Institutional Repository - Research Portal
Dépôt Institutionnel - Portail de la Recherche
researchportal.unamur.be
University of Namur

THESIS / THÈSE

MASTER EN SCIENCES INFORMATIQUES

Algorithmes de manipulation des grands nombres entiers : preuves de correction

Hastir, Michel

Award date:
1986

Awarding institution:
Universite de Namur

Link to publication

General rights
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

- Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
- You may not further distribute the material or use it for any profit-making activity or commercial gain
- You may freely distribute the URL identifying the publication in the public portal ?

Take down policy
If you believe that this document breaches copyright please contact us providing details, and we will remove access to the work immediately and investigate your claim.

Download date: 23. Jun. 2020

ALGORITHMES DE MANIPULATION DES GRANDS NOMBRES ENTIERS PREUVES DE CORRECTION.

Promoteur : B. Le Charlier

Mémoire présenté pour l'obtention du grade de Licencié et Maître en Informatique par

Année académique 1985-1986

Michel Hastir

Mes remerciements iront tout naturellement à mon épouse qui pendant ces deux années a su accepter les contraintes, voire parfois les sacrifices, que peut nécessiter le suivi d'études universitaires menées en parallèle à un emploi à horaire complet d'enseignant dans le secondaire.
Cette réussite n'aurait pas été possible si mes collègues de travail et mon directeur Monsieur Mengal ne m'avaient pas aidé à contourner certaines difficultés comme celles liées à des contraintes d'horaire.
Enfin, merci aussi à Monsieur Le Charlier, mon promoteur, qui au cours de ces deux années et notamment pendant la réalisation de ce mémoire a su me fournir aux moments adéquats l'encouragement nécessaire et les conseils bien utiles.

INTRODUCTION

Le sujet de ce mémoire ainsi que certains de ses principes sont issus d'un projet informatique traité dans le courant de l'année 1985. Le problème consistait à écrire un ensemble de programmes qui permettraient de manipuler des grands nombres entiers, entendons par là de leur appliquer les opérations (*) arithmétiques habituelles. Une série de programmes écrits en langage Basic avaient donc vu le jour. Le tout formait une "belle" application de la notion de "liste chaînée" utilisant elle-même d'ailleurs la notion bien connue de "pointeur", d'autant plus que nous avions dû simuler cette dernière non présente en Basic.

A la fin de ce travail, un certain nombre d'interrogations subsistaient et des souhaits déjà apparaissaient. Nous avions écrit les programmes en supposant que les nombres entiers traités étaient représentés dans le système habituel de base dix. C'était déjà là une limitation non négligeable. Cette limitation avait-elle des conséquences au niveau des temps d'exécution et/ou de la place mémoire nécessaire? Cela était bien possible, mais rien n'était moins sûr. Les programmes écrits étaient-ils corrects? Il semblait bien que oui puisqu'ils avaient résisté à toute une série de tests, mais qui aurait osé le garantir! L'idée alors naissait de traiter le problème bien plus théoriquement que cela n'avait été fait(en permettant entre autres (**) de travailler dans n'importe quelle base)mais aussi et surtout de fournir des algorithmes de manipulation des grands nombres entiers garantis corrects et non plus seulement à forte probabilité d'exactitude. C'est en ces deux idées que se résume très grossièrement l'objet de ce travail.

(*) Nous préciserons plus tard ce qu'il y a effectivement lieu d'entendre par grands nombres entiers.

(**) Le mot théoriquement est à prendre dans le sens de "de manière générale".

Reprendre quelque peu l'historique du déroulement de la réalisation de ce mémoire n'est pas dénué d'intérêt car celui-ci fait apparaître un certain nombre de points importants.
J'ai commencé par effectuer un petit sondage auprès de personnes susceptibles d'être intéressées par mes futurs résultats, manière de me convaincre moi-même que le problème n'était pas purement académique. Je dois avouer que je n'ai pas rencontré de ce côté tout l'enthousiasme que j'étais pourtant en droit d'espérer. Je n'avais tout de même pas perdu mon temps car j'étais, par le biais de ce sondage, parvenu à obtenir quelques références d'articles traitant du sujet. De fil en aiguille, je m'apercevais que la littérature relative à ce propos était loin d'être négligeable. Le sujet semblait en effet avoir inspiré bon nombre de personnes.
Les articles étaient loin d'être dépourvus d'intérêt. Nous en avons d'ailleurs retiré un certain nombre d'idées. Mais force est de reconnaître que nous sommes en droit de leur adresser un certain nombre de reproches. Ainsi trop souvent les problèmes auxquels les algorithmes prétendent apporter une solution sont peu ou mal spécifiés. Trop d'hypothèses non mises en évidence sont faites et trop de choses sont passées sous silence. Mais surtout trop d'algorithmes se voient dramatiquement érigés sur la base d'exemples forcément trop particuliers. Je ne nie pas ici l'intérêt ni même d'ailleurs la nécessité d'avoir toujours à l'esprit une série d'exemples lors de l'élaboration d'un algorithme mais ceci ne peut suffire à garantir que l'algorithme construit répond bien à ses spécifications. Enfin et nous arrêterons là notre critique négative, trop de "macro-instructions" sous le faux prétexte qu'elles sont simples à mettre en oeuvre constituent le squelette de beaucoup trop d'algorithmes. C'est avec l'intention de ne pas tomber dans le même mal, sans pour autant prétendre être parfait, que ce travail a été réalisé.
Pour poursuivre mon historique(et aussi le terminer)et par le fait même apaiser les personnes avides de "concret", je tiens à signaler que j'ai eu au début de cette année 1986 l'occasion de m'entretenir quelque peu avec Monsieur Quisquater occupé dans les laboratoires de la firme Philips pour y traiter des problèmes de sécurité informatique qui m'a mis en évidence que les problèmes de Cryptographie(une des techniques de solution

apportée à certains problèmes de sécurité informatique et relatifs à la transmission de données)faisaient appel à des algorithmes de traitement des grands nombres entiers.
Ainsi, sans vouloir nous étendre sur ce sujet et entrer dans des détails inutiles dans ce cadre, signalons que l'on rencontre dans la technique de sécurité dont question ci-dessus, l'obligation de multiplier entre eux des nombres premiers de l'ordre de 40.000 chiffres(expression faite en base dix), le caractère de sécurité reposant essentiellement sur l'impossiblité de retrouver en un temps matériellement raisonnable les facteurs du produit obtenu.
Effectuons à présent un survol des différents chapitres que nous allons rencontrer, présentant ainsi brièvement leur contenu.
Un premier chapitre est consacré à présenter un certain nombre de généralités concernant les nombres entiers . On commencera par y rappeler quelques caractéristiques fondamentales qui leur sont liées. On fera ensuite un rappel concernant les différents systèmes de représentation habituels(systèmes de position décimal, binaire, hexadécimal,octal)des nombres entiers. Nous consacrerons ensuite notre attention à présenter le mode de représentation des nombres entiers dans les machines de type IBM, mode de représentation partagé d'ailleurs par de nombreux constructeurs, y compris ceux de micro-ordinateurs.
Dans le second chapitre, nous présenterons d'abord une "définition" des grands nombres entiers, déduite de ce qui précède, plus exactement nous déciderons de ce qu'il y a lieu d'entendre par là. Nous présenterons alors quelques critères généraux permettant de comparer les degrés de qualité de différents modes de représentation des grands nombres entiers. Nous esquisserons ensuite un mode de représentation par contiguïté physique dont nous ne reparlerons plus ou du moins très peu après.
Le troisième chapitre qui constitue la partie principale de ce travail présente tout d'abord un mode général et théorique de représentation des grands nombres entiers, général dans le sens qu'il peut être "traduit", "interprété" en toute une série de modes de représentation différents dont ceux utilisant la contiguïté physique ou les listes.

C'est alors sur base de ce mode de représentation théorique que nous présenterons des algorithmes traitant les principales opérations arithmétiques dont bien sûr l'addition, la soustraction, la multiplication et la division. Mais nous ne nous contenterons pas de fournir les algorithmes, nous fournirons en plus pour chacun de ceux-ci une preuve de leur correction. C'est ici certes que l'on trouve l'originalité de ce mémoire, si originalité il y a. Il est effectivement tellement rare de trouver parallèlement à un algorithme de résolution de problème une preuve de sa correction, entendons par là une preuve que l'algorithme répond bien à ses spécifications, que nous n'hésitons pas à utiliser le mot "originalité". On se contente en effet bien souvent, bien trop souvent d'ailleurs d'une impression que l'exécution de l'algorithme fournira les résultats que l'on attend de lui.
Pour ce faire, nous utilisons essentiellement des démonstrations basées sur la technique de l'invariant.(Méthode de démonstration à posteriori, c'est-à-dire venant après construction de l'algorithme, basée sur la présence d'une relation sans cesse vérifiée (l'invariant)à un endroit bien précis de celui-ci)
En ce qui concerne l'algorithme de division, nous utiliserons plutôt une technique de démonstration à priori, entendons par là une technique permettant de construire un algorithme après ou du moins en même temps que la preuve de sa correction.(Utilisant ainsi l'aspect constructif de la méthode de l'invariant)
Dans ce chapitre, le schéma de travail sera souvent identique. On présentera les spécifications du problème, ensuite on fournira un algorithme de résolution agrémenté de temps à autre de l'un ou l'autre exemple suivi lui-même d'une preuve de sa correction mais aussi de remarques et de commentaires permettant de mieux comprendre l'algorithme ou d'en imaginer des variantes.
Notons déjà que les algorithmes qui sont fournis sont généralement courts mais que leur mise au point ou démonstration est longue. Mais bien que les démonstrations soient longues et parfois rebutantes, force est de reconnaître qu'elles ne font appel à rien d'autre qu'à un ensemble de concepts mathématiques fort élémentaires.

Un quatrième chapitre est consacré à donner une interprétation des algorithmes présentés auparavant dans le cadre où nous choisissons une représentation des nombres entiers(correspondant à la représentation théorique initiale)sous forme de listes, insistant d'ailleurs sur le caractère élémentaire des primitives à mettre en oeuvre à cet effet.
Enfin, dans le cinquième chapitre, on s'efforce de discuter le choix de la base B. Nous aboutissons alors à la mise en évidence d'un certain nombre de bases qu'il paraît intéressant de considérer pour d'éventuelles simulations futures.

