitcoin** (Dunod, 2022).

Les notations «base 2» et «base 10», que nous avons placées en indice, sont en général omises. Dans la suite de cet article, ce sera effectivement le cas pour la base 10, mais, comme nous passerons sans cesse d'un système à un autre, pour les autres systèmes de numération nous indiquerons soigneusement en indice celui qui est utilisé.

Les systèmes de numération positionnelle décimal et binaire, bien sûr, se généralisent pour donner les numérations positionnelles «en base b» pour tout entier $b>1$. Avec ces systèmes, on dispose d'une représentation assez concise de chaque entier, aussi grand soit-il, n'utilisant que b symboles différents, appelés «les chiffres de la base b». Pour écrire un entier n en base b, il faut exactement $[1+log(n)/log(b)]$ symboles, où la notation $[r]$ désigne la partie entière de r – c'est-à-dire le plus grand entier inférieur ou égal à r. Les algorithmes que nous avons appris à l'école pour additionner, soustraire, multiplier, etc. les entiers représentés en base 10 ont fait de la numération positionnelle décimale un outil parfait pour le commerce, les artisans, les percepteurs, etc. Ces outils ont sans mal été adaptés, par la suite, pour un usage informatique.

Pourtant, pour mesurer les durées, on utilise un système partiellement sexagésimal (c'est-à-dire en base 60), hérité des Sumériens du troisième millénaire avant notre ère. La numération utilisée pour les durées est même plus complexe que la numération positionnelle en base 60, puisque, s'il y a bien 60 secondes par minute et 60 minutes par heure, il y a en revanche 24 heures dans une journée, et 365 jours par an (ou 366 les années bissextiles). Cela signifie que notre système de représentation du temps est à «pas variable» 60-60-24-365... sans compter que, pour représenter des durées inférieures à une seconde, on revient cette fois à la base 10: dixièmes, centièmes, millièmes de seconde, etc.

SUITE DE FIBONACCI

Il existe bien d'autres systèmes de numération exotiques, moins connus que les précédents. Parmi les beaux systèmes positionnels non standard, il faut mentionner la «numération de Zeckendorf», du nom du mathématicien belge Édouard Zeckendorf (1901-1983), auteur du théorème d'arithmétique sur lequel repose le système. Ce résultat assure que tout entier s'écrit

NUMÉRATIONS DE POSITION

1 **Il existe un procédé général permettant de créer de nouveaux systèmes de numération positionnelle. Donnons-nous une suite de nombres entiers x_1, x_2, x_3, ... strictement croissante et commençant par $x_1 = 1$. On convient de représenter tout entier $n \geq 1$ par $a_k a_{k-1} ... a_2 a_1$ si et seulement si n se décompose sous la forme : $n = a_k \times x_k + a_{k-1} \times x_{k-1} + ... + a_1 \times x_1$. Le fait que la suite des (x_k) soit croissante et commence par 1 assure qu'il est toujours possible de décomposer un entier de cette manière. Il se peut, cependant, que cette décomposition ne soit pas unique. C'est par exemple le cas pour $n = 60$ et pour la suite $x_k = (3^k - 2)$:**
$60 = 2 \times 25 + 1 \times 7 + 3 \times 1 =$
$1 \times 25 + 5 \times 7 + 0 \times 1 =$
$8 \times 7 + 4 \times 1$.

On obtiendrait donc à la fois les représentations 213, 150 et 84, pour l'entier qui est noté 60 en base 10. Le système de numération ne serait donc pas très satisfaisant ! Cependant parmi ces différentes représentations, il y en a une plus grande que les autres au sens lexicographique – qui est le classement du dictionnaire, obtenu en lisant le nombre de gauche à droite (quitte à ajouter des zéros à gauche du nombre le plus court, pour comparer deux nombres de la même longueur). C'est cette représentation qui servira pour la numération positionnelle en base (x_k). En pratique, pour l'obtenir, on recherche le plus grand entier x_k inférieur ou égal à n. L'indice k de ce x_k fixe la longueur de la représentation de n. On soustrait alors ce x_k à n, et on recommence tant que le reste est positif.

Le nombre de soustractions effectuées fournit le coefficient a_k. On soustrait ensuite x_{k-1} au reste laissé par l'opération précédente, et on répète l'opération autant de fois que possible. Le nombre de soustractions effectuées fournit le coefficient a_{k-1}. On réitère l'opération pour trouver a_{k-2}, etc. Pour $n = 60$ et $x_k = (3^k - 2)$, on obtient bien que $60 = 213_{num\text{-}x}$. Il reste encore à choisir des symboles pour représenter les a_i. Ils ne seront en nombre fini que si pour tout k, $x_{k+1} / x_k < C$, où C est une constante. Cette méthode générale fournit le système décimal usuel avec la suite $x_k = 10^{k-1}$. Elle donne plus généralement la numération positionnelle en base b avec $x_k = b^{k-1}$. On retrouve aussi la numération positionnelle de Zeckendorf avec $x_i = f_{i+1}$, où (f_i) est la suite de Fibonacci ($f_1 = 1$; $f_2 = 1$; $f_3 = 2$; $f_4 = 3$; $f_5 = 5$; etc.).

de façon unique comme une somme de nombres de Fibonacci distincts et non consécutifs.

Rappelons que les nombres de Fibonacci sont définis par : $f_1=1$; $f_2=1$; $f_n=f_{n-1}+f_{n-2}$ si $n>2$. Les premiers termes de la suite sont donc : $f_1=1$; $f_2=1$; $f_3=2$; $f_4=3$; $f_5=5$; $f_6=8$; $f_7=13$; $f_8=21$; $f_9=34$; $f_{10}=55$. La démonstration du théorème de Zeckendorf tient en quelques lignes : si n est un entier quelconque supérieur ou égal à 1, on recherche le plus grand nombre de Fibonacci inférieur ou égal à n. On soustrait ce nombre de Fibonacci à n, et on recommence la même opération avec le résultat de la soustraction. On répète le processus jusqu'à obtenir 0. Cela décompose bien n en une somme de nombres de Fibonacci différents, car on vérifie facilement que pour tout $k>2$, $f_k \leq 2 \times f_{k-1}$. Il ne peut pas y avoir, au cours du procédé décrit, de soustractions successives de deux nombres de Fibonacci consécutifs f_{k-1} et f_{k-2}, car leur somme, qui vaut f_k, aurait été soustraite avant d'arriver à f_{k-1}.

Pour obtenir la représentation en numération de Zeckendorf d'un entier n, on procède de la même manière que pour la base 2 : on place un 1 en position i à partir de la droite si f_{i+1} est présent dans la décomposition de n donnée par le théorème de Zeckendorf, et un 0 sinon. On impose bien sûr à la représentation de commencer par un 1. Attention au décalage des indices : ici, on ne prend en compte que les f_i à partir de f_2. On obtient donc, par exemple, que $12=8+3+1=f_6+f_4+f_2$, donc que $12=10101_{zeck}$. De même, $41=34+5+2=f_9+f_5+f_3$ donc $41=10001010_{zeck}$.

Le fait que cette décomposition n'utilise pas deux nombres de Fibonacci consécutifs signifie que la représentation de Zeckendorf d'un entier ne comporte jamais deux 1 côte à côte. L'intérêt de ce système de numération pour l'informatique est qu'on en tire immédiatement une représentation « autodélimitée » des entiers, en faisant précéder chaque nombre d'un 1 : $12=10101_{zeck}=110101_{auto}$ et $41=10001010_{zeck}=110001010_{auto}$. Dans cette représentation « autodélimitée », il ne peut y avoir deux 1 consécutifs qu'au début d'un nombre : la représentation comporte en elle-même l'indication de son début – d'où son nom ! Cela permet d'écrire une suite de nombres en les collant les uns aux autres, sans symbole particulier de séparation : les doubles 1 dans une suite de nombres ainsi notés servent de séparateurs et permettent donc d'isoler les entiers utilisés et de les identifier. Une telle juxtaposition ne serait pas possible avec la représentation binaire usuelle. Le couple constitué des nombres 41 et 12 accolés donne donc, en représentation autodélimitée : 110001010110101_{auto}. S'il y a k fois un

2 ADDITION RAPIDE EN NUMÉRATION DE ZECKENDORF

En numération de Zeckendorf, les entiers strictement positifs sont ramenés à des sommes de nombres de Fibonacci distincts, non consécutifs, à partir de f_2. Cette notation n'utilise que les chiffres 0 et 1. Rappelons que la suite de Fibonacci est définie par : $f_1 = 1$; $f_2 = 1$ et $f_n = f_{n-1} + f_{n-2}$ si $n > 2$. Ainsi, l'entier noté $a_p a_{p-1} \ldots a_1 a_0{}_{zeck}$ représente le nombre $a_p \times f_{p+2} + a_{p-1} \times f_{p+1} + \ldots + a_1 \times f_3 + a_0$.