TABLE DES MATIERES

Chapitre 1. GENERALITES CONCERNANT LES NOMBRES ENTIERS ET LEUR REPRESENTATION

1.1 Avertissement

Avant toute chose, il y a lieu de faire une distinction très nette entre un nombre entier et les nombreuses représentations de ce nombre entier. En fait, nous pouvons dire dans un certain sens qu'aucune personne n'a jamais vu un nombre entier. Nous en voyons seulement des représentations. Cela ne signifie pas pour autant que les nombres entiers soient mal connus. Bien au contraire. S'il existe une catégorie d'"êtres" bien connus, ce sont bien les nombres entiers. Ils le sont par leurs liens avec le monde ambiant et les significations qui leur sont attachées, autrement dit par leurs propriétés ou caractéristiques. Ainsi, par exemple, une des caractéristiques principales d'un nombre entier positif ou nul est de permettre le comptage des choses. Il n'est pas de mon intention de passer en revue les différentes caractéristiques bien connues d'ailleurs des nombres entiers,(caractéristiques qui, rappelons-le, les définissent) pas plus que de rappeler la différence entre un nombre entier positif, un nombre entier négatif et le nombre entier nul. Avant de passer au problème de la représentation des nombres entiers et dans le même ordre d'idée que pour ce qui précède, c'est-à-dire l'incitation à la prudence, rappelons quelques opérations courantes relatives aux nombres entiers. Il s'agit bien entendu de l'addition(le résultat de l'addition de deux nombres étant leur somme), de la soustraction(le résultat de la soustraction de deux nombres entiers étant la différence entre le premier et le second), de la multiplication(le résultat de la multiplication de deux nombres entiers étant leur produit)mais aussi de la division(le résultat étant composé de deux nombres entiers, à savoir Ie quotient et le reste de la division du premier nombre entier par le second)qui constituent les opérations de base(dans le sens de "les plus couramment utilisées"). N'oublions pas cependant d'autres opérations

telles que par exemple l'exponentiation ou le calcul du plus grand commun diviseur ou encore le calcul du plus petit commun multiple. Mon intention n'est non plus pas de passer en revue de manière exhaustive toutes les opérations relatives aux nombres entiers, pas plus que de les redéfinir, mais plutôt de mettre en évidence que ces opérations relatives aux nombres entiers sont tout aussi "mystiques" que les nombres entiers eux-mêmes, dans le sens où ces opérations ne sont définies que via leur signification et non pas(comme c'est trop souvent fait pourtant)via des représentations trop spécifiques des nombres entiers sur lesquels elles opèrent. Remarquons au passage, et ceci permettra de mieux comprendre ce qui précède, que les mathématiques modernes ont remis à leur juste valeur les définitions relatives aux opérations sur les nombres entiers. Ainsi, par exemple, la multiplication de deux nombres entiers positifs, définie elle-même par la définition du produit de ces deux nombres entiers positifs l'est par la signification de ce produit. Elle fait, dans une large mesure, abstraction des représentations de ces deux nombres entiers. Jugez vous-même. "Le produit de deux nombres entiers positifs est le cardinal du produit cartésien AXB de deux ensembles(A et B) dont les cardinaux sont respectivement ces deux nombres entiers positifs".(*) Cette définition est à mon sens une "bonne" définition par opposition à une définition que je qualifierai de "mauvaise" cette fois et qui consiste à expliciter l'algorithme de recherche du produit de deux nombres entiers positifs représentés dans le système de numération de base dix.

(*) Je dis dans une large mesure car on peut de toute façon considérer que la définition fait appel à une représentation de type ensembliste.

Dans la suite de ce travail, nous aurons donc toujours à l'esprit la conception des nombres entiers et des opérations sur ceux-ci telle que suggérée ci-dessus. Néanmoins, pour des raisons de facilité, il nous arrivera, mais seulement dans des cas où l'assimilation ne porte pas à conséquences, de confondre un nombre entier avec une de ses représentations ou la définition d'une opération avec un algorithme conçu à partir d'une représentation spécifique des nombres entiers sur lesquels elle porte. Nous allons maintenant passer en revue quelques systèmes habituels de représentation des nombres entiers en général en y attachant une importance toute particulière.

1.2 Systèmes de représentation habituels des nombres entiers
Les systèmes de position

En guise d'introduction aux systèmes de position et afin d'en expliquer le principe général, observons le cas particulier bien connu du système de position décimal.
On y dispose des signes ø,1,2,3,4,5,6,7,8 et 9 appelés chiffres de base et du signe - (et parfois aussi du signe +). Le principe de base de ce système est de former des groupements de dix éléments, dès que cela est possible. Les éléments à compter sont regroupés par paquets de dix unités, le dernier paquet étant éventuellement incomplet. Les paquets de dix unités sont regroupés eux aussi par paquets de dix,(on obtient alors des paquets de dix paquets de dix unités)et ainsi de suite. On déduit de cette façon la représentation bien connue des nombres entiers positifs. Tout nombre entier positif est représenté par (*)
l'unique suite non vide de chiffres de base, soit
$a_{n-1}a_{n-2}\ldots.a_o$ obtenue selon le principe de base suggéré ci-dessus,(a_{n-1} étant différent de ø)a_o représentant le nombre d'unités, a_1 représentant le nombre de paquets de dix unités, a_2 le nombre de paquets de dix paquets de dix unités, etc...
Ainsi, 947 est la représentation d'un nombre entier positif, 7 étant le nombre des unités, 4 le nombre des dizaines et 9 le nombre des centaines. (**)
La méthode pour représenter un nombre entier négatif est alors très simple. Il suffit d'adjoindre à la gauche de la représentation de son opposé le signe conventionnel -. Ainsi, -947 est la représentation d'un nombre entier négatif.
Le nombre entier nul est quant à lui représenté par le signe ø, pris parmi les chiffres de base.

(*) Par "positif" il y a lieu d'entendre "strictement postif". De même, par "négatif", il y aura lieu d'entendre "strictement négatif".

(**) Il existe une variante souvent utilisée, à savoir qu'on adjoint à la gauche de la suite $a_{n-1}a_{n-2}\ldots.a_o$ le signe conventionnel +. Ainsi, on utilise +947 au lieu de 947.

Le système de position binaire est en tout point semblable au système de position décimal. La seule différence(de taille il est vrai)réside dans le fait que les éléments à grouper le sont par paquets de deux, en lieu et place de dix. En conséquence, les signes ø et 1 suffisent comme chiffres de base. Ainsi, la suite 1øøø11ø11ø est la représentation d'un nombre entier positif, -1øøø11ø11ø est la représentation d'un nombre entier négatif. Le nombre entier nul est de nouveau représenté par le signe ø pris parmi les chiffres de base.
Le système de position hexadécimal est lui aussi en tout point semblable aux deux systèmes précédents. On dispose néanmoins cette fois de seize chiffres de base, à savoir ø,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F. Le groupement des éléments se fait cette fois par paquets de seize. Ainsi, Aø17BDø3 est la représentation hexadécimale d'un entier positif, -Aø17BDø3 est la représentation hexadécimale d'un entier négatif(l'opposé du précédent), alors que ø est toujours la représentation du nombre nul.
Mentionnons encore à titre d'exemple le système octal basé sur l'existence des huit chiffres de base ø,1,2,3,4,5,6 et 7 et par conséquent sur le principe de groupement des éléments par paquets de huit.
On voit dès à présent que les systèmes de position ne diffèrent l'un de l'autre que par le nombre fixé arbitrairement de leurs chiffres de base et que par conséquent le nombre de tels systèmes de position est infini.
Une constatation qui est loin d'être sans intérêt nous est suggérée par les observations suivantes.
Considérons le nombre entier positif dont la représentation dans le système de base huit(système octal)est 167. Considérons aussi le nombre entier positif dont la représentation dans le système de base deux(système binaire) est 111ø111. En fait, ces deux nombres entiers sont égaux puisque $(167)_8=(1.8^2+6.8+7)_{10}=(64+48+7)_{10}=(119)_{10}$ et que $(111ø111)_2=(1.2^6+1.2^5+1.2^4+0.2^3+1.2^2+1.2^1+1)_{10}=(64+32+16+4+2+1)_{10}=(119)_{10}$. (*)

(*) *Par $(167)_8=(119)_{10}$, il y a lieu de lire : "Le nombre entier positif dont la représentation dans le système octal est 167 est le même que celui dont la représentation dans le décimal est 119".*

On a donc que $(111\emptyset111)_2=(167)_8$. Observons que 111 n'est rien d'autre que la représentation dans le système binaire du nombre entier dont la représentation dans le système octal est 7, c'est-à-dire que $(111)_2=(7)_8$. De même $(11\emptyset)_2=(6)_8$ et $(1)_2=(1)_8$.
Observons aussi que $2^3=8$.
Considérons, à titre de seconde observation, le nombre entier positif dont la représentation dans le système de base seize(système hexadécimal)est A6AF. Considérons aussi le nombre entier positif dont la représentation dans le système de base deux(système binaire) est $1\emptyset1\emptyset\emptyset11\emptyset1\emptyset1\emptyset1111$. En fait, ces deux nombres entiers sont égaux puisque $(A6AF)_{16}=(10.16^3+6.16^2+10.16+15)_{10}=(40960+1536+160+15)_{10}=(42671)_{10}$ et que $(1\emptyset1\emptyset\emptyset11\emptyset1\emptyset1\emptyset1111)_2=(1.2^{15}+0.2^{14}+1.2^{13}+0.2^{12}+0.2^{11}+1.2^{10}+1.2^9+0.2^8+1.2^7+0.2^6+1.2^5+0.2^4+1.2^3+1.2^2+1.2+1)_{10}=(32768+8192+1024+512+128+32+8+4+2+1)_{10}=(42671)_{10}$.
On a donc que $(1\emptyset1\emptyset\emptyset11\emptyset1\emptyset1\emptyset1111)_2=(A6AF)_{16}$.
Observons que 1111 n'est rien d'autre que la représentation dans le système binaire du nombre entier dont la représentation dans le système de position hexadécimal est F, c'est-à-dire que $(1111)_2=(F)_{16}$. De même $(1\emptyset1\emptyset)_2=(A)_{16}$, $(\emptyset11\emptyset)_2=(6)_{16}$ et $(1\emptyset1\emptyset)_2=(A)_{16}$.
Observons aussi que $2^4=16$. (*)

Ces deux observations nous amènent à énoncer la propriété suivante que nous démontrerons ensuite.

PROPRIETE Soit $b_{m-1}b_{m-2}\cdots\cdots b_2b_1b_o$ la représentation d'un nombre entier positif dans le système de position de base 2^k, k désignant un nombre entier positif et $a_{n-1}a_{n-2}\cdots\cdots a_5a_4a_3a_2a_1a_o$ la représentation de ce même nombre entier positif dans le système binaire, c'est-à-dire que

$(b_{m-1}b_{m-2}\cdots\cdots b_2b_1b_o)_{2^k} = (a_{n-1}a_{n-2}\cdots\cdots a_4a_3a_2a_1a_o)_2$.

On a que $(b_o)_{2^k} = (a_{k-1}a_{k-2}\cdots\cdots a_o)_2$ et

$(b_1)_{2^k} = (a_{2k-1}a_{2k-2}\cdots a_k)_2$ et .....

$(b_{m-2})_{2^k} = (a_{(m-1)k-1}\cdots a_{(m-2)k})_2$ et

$(b_{m-1})_{2^k} = (a_{n-1}\cdots\cdots a_{(m-1)k})_2$

(**)

(*) *Il est à remarquer que* $(\phi 11\phi)_2$ *n'est pas au* <u>*sens strict*</u> *la représentation du nombre entier dont la représentation dans le système hexadécimal est 6 puiqu'il y a eu adjonction par la gauche d'un chiffre* ϕ.

(**) *Il est à remarquer que excepté pour* $(a_{n-1}\ldots\ldots a_{(m-1)k})_2$ *qui est la représentation au* <u>*sens strict*</u> *du nombre entier dont la représentation dans le système de base* 2^k *est* $(b_{m-1})_{2^k}$, *il peut ne pas en être ainsi pour les autres groupes. (Voir (*))*

Démonstration

Considérons le nombre entier dont la représentation dans le système binaire est $a_{n-1}a_{n-2}\cdots\cdots a_4a_3a_3a_1a_o$.

On a évidemment que $a_{n-1}a_{n-2}\cdots a_{3k}a_{3k-1}\cdots a_{2k}a_{2k-1}\cdots a_ka_{k-1}\cdots a_o$ représente le nombre entier défini par

$$a_{n-1}2^{n-1}+a_{n-2}2^{n-2}+\ldots+a_{3k}2^{3k}+\underbrace{a_{3k-1}2^{3k-1}+\ldots+a_{2k}2^{2k}}+\underbrace{a_{2k-1}2^{2k-1}+\ldots a_k2^k}$$
$$+\underbrace{a_{k-1}2^{k-1}+\ldots+a_o}=a_{n-1}2^{n-1}+a_{n-2}2^{n-2}+\ldots a_{3k}2^{3k}+\underbrace{(a_{3k-1}2^{k-1}+\ldots+a_{2k}2^o).}$$
$$\underbrace{(2^k)^2}+\underbrace{(a_{2k-1}2^{k-1}+\ldots+a_k2^o).(2^k)^1}+\underbrace{(a_{k-1}2^{k-1}+\ldots+a_o).(2^k)^o}. \quad (1)$$

Considérons le même nombre entier dont la représentation dans le système de base 2^k est $b_{m-1}b_{m-2}\cdots\cdots b_4b_3b_2b_1b_o$.

On a évidemment que $b_{m-1}b_{m-2}\cdots\cdots b_2b_1b_o$ représente le nombre entier défini par

$$b_{m-1}(2^k)^{m-1}+b_{m-2}(2^k)^{m-2}+\ldots+b_3(2^k)^3+\underbrace{b_2(2^k)^2}+\underbrace{b_1(2^k)^1}+\underbrace{b_o(2^k)^o}. \quad (2)$$

La décomposition suivant les puissances de 2^k étant unique, et puisqu'il s'agit nécessairement du même nombre entier, en comparant les expressions (1) et (2), on obtient automatiquement que

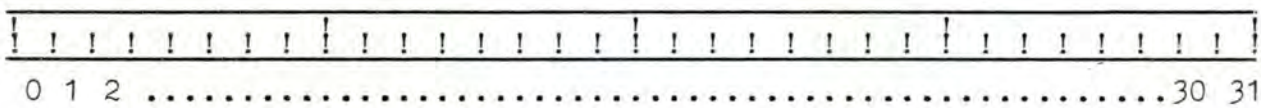
$(b_o)_{2^k} = (a_{k-1}a_{k-2}\cdots a_o)_2$ et $(b_1)_{2^k} = (a_{2k-1}a_{2k-2}\cdots a_k)_2$

et.....et $(b_{m-2})_{2^k} = (a_{(m-1)k-1}\cdots\cdots a_{(m-2)k})_2$ et $(b_{m-1})_{2^k} =$

$(a_{n-1}\cdots\cdots a_{(m-1)k})_2$, ce qui est bien le résultat annoncé.

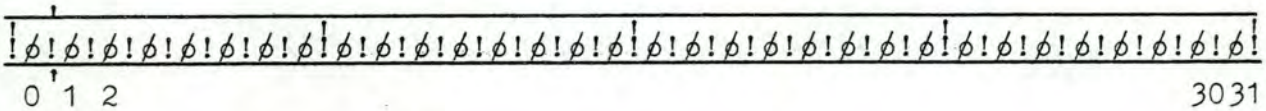
1.3 Mode de représentation des nombres entiers dans les machines de type IBM

Les modes de représentation des nombres entiers présentés jusqu'ici(systèmes de position)sont théoriques dans le sens qu'ils ne font pas appel aux supports "physiques" relatifs à l'ordinateur et dont principalement la mémoire centrale.
Comme tout informaticien sait, un ordinateur, fondamentalement, n'est en mesure que de manipuler des "ø" et des "1". Il y a donc lieu de trouver des modes de représentation adéquats(c'est-à-dire ne faisant intervenir que les notions de "ø" et "1")pour les nombres entiers si l'on veut les représenter et les manipuler en ordinateur. Comme nous allons le voir, ces modes ne s'éloignent pas nécessairement beaucoup des modes présentés précédemment et plus particulièrement du système de position binaire.
A titre d'exemple très représentatif(j'ai envie de dire à titre de modèle), nous allons présenter le mode de représentation des nombres entiers positifs ou nul et négatifs dans les machines de type IBM.(Ce mode de représentation, du moins son principe, est celui adapté par la plupart des constructeurs dont ceux de micro-ordinateurs)
La mémoire de l'ordinateur est divisée en bytes(groupe de 8 bits consécutifs)eux-mêmes regroupés en mots de 4 bytes consécutifs. (On parlera alors de mots entiers(FULLWORD : F/W)).
On peut aussi considérer des groupements de 2 bytes.(On parlera alors de demi-mots(HALFWORD : H/W)). Enfin, on considère aussi des groupements de 8 bytes.(On parle alors de mots doubles(DOUBLEWORD : D/W)).
Fixons notre attention sur le mot(F/W). De manière schématique, on peut considérer un mot de la manière suivante.

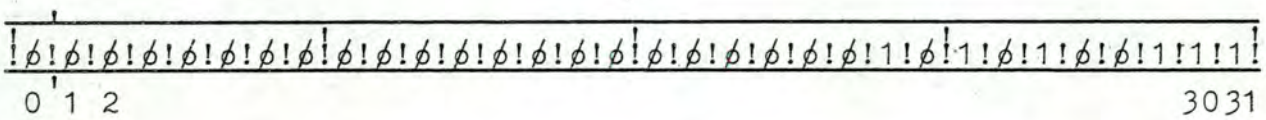


(4 bytes consécutifs de 8 bits consécutifs, chaque bit ne pouvant par définition prendre que la valeur ø ou 1)

-Le nombre entier nul est représenté par la succession des 32 bits mis à ø, soit

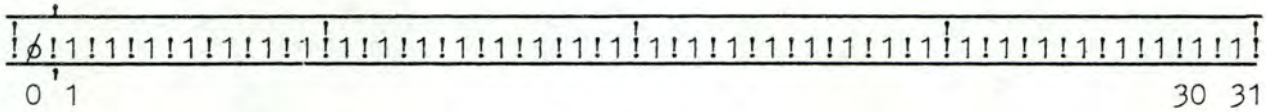


-Un nombre entier positif est représenté par le bit numéro 0 mis à ø suivi de la représentation dans le système de position binaire (voir le système de position binaire présenté auparavant)de ce nombre entier positif, représentation éventuellement précédée du nombre nécessaire de bits ø(bits non significatifs), de manière à "remplir" les 31 cases qui y sont consacrées. A titre d'illustration, voici la représentation du nombre entier positif dont la représentation dans le système de position décimal est 679



puisque $(1ø1ø1øø111)_2=(679)_{10}$.

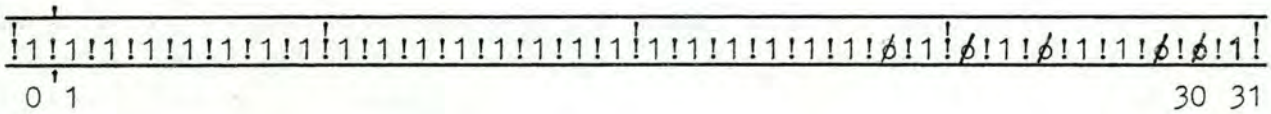
Il apparaît donc clairement que le plus grand nombre entier positif représentable de cette manière(c'est-à-dire dans un mot) correspond à la représentation suivante.