Christiane Frougny, de l'Institut de recherche en informatique fondamentale, à Paris, a découvert un étonnant algorithme permettant d'additionner deux nombres en numération de Zeckendorf sans passer par une autre représentation. Sa méthode est détaillée ci-dessous sur un exemple.

Considérons la somme 114 + 142, c'est-à-dire : $1001000101_{zeck} + 1010101001_{zeck}$. On opère d'abord une addition chiffre par chiffre, en utilisant le symbole 2 pour la somme 1 + 1. Cela fournit la somme recherchée dans ce qu'on peut appeler une représentation « Zeckendorf étendue » (zeck-e) : $2011101102_{zeck\text{-}e}$. Le problème revient alors à transformer cette représentation en représentation de Zeckendorf. Pour cela, on fait précéder la représentation Zeckendorf étendue d'un zéro, et on la parcourt trois fois, comme expliqué ci-dessous.

Le premier parcours fait se déplacer case par case, de la gauche vers la droite, une fenêtre de largeur quatre, et modifie la représentation en appliquant, lorsque c'est possible, les quatre règles r_1, r_2, r_3 et r_4 détaillées ci-dessous. On convient de noter x' pour désigner $x + 1$.

(r_1) $020x \rightarrow 100x'$
(r_2) $030x \rightarrow 110x'$
(r_3) $021x \rightarrow 110x$
(r_4) $012x \rightarrow 101x$.

La présence du 3, qui n'existait pas dans la représentation Zeckendorf étendue, doit à présent être envisagée à cause de l'utilisation du x', qui peut l'introduire. Dans notre exemple, le calcul donne :

$02011101102_{zeck\text{-}e} \rightarrow 10021101102_{zeck\text{-}e}$ (utilisation de la règle r_1)
$10021101102_{zeck\text{-}e}$ (aucune règle applicable)
$10021101102_{zeck\text{-}e} \rightarrow 10110101102_{zeck\text{-}e}$ (utilisation de règle r_3)
$10110101102_{zeck\text{-}e}$ (aucune règle applicable)
$10110101102_{zeck\text{-}e}$ (aucune règle applicable)
$10110101102_{zeck\text{-}e}$ (aucune règle applicable)
$10110101102_{zeck\text{-}e}$ (aucune règle applicable)
$10110101102_{zeck\text{-}e}$ (aucune règle applicable)

Arrivé au dernier quadruplet, on remplace les 2 et les 3 qui pourraient rester en utilisant les propriétés de la suite de Fibonacci (f_n). En effet, $f_2 = 1$, $f_3 = 2$, $f_4 = 3$ et $f_5 = 5$, ce qui permet, dans notre exemple, de remplacer le 02 final (qui représente $2 \times f_2 = 2$) par 10 (qui représente $f_3 = 2$). On obtient ainsi la notation $10110101110_{zeck\text{-}e}$.

Dans le cas général cette phase du calcul produit une suite débarrassée des 2 et des 3, mais laissant par endroits deux 1 consécutifs – ce qui signifie que la représentation obtenue n'est pas une représentation de Zeckendorf.

Pour les deuxième et troisième parcours, on ajoute de nouveau un 0 devant la représentation, et on fait passer une fenêtre de largeur trois qui se déplace case par case, d'abord de droite à gauche puis de gauche à droite, en remplaçant à chaque fois que c'est possible 011 par 100.

Dans notre exemple, lors du passage de droite à gauche, quatre substitutions sont effectuées :

$01011010111 0_{zeck\text{-}e} \rightarrow 01011011001 0_{zeck\text{-}e}$
$01011011001 0_{zeck\text{-}e} \rightarrow 01011100001 0_{zeck\text{-}e}$
$01011100001 0_{zeck\text{-}e} \rightarrow 01110010000 10_{zeck\text{-}e}$
$01100100001 0_{zeck\text{-}e} \rightarrow 10000100001 0_{zeck\text{-}e}$

Lors du passage de gauche à droite, il n'y a ici aucune substitution nouvelle.