Le nombre entier positif suivant qui lui n'est plus représentable dans un mot a comme représentation dans le système de position binaire 1øøø.....ø.
31 fois

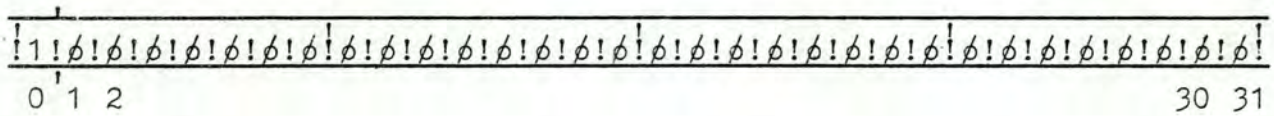
On en déduit immédiatement que le plus grand nombre entier positif représentable dans un mot(comme présenté ci-dessus)est le nombre entier positif correspondant à $2^{31}-1$.

-<u>Un nombre entier négatif</u> -N est représenté par la représentation dans le système de position binaire(voir le système de représentation binaire présenté auparavant)du nombre entier positif correspondant à 2^{32}-N.
A titre d'illustration, voici la représentation du nombre entier négatif dont la représentation dans le système de position décimal est -679



puisque $(2^{32}-679)_{10}=(4\ 294\ 966\ 617)_{10}$ et que
$(11111111\ 11111111\ 111111ø1\ ø1ø11øø1)_2 = (4\ 294\ 966\ 617)_{10}$.

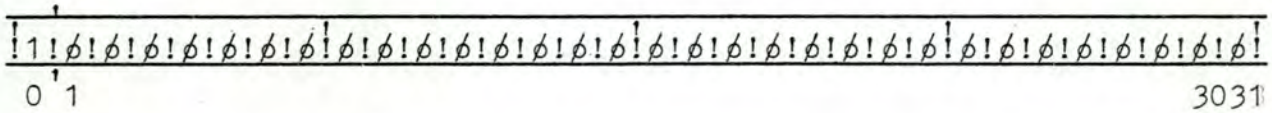
De manière à ne pas interférer avec la représentation des nombres entiers positifs(c'est-à-dire de manière à ce que le bit numéro ø soit 1), on déduit que le plus petit entier négatif représentable par ce système a comme représentation



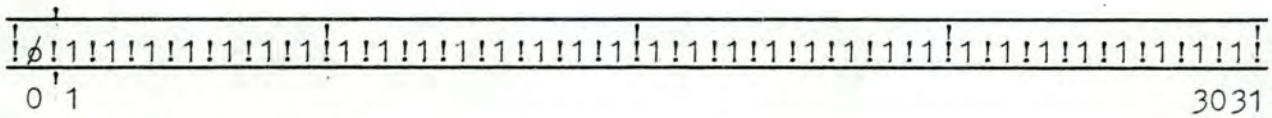
Or, cette représentation n'est rien d'autre que la représentation dans le système de position binaire du nombre entier correspondant à 2^{31}. Par conséquent, si l'on désigne par -N <u>le plus petit entier négatif représentable dans un mot</u>, on a que $2^{32}-N=2^{31}$ et donc que $-N=\underline{-2^{31}}$.

On peut montrer que pour obtenir "mécaniquement" la représentation du nombre entier négatif -N(qu'on supposera plus grand ou égal à -2^{31}), il suffit de prendre la représentation dans le système de position binaire sur 32 bits(c'est-à-dire qu'on adjoindra autant de ø non significatifs que nécessaire)du nombre entier positif N, d'ensuite transformer les ø en 1 et les 1 en ø et d'ajouter enfin 1 au sens de l'addition binaire. A titre d'exemple, observons la technique "mécanique" pour obtenir la représentation du nombre entier négatif correspondant à -2^{31}.

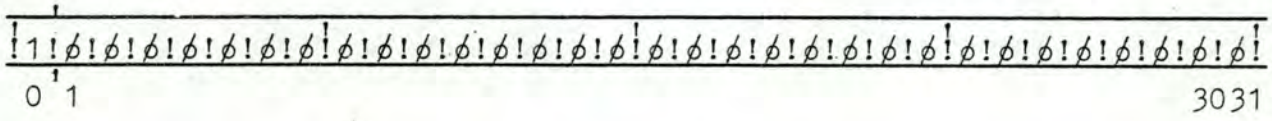
a)Représentation sur 32 bits du nombre entier correspondant à 2^{31}.
(dans le système de position binaire)



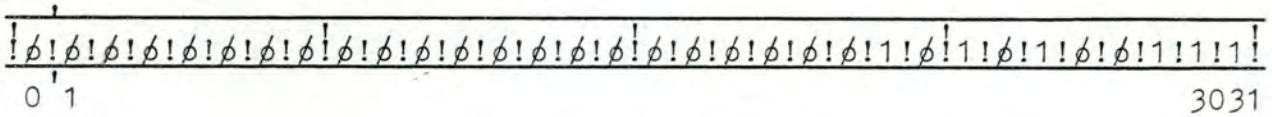
b)Transformation des ø en 1 et des 1 en ø

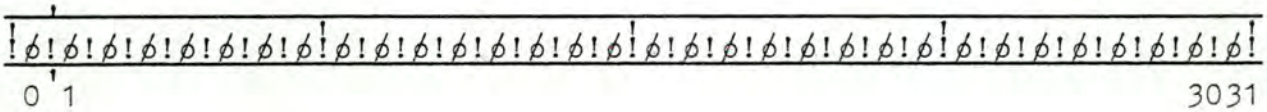


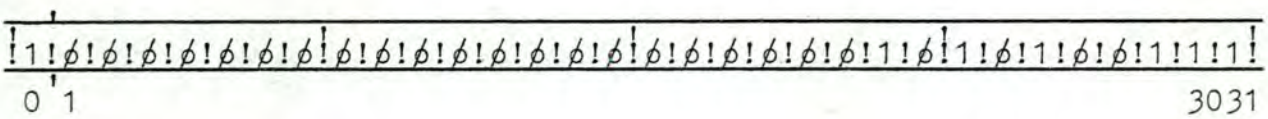
c)Addition binaire de 1



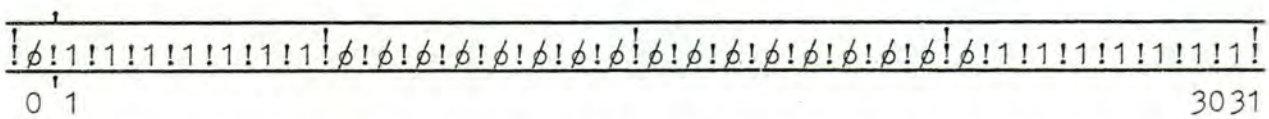
L'étape de passage de a) à b) étant appelée <u>complémentation</u>, nous appellerons cette technique de représentation des nombres entiers <u>technique du complément</u>.
Remarquons qu'il eût été plus simple <u>à priori</u> de choisir d'autres modes de représentation que la technique du complément. Ainsi, un mode de représentation qui sautait à l'esprit est le suivant.
Pour un <u>nombre entier positif ou nul</u>, même principe de représentation que pour la technique du complément.
Pour un <u>nombre entier négatif</u>, représentation de son opposé et changement du bit 0 de valeur ø en la valeur 1.
A titre d'illustration, voici les représentations correspondantes des nombres entiers dont les représentations dans le système de représentation décimal sont respectivement 679, 0 et -679.

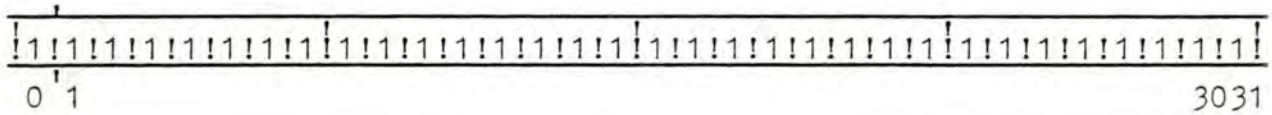




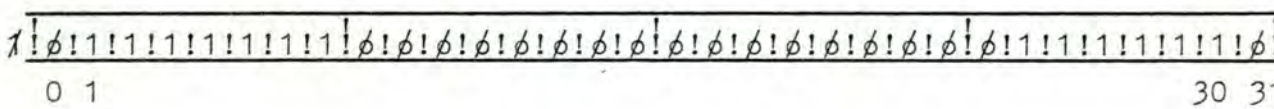


Remarquons que dans un tel mode de représentation, les représentations de deux nombres entiers opposés ne diffèrent l'une de l'autre que par le bit numéro 0, soit le bit de signe. C'est d'ailleurs ce qui fait l'apparente simplicité de ce système.
Le fait que ce mode de représentation (pourtant simple) ne soit pas retenu, au profit de la technique du complément (apparemment plus compliquée), réside dans le fait principal que ce mode de représentation se comporte relativement mal vis-à-vis des opérations arithmétiques, le contraire étant vrai pour la technique du complément. Par mauvais comportement vis-à-vis des opérations arithmétiques, il y a lieu d'entendre la difficulté à obtenir une représentation du résultat d'une opération à partir des représentations des éléments sur lesquels porte cette opération (les opérandes).
Sans vouloir entrer dans plus de détails, signalons au passage l'extrême facilité à obtenir, dans le système de représentation des nombres entiers basé sur la technique du complément, la représentation de la somme de deux entiers dont on connaît les représentations. Il suffit en effet d'effectuer une addition binaire classique, en négligeant le dernier bit de report éventuel comme illustré par l'exemple suivant. (Supposant que le résultat de l'addition est représentable dans un mot)
Soit les représentations suivantes de deux nombres entiers.





La première représentation est celle d'un nombre positif, celui dont la représentation dans le système de position décimal est 2 130 706 559.
La seconde représentation est celle d'un nombre entier négatif, celui dont la représentation dans le système de position décimal est -1.
Effectuons l'addition binaire classique entre ces deux représentations, on obtient,



en négligeant le dernier bit de report 1, ce qui n'est rien d'autre que la représentation du nombre entier positif dont la représentation dans le système de position décimal est 2 130 706 558, soit de la somme des deux précédents.
<u>La représentation de la somme s'obtient donc comme somme des représentations, le dernier bit de report étant supprimé</u>.
Cette technique fonctionne dans tous les cas de figures. La preuve nous en est fournie ci-dessous.

<u>Preuve</u> (Nous supposons l'absence d'underflow et d'overflow, c'est-à-dire le résultat représentable)

Soient N_1 et N_2 les deux nombres entiers à additionner.

.Si N_1 ou N_2 est le nombre entier nul, le résultat est évident.
.Supposons donc que ni N_1 ni N_2 n'est le nombre entier nul.
-Si N_1 et N_2 sont positifs, le résultat est encore évident.
-Si N_1 et N_2 sont négatifs, leur représentation par la technique du complément est en fait leur représentation dans le système de position binaire(sur 32 bits)des nombres entiers correspondant respectivement à $2^{32}+N_1$ et $2^{32}+N_2$ (*)

(*) Remarquons que nous associons ici, abusivement, 2^{32} avec l'entier qu'il représente(d'où l'expression "correspondant à")

En effectuant l'addition binaire de ces représentations, dernier report compris, on obtient donc la représentation dans le système de position binaire(sur 33 bits) du nombre entier correspondant à $2^{32}+N_1+2^{32}+N_2$, c'est-à-dire $2^{32}+2^{32}+N_1+N_2$.
En négligeant ce dernier bit de report(qui est nécessairement 1 puisque les bits numéro 0 des représentations de N_1 et N_2 sont 1 puisque N_1 et N_2 sont négatifs)on obtient la représentation dans le système de position binaire(sur 32 bits) du nombre entier correspondant à $2^{32}+(N_1+N_2)$. Puisque N_1+N_2 est négatif, celle-ci n'est rien d'autre que la représentation par la technique du complément du nombre entier N_1+N_2.

-Si N_1 est négatif et N_2 positif, la représentation de N_1 par la technique du complément est en fait la représentation dans le système de position binaire(sur 32 bits)du nombre entier correspondant à $2^{32}+N_1$ et la représentation de N_2 par la technique du complément est en fait la représentation dans le système de position binaire(sur 32 bits)de N_2. En effectuant l'addition binaire de ces représentations, dernier report compris, on obtient donc la représentation dans le système de position binaire(sur 33 bits) de l'entier correspondant à $2^{32}+(N_1+N_2)$. On peut facilement voir que le dernier bit de report est ø si N_1+N_2 est négatif et 1 si N_1+N_2 est positif ou nul. Dans le premier cas, en supprimant le dernier bit de report(ø), aucun changement n'intervient. On obtient ainsi la représentation dans le système de position binaire(sur 32 bits)du nombre entier correspondant à $2^{32}+(N_1+N_2)$. Mais puisque N_1+N_2 est négatif, cette représentation n'est rien d'autre que la représentation par la technique du complément du nombre entier N_1+N_2.
Dans le second cas, en supprimant le dernier bit de report(1), on obtient la représentation dans le système de position binaire (sur 32 bits)du nombre entier N_1+N_2 qui est la même que la représentation par la technique du complément de ce même nombre entier N_1+N_2 puisque ce dernier est positif ou nul.

Tout ce qui vient d'être exposé dans le cadre du mot(4 bytes consécutifs de 8 bits consécutifs) reste valable dans le cadre du demi-mot(2 bytes consécutifs de 8 bits consécutifs)ou du mot double à condition de tenir compte des quelques adaptations suivantes. Dans le cadre du demi-mot, un entier négatif -N est représenté par la représentation dans le système de position binaire(sur 16 bits) du nombre entier correspondant à 2^{16}-N(au lieu de 2^{32}-N dans le cadre du mot entier). De même, dans le cadre du mot double, un entier négatif -N est représenté par la représentation dans le système de position binaire(sur 64 bits)du nombre entier correspondant à 2^{64}-N.
En conséquence, dans le cadre du demi-mot, le plus grand entier positif représentable correspond à 2^{15}-1(au lieu de 2^{31}-1 dans le cadre du mot entier)et le plus petit nombre entier négatif représentable correspond à -2^{15}(au lieu de -2^{31} dans le cadre du mot entier).
Dans le cadre du double mot, on a les entiers correspondant respectivement à 2^{63}-1 et -2^{63} comme limites.

Chapitre 2. GENERALITES CONCERNANT LES GRANDS NOMBRES ENTIERS ET LEUR REPRESENTATION

2.1 Essai de définition des grands nombres entiers

Le problème qui nous intéresse précisément est celui de la représentation des "grands" nombres entiers(lié aux opérations arithmétiques portant sur ces nombres entiers).
Remarquons qu'il est bien difficile de définir de manière tout à fait absolue ce qu'il y a lieu d'entendre par "grand" nombre entier. Néanmoins, afin de fixer les idées, dès à présent, quand nous parlerons de "grands" nombres entiers, nous signifierons par là tout nombre entier dont la représentation dans le système choisi est impossible.(*) Ainsi, si nous choisissons le système de représentation par la technique du complément dans le cadre du mot, par grand nombre entier, nous signifierons par là tout nombre entier plus grand que le nombre entier correspondant à $2^{31}-1$ ou tout nombre entier plus petit que le nombre entier correspondant à -2^{31}. Par contre, si nous choisissons le système de représentation par la technique du complément dans le cadre du demi-mot, par grand nombre entier, il y aura lieu de comprendre tout nombre entier plus grand que le nombre entier correspondant à $2^{15}-1$ ou tout nombre entier plus petit que le nombre entier correspondant à -2^{15}.

Remarque

Une définition sensée aussi, mais plus restrictive, de "grand" nombre entier, consisterait à dire "tout nombre entier dont la représentation dans un système de position nécessite beaucoup de chiffres".(Malheureusement, cette définition est plus imprécise car il faudrait se mettre d'accord sur la signification du mot beaucoup)

(*) Remarquons que cette définition de "grand" nombre entier porte sur la valeur absolue des nombres entiers.

2.2 Critères de comparaison des modes de représentation des grands nombres entiers

On peut, bien évidemment, imaginer différents modes de représentation des grands nombres entiers. Parmi tous ces modes, certains sont bien meilleurs que d'autres, certains aussi sont franchement mauvais.
Plusieurs critères permettent de comparer les degrés de qualité de différents modes de représentation.
Un premier critère, qui sans contesteest capital, est celui de ce que j'appellerai l'adéquation aux opérations arithmétiques. Ainsi, un mode de représentation des nombres entiers qui ne permettrait pas ou qui permettrait très difficilement d'obtenir une représentation de la somme de deux nombres entiers dont on connaît les représentations respectives serait à proscrire.
Par contre, tout mode de représentation qui se comporte bien vis-à-vis des opérations arithmétiques, dans le sens où il est possible d'obtenir, de manière raisonnable, la(les) représentation(s) du(des) résultat(s) de l'opération portant sur des nombres entiers dont on connaît les représentations, mérite d'être pris en considération. Dans une large mesure, le critère d'adéquation aux opérations arithmétiques est lié au niveau de complexité des algorithmes réalisant ces opérations. Le seul moyen d'étudier valablement un mode de représentation suivant ce critère est donc de mesurer le degré de complexité des algorithmes liés aux opérations.
Un second critère, lui aussi d'importance, est celui de la taille mémoire occupée par une représentation d'un nombre entier et aussi de la taille mémoire nécessaire pour que les algorithmes réalisant les opérations arithmétiques soient exécutables.
Enfin, citons un troisième critère, à savoir celui de la vitesse d'exécution. A complexités d'algorithmes "égales" et tailles mémoire "égales", on préférera le(s) mode(s) de représentation offrant les plus faibles temps d'exécution. (*)

(*) Il est vrai qu'à ce stade, les notions de complexité d'algorithme, de taille mémoire et de temps d'exécution sont imprécises. La difficulté réside d'ailleurs dans la manière de mesurer de tels paramètres.

2.3 Un exemple de mode de représentation des grands nombres entiers basé sur le principe de la contiguïté physique

Présentons à présent un exemple précis de mode de représentation des grands nombres entiers, basé sur le principe de la contiguïté physique. Cet exemple n'a pas d'autre prétention que celle de montrer au lecteur ce que pourrait être un mode de représentation des grands nombres entiers.

Supposons disposer d'une machine dont la mémoire centrale est divisée en bytes(8 bits consécutifs). Tout nombre entier est représenté dans une zone de bytes contigus.(Le nombre de bytes qu'il est nécessaire d'utiliser étant fonction de la grandeur du nombre à représenter)
Le premier byte, celui de numéro 0, est destiné à recevoir le signe, plus exactement une représentation sur un byte du signe du nombre entier à représenter. Pour un nombre entier négatif, le contenu de ce premier byte serait par exemple

ø	ø	ø	ø	ø	ø	ø	1

Pour le nombre nul, le contenu de ce premier byte serait par exemple

ø	ø	ø	ø	ø	ø	ø	ø

Enfin, pour un nombre entier positif, le contenu de ce premier byte serait par exemple

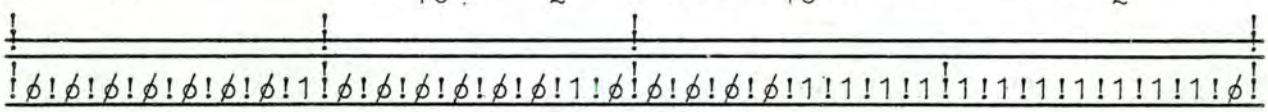
ø	ø	ø	ø	ø	ø	1	ø

Un ou plusieurs bytes(mais en tout cas un nombre fixé) suivant(s) est(sont) destiné(s) à recevoir la représentation dans le système de position binaire de la longueur en bytes de la représentation (qui suit)dans le système de position binaire de la valeur absolue du nombre entier considéré.