La représentation finalement obtenue est donc 100001000010_{zeck}, qui correspond bien à l'entier 256 = 114 + 142.

double 1, cela signifie qu'on représente une suite de k entiers. Cette propriété est utilisée dans certains algorithmes de compression de données ainsi qu'en cryptographie.

Bien sûr, faire une addition ou une multiplication entre deux nombres écrits en représentation de Zeckendorf ou autodélimitée exige de nouveaux algorithmes. Une première solution consiste à passer par la représentation binaire ou la représentation en base b, pour lesquelles on connaît bien les algorithmes correspondants, puis à revenir à la représentation de Zeckendorf. Cette solution n'est cependant pas très élégante, et est coûteuse en calculs et en mémoire.

Pour l'addition, une autre méthode a été présentée dans un article de 2013 par une équipe réunie autour de Connor Ahlbach, aujourd'hui professeur à l'université du Texas, aux États-Unis. Les chercheurs ont exposé un algorithme d'addition préalablement découvert par Christiane Frougny, qui travaille maintenant à l'Institut de recherche en informatique fondamentale, à Paris. Cet algorithme fournit la représentation de Zeckendorf de la somme de deux nombres A et B écrits en numération de Zeckendorf, sans passer par la représentation binaire ni une représentation en base b. Il fonctionne en un temps proportionnel au maximum du nombre de chiffres de la représentation de Zeckendorf de A et B. On dispose donc d'une méthode qui est, à peu de chose près, aussi efficace pour additionner des nombres en numération de Zeckendorf que celle que nous pratiquons habituellement avec les représentations usuelles décimale ou binaire (*voir l'encadré 2*).

Pour rendre le système véritablement fonctionnel, il fallait obtenir un algorithme similaire pour la multiplication. Ce n'est qu'en 2021 que le chercheur indépendant polonais Tomasz Idziaszek a proposé un tel programme de multiplication des nombres en numération de Zeckendorf. Sa méthode, trop complexe pour être détaillée ici, fonctionne en temps proportionnel à $k \times \log(k)$, où k est le nombre de chiffres du plus long des entiers qu'on veut multiplier. Ce résultat correspond, à peu de chose près, à ce qu'on sait faire de mieux avec la numération positionnelle en base 2, 10, ou b. Ces progrès récents rendent donc parfaitement utilisable, en pratique, cette belle numération de Zeckendorf.

CODE DE GROS-GRAY

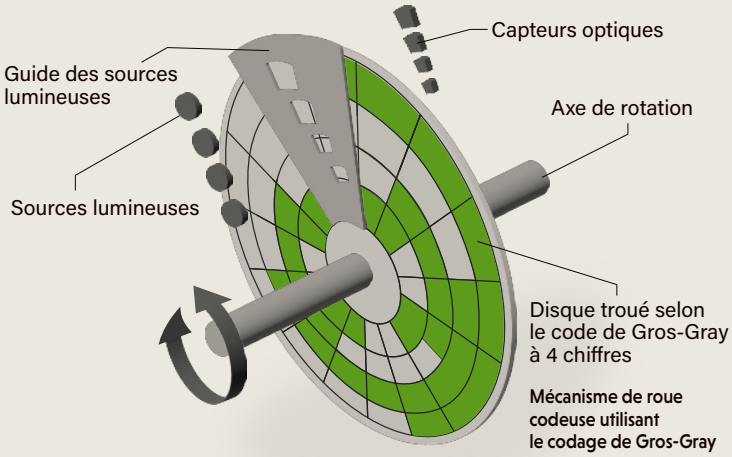
Une autre numération exotique s'est révélée utile en informatique et en robotique : il s'agit du système de numération appelé « code de Gray ». Par abus de langage, dans la suite, on parlera du « code » d'un entier pour désigner sa représentation dans ce système de numération.