Pour fixer les idées, considérons les représentations des nombres entiers dont les représentations dans le système de position décimal sont respectivement -4094, 0 et 4094.
(On suppose qu'un byte seulement est consacré à la longueur dont question ci-dessus)
On a :

Pour -4094

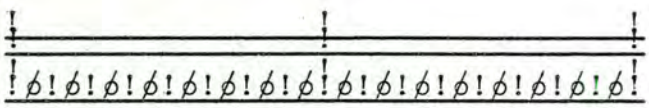
Il s'agit d'un négatif $(2)_{10}=(1\phi)_2$ $(4094)_{10}=(111111111110)_2$



byte 0 : byte de signe
byte 1 : byte de longueur
byte 2 : représentation de la valeur absolue du nombre entier considéré (système de position binaire)

Pour 0

Il s'agit du nombre nul $(0)_{10}=(\phi)_2$

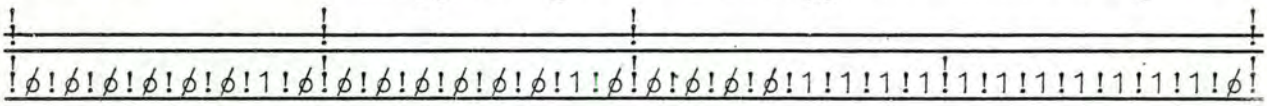


Remarquons que dans ce cas, la représentation est une suite vide de bytes

byte 0 : byte de signe
byte 1 : byte de longueur

Pour 4094

Il s'agit d'un positif $(2)_{10}=(1\phi)_2$ $(4094)_{10}=(111111111110)_2$



byte 0 : byte de signe
byte 1 : byte de longueur
byte 2 : représentation de la valeur absolue du nombre entier à considérer (système de position binaire)

Relativement à un tel mode de représentation, nous pouvons fournir les commentaires suivants.
On peut croire que ce mode de représentation soit particulièrement très bien adapté pour minimiser les temps d'exécution, sans aller toutefois à l'encontre de la minimisation de la place mémoire nécessaire lors de ces exécutions. En ce qui concerne ce dernier point, il semble même que, en tout cas en ce qui concerne la place mémoire nécessaire pour la représentation des nombres à traiter et des résultats obtenus(nous ne parlons pas ici de la place mémoire "intermédiaire" nécessaire en cours d'exécution pour obtenir les résultats à partir des données), ce mode de représentation soit même le meilleur que l'on puisse trouver. (*)
En effet,
-le fait de travailler à partir du système de position binaire (base 2) ne modifie en rien(en tout cas pas de manière sensible) la taille mémoire obligatoire pour la représentation d'un nombre entier. La preuve nous en est d'ailleurs fournie par la propriété énoncée page 6 et qui met en évidence que le fait de travailler dans une base bien plus grande que 2(par exemple 2^{16}=65536)ne modifie en rien la taille de la place mémoire nécessaire pour la représentation d'un nombre entier. En effet, le fait de travailler dans une grande base permet de diminuer sensiblement le nombre de coefficients(relatifs aux différentes puissances de cette base)nécessaires. Mais la place mémoire(en nombre de bits)alors nécessaire pour représenter de tels coefficients croît de manière inversement proportionnelle à la diminution du nombre de coefficients, de telle sorte même que l'on peut dire que le produit du nombre de coefficients nécessaires pour représenter un nombre entier dans une base donnée par le nombre de bits nécessaires pour représenter ces coefficients est une constante, indépendante donc de la base dans laquelle on travaille. (**)

(*) Par "meilleur", il y a lieu d'entendre "meilleur à quelques adaptations non significatives près".

(**) Pour s'en convaincre, on se référera à la page 6 dont déjà question ci-dessus.

-de plus, ce mode de représentation ne nécessite(à l'exception du(des) byte(s) nécessaire(s) à la représentation de la longueur dont question pages 19 et 20, et d'ailleurs négligeable(s))aucune place mémoire supplémentaire, ceci étant obtenu par le principe de la contiguïté physique.(On évite en effet notamment la place mémoire qui serait nécessaire dans un système de représentation par listes et dont nous reparlerons plus loin)
Cependant, et nous en reparlerons plus loin, la représentation par contiguïté physique n'offre pas la "maniabilité" qu'offre la représentation par listes.

Chapitre 3. MODE GENERAL ET THEORIQUE DE REPRESENTATION DES GRANDS NOMBRES ENTIERS ALGORITHMES DE MANIPULATION CORRESPONDANTS AVEC PREUVES DE CORRECTION

3.1 Mode général et théorique de représentation des grands nombres entiers

Considérons un nombre entier, noté a. Nous allons représenter tout nombre entier dans un système de position de base B, de la manière qui suit.
Soit B, un nombre entier supérieur ou égal à 2 appelé base de ce système de position. Soit aussi $\{-(B-1),\ldots,-1,0,1,\ldots.(B-1)\}$ l'ensemble des nombres entiers(négatifs, positifs ou nul)dont la valeur absolue est inférieure à B, appelés B-entiers.
Nous allons représenter un nombre entier positif a par l'unique suite $(a_0,a_1,a_2,\ldots.,a_{n-2},a_{n-1})$ de B-entiers non négatifs telle que $a_{n-1}>0$ et $a=\sum_{i=0}^{n-1} a_i B^i$.

Un nombre entier négatif a sera représenté par l'unique suite $(a_0,a_1,a_2,\ldots.,a_{n-2},a_{n-1})$ de B-entiers non positifs telle que $a_{n-1}<0$ et $a=\sum_{i=0}^{n-1} a_i B^i$.

Le nombre entier nul quant à lui sera représenté par la suite vide, notée ( ).
Pour être précis, disons plutôt que le nombre entier a non nul sera représenté par la suite des représentations(dans un système de représentation adéquat)des B-entiers $a_0,a_1,a_2,\ldots.,a_{n-2},a_{n-1}$ dont question ci-dessus. De plus, le nombre entier non nul sera représenté par une représentation jugée adéquate d'une suite vide.
Ici apparaissent tout naturellement et volontairement un certain nombre de libertés. Liberté dans le sens que nous acceptons, à priori, tout mode de représentation des B-entiers $a_0,a_1,a_2,\ldots.,a_{n-2},a_{n-1}$. Signalons au passage, à titre purement indicatif, qu'un

mode "sensé" de représentation des B-entiers est celui basé sur la technique du complément(Voir à ce propos la page 12).
Liberté aussi dans le sens que nous ne présupposons aucune forme précise de représentation de la notion de "suite". Signalons au passage et à titre purement indicatif qu'un mode "sensé" de représentation de la notion de "suite" est celui faisant intervenir la notion bien connue de liste qui elle fait intervenir la notion tout aussi connue de pointeur.
En vue de mieux comprendre ce qui précède, prenons la peine de regarder l'exemple suivant.

Exemple Soit B=10, base du système de position considéré.
Les B-entiers sont alors -9,-8,-7,-6,-5,-4,-3,-2,-1,0, 1,2,3,4,5,6,7,8,9.

Soit $a > 0$

a :--: $(a_0,a_1,....,a_{n-1})$ où $a_0,a_1,....,a_{n-1}$ sont des (*)
B-entiers positifs ou nuls et tels que $a_{n-1} > 0$ où

$$a=\sum_{i=0}^{n-1} a_i B^i.$$

exemple 1243007:--:(7,0,0,3,4,2,1) $\smile > 0$
(représentation classique)

Soit $a < 0$

a :--: $(a_0,a_1,....,a_{n-1})$ où $a_0,a_1,....,a_{n-1}$ sont des
B-entiers négatifs ou nuls et tels que $a_{n-1} < 0$ où

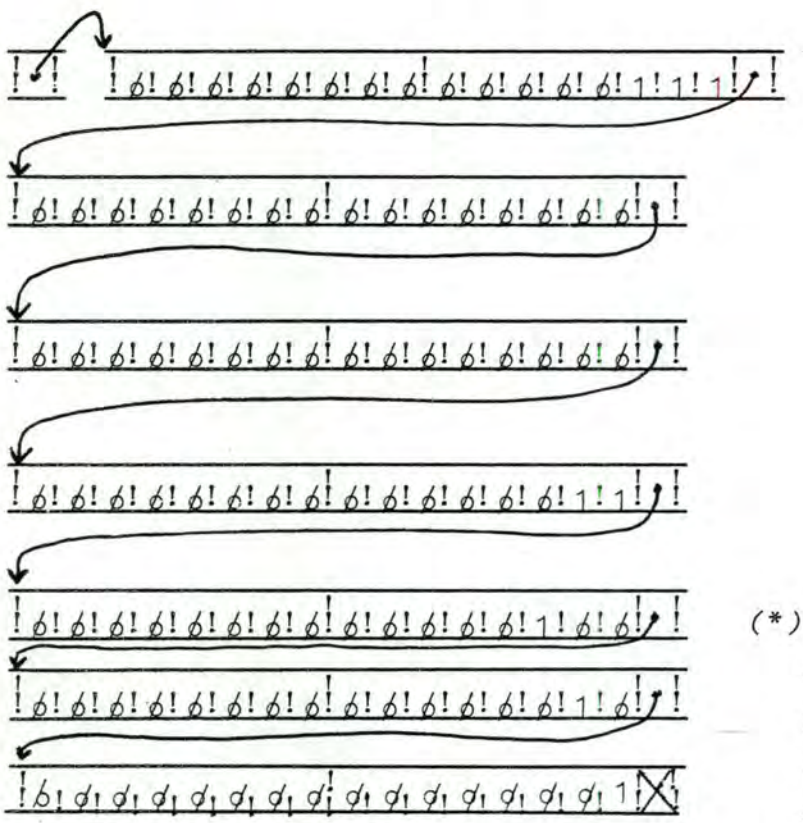
$$a=\sum_{i=0}^{n-1} a_i B^i.$$

exemple -1243007:--:(-7,0,0,-3,-4,-2,-1)
(représentation classique)

(*) :--: *signifie "est représenté par".*

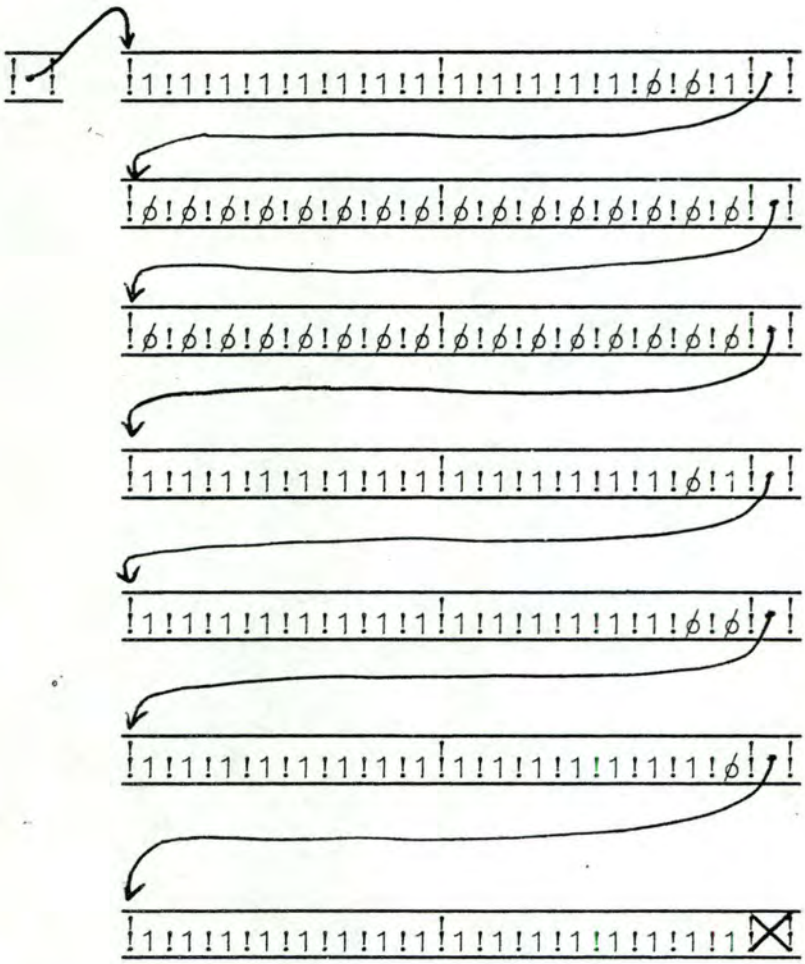
Soit a=0
a:--:( ) (suite vide)

Comme évoqué page 23, la suite (7,0,0,3,4,2,1) est encore une représentation "théorique"(ou "logique")du nombre entier dont la représentation dans le système décimal est 1243007 puisque cette suite peut être représentée de diverses manières d'un point de vue "physique", la liberté se situant d'une part au niveau du choix d'un mode de représentation des B-entiers qui interviennent dans la suite et d'autre part au niveau du mode de représentation de la notion de suite.
Remarquons que le choix d'une représentation physique particulière au détriment d'une autre est inévitablement lié au type de machine sur laquelle on travaille ainsi qu'au type de langage dans lequel sont écrits les algorithmes qui traduisent les manipulations sur les nombres entiers. (*)
Ainsi, sur une machine de type IBM, utilisant la représentation des B-entiers par la technique du complément dans des demi-mots (H/W) et supposant travailler dans un langage manipulant la notion de pointeur, une représentation plus "physique" du nombre entier dont la représentation dans le système de position décimal est 1243007 serait par exemple



(*)Ce qui ne veut nullement dire que pour un langage particulier sur une machine particulière, un certain choix ne soit encore possible.

tandis qu'au nombre entier dont la représentation dans le système de position décimal est -1243007 correspondrait la représentation



(*)

Enfin, au nombre entier nul correspondrait la représentation

(*) Faisons remarquer que, bien que ceci soit sans grande importance pour l'instant, le mode de représentation dont question ci-dessus s'avère insuffisant, un pointeur arrière pour chaque B-entier composant le nombre entier étant nécessaire, en plus du pointeur avant.

Remarquons que la longueur des entiers n'étant pas connue à l'avance, la représentation sous forme de listes présente le grand avantage d'éviter <u>certains</u> gaspillages "mémoire" dûs à l'allocation statique.
Signalons qu'une alternative à la représentation du nombre entier a par $(a_0, a_1, \ldots, a_{n-1})$ consiste en la représentation de ce même

B-entier le moins significatif — B-entier le plus significatif (1)

nombre entier par $(a_{n-1}, \ldots, a_1, a_0)$ (2)

B-entier le plus significatif — B-entier le moins significatif

Sans pour autant vouloir entrer dans des détails presque sûrement superflus, comparons brièvement au point de vue des avantages et désavantages la façon (1) avec la façon (2) dont question ci-dessus. Au niveau de l'addition et de la multiplication mais aussi de la soustraction, conceptuellement parlant, la façon (1) semble plus logique puisqu'il y a propagation dans le sens des listes. Par contre, au niveau de la division, les chiffres les plus significatifs étant d'abord requis, c'est la façon (2) qui semble la plus logique.(Voir à cet effet les algorithmes d'addition, de multiplication, de soustraction et de division présentés ultérieurement. De plus,la détermination du signe d'un nombre n'est pas toujours immédiate si l'on adopte la façon (1), contrairement au cas où l'on adopte la façon (2), ceci se produisant lorsque un ou plusieurs B-entiers les moins significatifs sont nuls. Comme nous le montrerons ultérieurement(voir l'algorithme de détermination du signe d'un nombre entier), cette différence entre les deux façons n'est pas significative. Nous voyons donc qu'il n'est pas possible de mettre en évidence de manière sensible la supériorité d'une des deux façons par rapport à l'autre. Même d'ailleurs, peu importe l'ordre d'écriture, ce qui compte plus c'est l'ensemble des <u>primitives</u> applicables aux suites.
Pour la suite, nous choisirons donc, de manière arbitraire, la façon (1).

Définition

Soit a, un nombre entier non nul représenté par la suite $(a_0, a_1, \ldots, a_{n-1})$. Nous appellerons n la B-longueur de a que nous noterons $L_B(a)$.

Soit a, nombre nul représenté par ( ). Nous poserons tout naturellement $L_B(a)=0$.

Dans ce qui suit et tout particulièrement au niveau de la présentation des algorithmes correspondant à différentes opérations sur les nombres entiers, nous resterons au niveau des représentations supérieures que nous avons qualifiées de logiques. Il ne sera donc pas question des représentations plus spécifiques que nous avons qualifiées de physiques. Ceci est fait intentionnellement, dans le souci essentiel de présenter des algorithmes généraux, aussi indépendants que possible de toute architecture physique. Néanmoins, de temps à autre, nous descendrons à un niveau plus physique, essentiellement lorsqu'il s'agira de mettre en évidence les avantages ou désavantages de telle représentation physique spécifique par rapport à telle autre. Dans un certain sens donc, nous pouvons dire que les algorithmes qui seront présentés sont théoriques, mais ils ont le grand avantage d'être adaptables à presque tout type de matériel et de langage.

3.2 Les algorithmes relatifs aux opérations arithmétiques Preuves de correction accompagnantes

Nous allons maintenant mettre en évidence des algorithmes relatifs aux opérations de base portant sur les nombres entiers. Nous présenterons ces algorithmes de manière progressive, montrant comment les premiers sont indispensables ou utiles à la conception des suivants.(Voir à ce propos le schéma récapitulatif qui sera dressé)
Chaque fois que celà sera possible, nous donnerons des renseignements relatifs aux temps d'exécution des différents algorithmes. Nous parlerons de temps maximum(noté t^+), de temps minimum(noté t^-) mais aussi de temps moyen(noté t^*) ou tout simplement de temps(noté t).

Dans beaucoup d'algorithmes, nous avons besoin de la détermination du signe d'un nombre entier. Considérons dès lors l'algorithme ISIGN(ISIGN pour "integer sign").

s ← ISIGN(a) ou si l'on veut ISIGN(a,s)

-Spécifications

a est un nombre entier. s est un B-entier.
A la fin de l'exécution de ISIGN, la variable s comprendra la valeur

-1 si a est négatif
ϕ si a est nul
1 si a est positif

-Algorithme

(1) s ← ϕ
(2) successivement vérifier $a_0, a_1, a_2, \ldots$ jusqu'à ce que le premier chiffre non nul a_k soit trouvé. Dans l'affirmative, si $a_k < 0$ alors s ← -1
sinon s ← 1

-Preuve de la correction de ISIGN(a,s)

La preuve de la correction de ISIGN(a,s) est presque évidente. Nous nous contenterons de signaler que si a est nul, l'étape (2) se réduit à néant puisque de par notre structure de représentation, la suite représentant le nombre entier nul est la suite vide(( )). Dans pareil cas, à la fin de l'exécution, la variable s comprendra bien la valeur ϕ(de par l'étape (1)).

-Remarques, commentaires

Si l'on avait choisi de représenter un nombre entier a par $(a_{n-1},....,a_1,a_o)$ plutôt que par $(a_o,a_1,....,a_{n-1})$, la détermination du signe d'un nombre entier eût été plus directe puisque pour un nombre entier a non nul, le signe de a n'est rien d'autre que le signe de a_{n-1}. Nous allons cependant montrer par une évéluation du temps moyen d'exécution de ISIGN(a,s) que contrairement à l'intuition, la représentation de a par $(a_o,a_1,....,a_{n-1})$ est comparable à celle de a par $(a_{n-1},....,a_1,a_o)$.

Notons par t^+_{ISIGN} le temps maximum d'exécution de ISIGN(a,s).
Notons par t^-_{ISIGN} le temps minimum d'exécution de ISIGN(a,s).
Enfin, notons par t^*_{ISIGN} le temps moyen d'exécution de ISIGN(a,s).
(Il y a lieu à ce stadede signaler que quand on parle de temps d'exécution(que ce soit maximum, minimum ou moyen), on entend par là une mesure de ce temps liéeau temps réel d'exécution. Il est en effet impossible de parler du temps réel d'exécution puisque ceci nécessiterait la connaissance des temps réels d'exécution de toutes les instructions de base disponibles dans le langage utilisé sur une machine particulière, ce dont nous ne disposons évidemment pas puisque nous parlons d'algorithmes théoriques)
En faisant abstraction du nombre entier nul et en négligeant l'étape (1) de l'algorithme, on a que

$$t^+_{ISIGN(a,s)} \hat{} L_B(a) \qquad (*)$$

Ceci arrive lorsque seul le chiffre(B-entier)le plus significatif de la représentation $(a_o,a_1,....,a_{n-1})$ est non nul, c'est-à-dire si seulement $a_{n-1} \neq \phi$

(*) Nous utiliserons la notation $\hat{}$ pour "est dominé par", "est conditionné par".

*$t^{-}_{ISIGN(a,s)} \hat{=} 1$

Ceci arrive lorsque le chiffre(B-entier)le moins significatif de la représentation $(a_o,a_1,\ldots,a_{n-1})$ de a est non nul, c'est-à-dire lorsque $a_o \neq \phi$.