Ce système fut breveté par l'ingénieur américain Franck Gray en 1947, qui ignorait que Luc Agathon Louis Gros, clerc de notaire puis conseiller à la Cour d'appel de Lyon, avait déjà

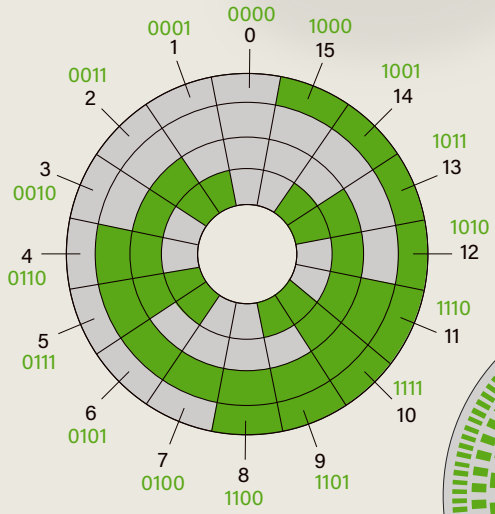
3

ROUES CODEUSES ET CODE DE GROS-GRAY

Le code de Gros-Gray utilise uniquement des 0 et des 1 pour représenter les entiers. Il a la propriété que, pour passer du code de n à celui de $n + 1$, un seul chiffre change. Cette propriété est utilisée dans les roues codeuses : des roues munies de dispositifs optiques permettant d'en mesurer la position angulaire, et donc d'en suivre et d'en piloter les mouvements de rotation. L'angle de rotation par rapport à une position de base est mesuré en lisant optiquement des traces dessinées sur la roue ou sur un disque solidaire de la roue. Parfois, il s'agit d'un disque dans lequel des trous ont été découpés. Ces découpages peuvent être basés sur le code de Gros-Gray : le fait qu'un seul chiffre change à chaque mouvement unitaire rend le dispositif plus fiable et plus robuste que ce que donnerait un dispositif du même type avec, par exemple, des représentations binaires usuelles.



Mécanisme de roue codeuse utilisant le codage de Gros-Gray



Disques optiques utilisant le codage de Gros-Gray, à quatre chiffres *(ci-dessus)* et neuf chiffres *(ci-contre)*. Pour le disque à quatre chiffres, on indique en noir la position de la roue, représentée en décimal, et en vert le code de Gros-Gray correspondant.

4 CALCULS EN TERNAIRE ÉQUILIBRÉ

En ternaire équilibré, on utilise uniquement des 1, des 0 et des – 1 (notés $\underline{1}$). La représentation $a_p a_{p-1}...a_1 a_{0\,TE}$ désigne le nombre entier positif ou négatif : $a_p \times 3^p + a_{p-1} \times 3^{p-1} + ... + a_1 \times 3^1 + a_0$. Les additions s'effectuent en suivant exactement la méthode classique utilisée avec notre système de numération décimal. Il faut simplement prendre garde à reporter soigneusement les retenues, qui peuvent être négatives. On utilise pour cela le fait que : $1_{TE} + 1_{TE} = 1\underline{1}_{TE}$; $\underline{1}_{TE} + \underline{1}_{TE} = \underline{1}1_{TE}$; etc. On peut ainsi poser, par exemple, l'opération suivante :

$$\begin{array}{r} \scriptstyle 1\ 1\ \ \underline{1}\ \ (\text{retenues}) \\ \underline{1}\,1\,1\,0\,\underline{1}_{TE} \\ +\,1\,1\,1\,1\,\underline{1}_{TE} \\ \hline 1\,0\,\underline{1}\,0\,1_{TE} \end{array}$$

Notons que le temps nécessaire pour l'addition est proportionnel au nombre de chiffres du plus grand des nombres additionnés.
Pour la multiplication, la méthode de notre système décimal s'adapte elle aussi parfaitement, en notant que multiplier un nombre par 1 ne change rien, multiplier par $\underline{1}$ revient à changer chaque 1 en $\underline{1}$ et chaque $\underline{1}$ en 1, et multiplier par 0 donne toujours 0.

Si, par exemple, on souhaite calculer le carré de 17, on part de $17 = 27 - 9 - 1 = 1\underline{1}01_{TE}$, et on pose :

$$\begin{array}{r} 1\,\underline{1}\,0\,\underline{1}_{TE} \\ \times\ 1\,\underline{1}\,0\,\underline{1}_{TE} \\ \hline \underline{1}\,1\,0\,\underline{1}_{TE} \\ +\quad 0\,0\,0\,0_{TE} \\ +\ \ 1\,\underline{1}\,0\,\underline{1}_{TE} \\ +\,1\,\underline{1}\,0\,\underline{1}_{TE} \\ \hline 0\,1\,1\,\underline{1}\,\underline{1}\,0\,1_{TE} \end{array}$$

On trouve bien que $11\underline{11}01_{TE} = 243 + 81 - 27 - 9 + 1 = 289 = 17 \times 17$.

publié en 1872 un petit livre le décrivant. Le nom « code de Gros-Gray » ou même « code de Gros » devrait donc être utilisé… ce qui n'est pourtant pas l'usage.

Ce système de numération, comme le binaire usuel, n'utilise que les chiffres 0 et 1. Son intérêt est que, lorsqu'on passe de la représentation de l'entier n à la représentation de l'entier $n+1$, il n'y a qu'un seul chiffre de la représentation qui change. Il est possible de reconstituer le code de Gros-Gray de n'importe quel entier avec les informations suivantes.

(a) Chaque code doit commencer par un 1 sauf celui de l'entier 0 qui est 0.

(b) Il ne faut ajouter un chiffre supplémentaire pour passer de n à $n+1$ que lorsque c'est absolument nécessaire, c'est-à-dire lorsque toutes les séquences possibles de 0 et de 1 de même longueur que la représentation de l'entier n et commençant par 1 ont été épuisées pour les nombres précédents.

(c) Le chiffre unique changé pour passer du code de n à celui de $n+1$ est celui le plus à droite parmi ceux qui sont possibles.

À l'aide de ces trois règles, on détermine les codes des premiers entiers, présentés ci-contre.

Une manière équivalente d'obtenir le code de l'entier $n+1$ connaissant celui de l'entier n consiste à changer le dernier chiffre du code de n si le nombre de 1 dans ce code est pair, et à changer le chiffre à gauche du 1 le plus à droite sinon.

Ces définitions ne sont utilisables pour déterminer le code d'un entier que si l'on connaît le code de l'entier précédent. Il semble donc que, pour avoir le code d'un entier n, il faille recalculer un à un tous les codes des entiers précédents, ce qui est très long si n est grand. Heureusement, il existe un autre moyen permettant de déterminer rapidement le code de Gros-Gray d'un entier n quelconque. On commence par calculer la représentation binaire de n. On sait que le premier chiffre du code de Gros-Gray recherché est un 1. Ensuite, si les chiffres de la représentation binaire en positions $i-1$ et i (en comptant à partir de la gauche) sont égaux, le chiffre en position i du code de Gros-Gray de n sera un 0, sinon ce sera un 1. Ainsi, l'entier $25 = 16+8+1$ s'écrit 11001_{base2}. Le code de Gros-Gray de 25, déduit grâce au procédé ci-dessus, est donc 10101_{GG}. En effet, le premier 1 (à gauche) de la représentation binaire est recopié dans le code de Gros-Gray. Le 0 en position 2 de 10101_{GG} provient de ce que le début de 11001_{base2} est 11, soit deux chiffres égaux. Le 1 en troisième position de 10101_{GG} provient de ce que les chiffres en position 2 et 3 de 11001_{base2} sont différents, etc.

Il existe aussi une méthode qui, à l'inverse, permet de passer du code de Gros-Gray d'un entier à sa représentation binaire. On commence par placer un 1 tout à gauche, puis le chiffre en position i (à partir de la gauche) de la représentation binaire est un 0 si et seulement si le nombre de 1 dans le code de Gros-Gray parmi les positions 1, 2, …, i est pair. Ainsi, partant du code de Gros-Gray 10101_{GG}, on en déduit que le nombre représenté est 11001_{base2}. Pour le déterminer, on place d'abord un 1 en tête de la représentation binaire recherchée. Le second 1 de 11001_{base2} provient de ce que les deux premiers chiffres de 10101_{GG} comportent un nombre impair de 1. Le 0 en troisième position de 11001_{base2} provient de ce que les trois premiers chiffres de 10101_{GG} comportent un nombre pair de 1, etc.

Ces deux méthodes de conversion utilisent un temps de calcul proportionnel à la longueur des représentations manipulées, car on les parcourt une fois en effectuant, à chaque pas de calcul, une même quantité finie d'opérations. Cela a pour conséquence que, pour additionner ou multiplier des nombres représentés avec le code de Gros-Gray, la méthode consistant à passer par la représentation binaire ou décimale est assez satisfaisante : elle ne nécessitera qu'un temps de calcul supplémentaire proportionnel à la longueur des représentations, ce qui est raisonnable – surtout pour la multiplication.

CODES DE GROS-GRAY DES PREMIERS ENTIERS

$0 = 0_{GG}$
$1 = 1_{GG}$
$2 = 11_{GG}$
$3 = 10_{GG}$
$4 = 110_{GG}$
$5 = 111_{GG}$
$6 = 101_{GG}$
$7 = 100_{GG}$
$8 = 1100_{GG}$
$9 = 1101_{GG}$
$10 = 1111_{GG}$
$11 = 1110_{GG}$
$12 = 1010_{GG}$
$13 = 1011_{GG}$
$14 = 1001_{GG}$
$15 = 1000_{GG}$
$16 = 11000_{GG}$
$17 = 11001_{GG}$

Le code de Gros-Gray est souvent utilisé en informatique, par exemple dans les systèmes de correction d'erreur. Il est un élément important des dispositifs électriques ou électroniques de positionnement et de commande de roues en robotique (*voir l'encadré 3*).

L'utilisation exclusive de 0 et de 1 dans les représentations binaires, de Zeckendorf et de Gros-Gray oblige, pour représenter les nombres négatifs, à utiliser un symbole supplémentaire, le signe «−», dont le rôle est peu homogène avec le reste du codage. Quitte à ajouter un troisième symbole, autant lui donner un rôle plus important: c'est ce que propose le système positionnel suivant, appelé «ternaire équilibré».

TERNAIRE ÉQUILIBRÉ

Ce dernier mérite une attention particulière, car il est lui aussi utile en informatique. Le célèbre informaticien américain Donald Knuth disait de lui qu'il était «le plus joli de tous les systèmes de numération». Le théorème qui justifie son existence affirme que tout nombre entier non nul, positif ou négatif, peut s'écrire de manière unique sous la forme: $a_p\times 3^p+a_{p-1}\times 3^{p-1}+\ldots+a_1\times 3^1+a_0$, avec chaque a_i égal à −1, 0, ou 1, et a_p non nul. La représentation ternaire équilibrée d'un tel entier est alors $a_pa_{p-1}\ldots a_1a_{0\,\mathrm{TE}}$.

En pratique, plutôt que d'utiliser le chiffre −1, qui comporte deux symboles, on préférera utiliser un symbole unique qui, pour nous, sera le «1 souligné»: $\underline{1}$. Ainsi, $208=243-27-9+1=3^5-3^3-3^2+1=10\underline{1}\underline{1}01_{\mathrm{TE}}$.

Pour obtenir la représentation en ternaire équilibré d'un entier positif n, on commence par en calculer la représentation en base 3 par les méthodes usuelles: rechercher la plus grande puissance de 3 inférieure ou égale à n, la soustraire à n, recommencer l'opération, etc. Notons qu'une même puissance de 3 peut être soustraite 0, 1 ou 2 fois d'affilée. Pour l'entier 208, on obtient: $208=2\times 81+1\times 27+2\times 9+1=2\times 3^4+1\times 3^3+2\times 3^2+1=21201_{\text{base }3}$. On additionne ensuite à la représentation obtenue un nombre de la forme $1111\ldots11_{\text{base }3}$, plus long d'un chiffre que notre représentation: $21201_{\text{base }3}+111111_{\text{base }3}=210012_{\text{base }3}$. On soustrait enfin une unité à chaque chiffre obtenu, ce qui fournit le résultat: $208=10\underline{1}\underline{1}01_{\mathrm{TE}}$. Pour obtenir la représentation en ternaire équilibré d'un entier négatif n, il suffit de calculer la représentation en ternaire équilibré de $-n$ par la méthode précédente, puis de changer chaque 1 en $\underline{1}$, et chaque $\underline{1}$ en 1.

Cette numération positionnelle ternaire a de nombreux avantages:

– Même les nombres négatifs ont une représentation, qui n'exige l'introduction d'aucun symbole supplémentaire.

– Changer le signe d'un nombre est simple: il suffit d'intervertir les 1 et les $\underline{1}$.

– Le signe d'un nombre est celui de son premier chiffre.

– La comparaison de deux nombres s'opère comme en base b, en comparant de gauche à droite les chiffres des représentations.

– Un nombre est un multiple de 3 si son dernier chiffre est 0.

– L'addition et la multiplication s'opèrent de manière très simple, comme pour les représentations usuelles en base b (*voir l'encadré 4*).

– La soustraction se ramène à l'addition, puisqu'il est immédiat d'avoir la représentation de l'opposé d'un nombre en intervertissant les 1 et les $\underline{1}$.

Ce système, peu connu aujourd'hui, a en réalité une histoire ancienne: on en trouve des traces dès le IXe siècle en Perse, où il est expliqué par le mathématicien Al-Tabari. Sa description comme système de numération est due à John Colson en 1726, puis à Sir John Leslie qui en proposa une reformulation en 1817. Sérieusement envisagé pour être utilisé à la place du système binaire par les concepteurs des premiers calculateurs américains au milieu du XXe siècle, le ternaire équilibré était aussi au cœur de calculateur soviétique Setun, fabriqué en 1958. Il a depuis été supplanté par le binaire, et progressivement oublié.

UN TOUT NOUVEAU SYSTÈME

Tout récemment, un nouveau système de représentation des entiers, que nous nommerons «binaire alternant», a été proposé. Il possède une incontestable beauté mathématique, mais son intérêt pratique ou informatique reste encore à évaluer. Son existence repose sur un théorème démontré en 2024 par le chercheur hongrois Péter Hajnal. Ce résultat assure qu'en additionnant et en soustrayant alternativement des puissances de 2 décroissantes, on peut obtenir tout nombre entier positif, et que cette décomposition est unique si l'on impose aux deux derniers exposants utilisés (s'il y en a deux ou plus) de différer d'au moins de 2 unités. Énoncé de manière plus formelle: tout entier non nul n, s'écrit de manière unique sous la forme $a_1\times 2^{p_1}+a_2\times 2^{p_2}+\ldots+a_{k-1}\times 2^{p_{k-1}}+a_k\times 2^{p_k}$, où les p_i sont des entiers tels que $p_1>p_2>\ldots>p_{k-1}>1+p_k$, où chaque a_i vaut alternativement 1 ou −1, et où a_1 est non nul.

Comme en ternaire équilibré, le −1 sera noté $\underline{1}$. On obtient donc, par exemple, que $31=32-1=2^5-2^0=100001_{\mathrm{BA}}$, ou encore que $-30=-32+2=-2^5+2^1=\underline{1}00010_{\mathrm{BA}}$. On vérifie de la même manière que $412=10\underline{1}0100\underline{1}00_{\mathrm{BA}}$, et que $-2023=\underline{1}00000\underline{1}0\underline{1}001_{\mathrm{BA}}$.

Il faut cependant noter qu'aucun algorithme efficace d'addition ou de multiplication n'a encore été proposé: il est aujourd'hui nécessaire de repasser par un autre système de numération pour pouvoir effectuer des calculs. ■

BIBLIOGRAPHIE

P. Hajnal, Binary numeration system with alternating signed digits and its graph theoretical relationship, *Algorithms*, 2024.

L. Rougetet, *Le Binaire au bout des doigts*, EDP Sciences, 2023.

Y. Wu *et al.*, Cloud-edge data encryption in the internet of vehicles using Zeckendorf representation, *Journal of Cloud Computing*, 2023.

T. Idziaszek, Efficient algorithm for multiplication of numbers in Zeckendorf representation, *FUN 2021*, 2020.

C. Ahlbach *et al.*, Efficient algorithms for Zeckendorf arithmetic, *Fibonacci Quarterly*, 2013.

D. Jones, Ternary number systems, *page web personnelle*, 2013.

D. Knuth, *The Art of Computer Programming (vol. 2)*, Addison-Wesley, 1997.

C. Frougny, Fibonacci representations and finite automata, *IEEE Transactions on Information Theory*, 1991.