*$t^{*}_{ISIGN(a,s)} \hat{=} 1$ et plus précisément $t^{*}_{ISIGN(a,s)} < (1-B^{-1})^{-2} = \frac{B^2}{(1-B)^2}$ (1)

Cette dernière formule, surprenante dans un certain sens est loin d'être aussi évidente que les deux précédentes. Nous allons donc la démontrer de manière mathématique. Avant tout essai de démonstration, explicitons le sens précis de la formule (1) ci-dessus.

Soit $n \geqslant 1$ (nous avons en effet fait abstraction du nombre entier nul)(*)

Notons par N_n le nombre moyen de chiffres à vérifier pour déterminer le signe de nombres entiers a étant supposé que $L_B(a)=n$. Nous avons $N_n < (1-B^{-1})^{-2}$.

PREUVE

*Si n=1, il est évident que $N_n=N_1=1$ puisque tout entier de longueur 1 est représenté par (a_o) avec $a_o \neq \phi$.
Dans ce cas, on a bien $N_1=N_n < (1-B^{-1})^{-2}$ puisque $(1-B^{-1})^{-2} = \frac{B^2}{(B-1)^2}$ est strictement supérieur à 1.

*Si $n \geqslant 2$

Le nombre d'entiers de longueur n pour lesquels k chiffres doivent être vérifiés(pour pouvoir déterminer le signe de ces entiers)est

a) $2(B-1)$ pour $k=n$

b) $2(B-1)^2 B^{n-k-1}$ pour $1 \leqslant k < n$

Pour se convaincre de a), il suffit de remarquer que si n chiffres doivent être envisagés, c'est que les n-1 chiffres les moins significatifs $a_o,a_1,\ldots,a_{n-2}$ sont nuls. Il y a donc autant de nombres entiers à considérer que de configurations possibles pour a_{n-1}. Or ce nombre est égal à $2(B-1)$ puisque $-(B-1) \leqslant a_{n-1} \leqslant B-1$ et que $a_{n-1} \neq \phi$.

(*) Remarquons que même pour n=0, la formule $N_n < (1-B^{-1})^{-2}$ est valable puisque l'on aurait $N_0=0 < (1-B^{-1})^{-2}$

Pour se convaincre de b), de manière assez analogue, il suffit de remarquer que si k chiffres doivent être envisagés, c'est que les k-1 chiffres les moins significatifs $a_o, a_1, \ldots, a_{k-2}$ sont nuls. Dans un premier temps, ne considérons que les nombres entiers positifs. Pour a_{k-1}, nous avons B-1 possibilités puisque $0 \leq a_{k-1} \leq B-1$ et que $a_{k-1} \neq \phi$. Pour chacun des chiffres $a_k, a_{k+1}, \ldots, a_{n-2}$, nous avons B possibilités puisqu'ils doivent être compris entre 0 et B-1. Enfin, pour a_{n-1}, nous avons seulement B-1 possibilités car $a_{n-1} \neq \phi$. Tenant compte de toutes ces remarques, on en déduit que le nombre de nombres entiers positifs à considérer est égal à $(B-1)B^{n-k-1}(B-1)$, c'est-à-dire $(B-1)^2B^{n-k-1}$.
Si l'on y ajoute les négatifs, on obtient $2(B-1)^2B^{n-k-1}$ nombres entiers à considérer.

Remarquons maintenant qu'il y a $2(B-1)B^{n-1}$ nombres entiers tels que $L_B(a)=n (n \geq 2)$.
Dès lors, le nombre <u>moyen</u> de chiffres(B-entiers)à vérifier pour déterminer le signe d'un entier de B-longueur n est

$$N_n = \frac{\left[\sum_{k=1}^{n-1} k2(B-1)^2B^{n-k-1}\right] + n2(B-1)}{2(B-1)B^{n-1}}$$

$$= \frac{\left[\sum_{k=1}^{n-1} k(B-1)B^{n-1}B^{-k}\right] + nB^{n-1}B^{-n+1}}{B^{n-1}}$$

$$= (B-1)\left(\sum_{k=1}^{n-1} kB^{-k}\right) + nB^{-n+1}$$

$$= \left(\sum_{k=1}^{n-1} kB^{-k+1}\right) + nB^{-n+1} - \sum_{k=1}^{n-1} kB^{-k}$$

$$= \sum_{k=1}^{n} kB^{-k+1} - \sum_{k=1}^{n-1} kB^{-k}$$

$$< \sum_{k=1}^{n} kB^{-k+1} < \sum_{k=1}^{\infty} kB^{-k+1} = \sum_{k=1}^{\infty} kB^{-(k-1)} = \sum_{k=1}^{\infty} k(1/B)^{k-1}$$

$$= \left(\sum_{k=1}^{\infty} (1/B)^{k}\right)' \cdot (-B^2) = \left(\frac{\frac{1}{B} - 0 \cdot \frac{1}{B}}{1-\frac{1}{B}}\right)' \cdot (-B^2) = \left(\frac{1/B}{\frac{B-1}{B}}\right)' \cdot (-B^2) = \left(\frac{1}{B-1}\right)' \cdot (-B^2)$$

$$= \frac{-1}{(B-1)^2} \cdot (-B^2) = \frac{B^2}{(B-1)^2},$$

ce qui termine notre preuve.

Remarquons que si $B=2^{15}$ (par exemple), alors

$$N_n < \frac{B^2}{(B-1)^2} \simeq 1{,}000061$$

Remarquons aussi que si B=2(par exemple), alors

$$N_n < \frac{B^2}{(B-1)^2} = 4$$

Ces deux observations relatives à des valeurs de B assez extrêmes tendrait à prouver qu'en ce qui concerne le choix de B, aussi longtemps que c'est possible, il y a lieu d'augmenter la valeur B, du moins en ce qui concerne le temps d'exécution(moyen) de ISIGN(a,s), ce qui est d'ailleurs conforme à l'intuition. (*)

(*) Nous verrons ultérieurement ce qu'on entend par l'expression "aussi longtemps que possible" en ce qui concerne le choix de B. Signalons dès maintenant que tout naturellement, les valeurs de B sont plafonnées du fait de contraintes matérielles(en relation avec la taille d'un mot mémoire)

Venons-en maintenant à l'algorithme de changement de signe d'un nombre entier. Considérons dès lors l'algorithme INEG (INEG pour "integer negative").

b ← INEG(a) ou si l'on veut INEG(a,b)

-Spécifications

a est un nombre entier. b est un nombre entier.
A la fin de l'exécution de INEG, b sera égal à l'opposé de a, c'est-à-dire b=-a.

-Algorithme

(1) Si a=ϕ alors b ← ϕ (*)
(2) Successivement parcourir $a_0, a_1, \ldots, a_{n-1}$ en faisant pour chaque a_i : $b_i \leftarrow -a_i$

-Preuve de la correction de INEG(a,b)

La preuve de la correction de INEG(a,b) est presque évidente. Remarquons que si a est nul, l'étape (2) se réduit à néant puisque de par notre structure de représentation, la suite représentant le nombre entier nul est vide(( )). A la fin de l'exécution, de par l'étape (1), b représentera bien l'opposé de a.

-Remarques, commentaires

De par notre structure de représentation d'un nombre entier, la prise de l'opposé d'un nombre entier a nécessite la prise de l'opposé de chaque composante a_i de la représentation de a. Ceci n'est pas sans conséquence néfaste sur le temps d'exécution d'une telle opération, comme nous allons le voir. Ainsi, une structure plus intéressante(à ce point de vue)pour représenter un nombre entier a consisterait à introduire dans la représentation de a une composante exclusivement destinée au signe de a.

(*) De par notre structure de représentation, cela signifie que si a est représenté par ( ), alors b aussi.

Néanmoins, l'avantage acquis serait contrebalancé par des effets néfastes dûs à une certaine hétérogénéité d'une pareille structure de représentation.
En ce qui concerne les temps d'exécution de INEG(a,b), nous pouvons écrire, faisant abstraction de l'étape (1) de l'algorithme, que

$$t^{+}_{INEG(a,b)} = t^{-}_{INEG(a,b)} = t^{*}_{INEG(a,b)} \hat{=} L_B(a)$$

Dans le même ordre d'idée que pour l'algorithme INEG, considérons l'algorithme de <u>détermination de la valeur absolue d'un nombre entier</u>, soit <u>IABS</u>(IABS pour "integer absolute").

b ← IABS(a) ou si l'on veut IABS(a,b)

-<u>Spécifications</u>

a est un nombre entier. b est un nombre entier.
A la fin de l'exécution de IABS, b sera égal à la valeur absolue de a, c'est-à-dire b=!a!.

-<u>Algorithme</u>

Remarque : Un algorithme naturel du calcul de la valeur absolue d'un nombre entier consisterait à combiner les algorithmes ISIGN de détermination du signe d'un nombre entier et IABS de détermination de la valeur absolue d'un nombre entier. Néanmoins, pour des raisons évidentes relatives aux temps d'exécution, il est préférable de considérer l'algorithme suivant.

(1) Si a=ϕ alors b ← ϕ
(2) Successivement parcourir $a_0, a_1, \ldots$ en faisant $b_i \leftarrow \phi$ jusqu'à ce que le premier chiffre a_k soit non nul et si $a_k > 0$ alors pour i variant de k à n-1 faire $b_i \leftarrow a_i$ sinon pour i variant de k à n-1 faire $b_i \leftarrow -a_i$

-Preuve de la correction de IABS(a,b)

La preuve de la correction de IABS(a,b) est presque évidente. Remarquons simplement que si a est nul, alors l'étape (2) se réduit à néant puisque de par notre structure de représentation, la suite représentant le nombre entier nul est vide(( )). A la fin de l'exécution, de par l'étape (1), b représentera bien la valeur absolue de a.

-Remarques, commentaires

Une remarque semblable à celle énoncée pour l'algorithme INEG peut être faite. En effet, de par notre structure de représentation, la prise de la valeur absolue de a nécessite la prise de la valeur absolue de chaque composante a_i.(Voir pages 34 et 35)
En ce qui concerne les temps d'exécution de IABS(a,b), nous pouvons écrire, faisant abstraction de l'étape (1) de l'algorithme, que

$$t^{+}_{IABS(a,b)}=t^{-}_{IABS(a,b)}=t^{*}_{IABS(a,b)} \hat{=} L_B(a)$$

Les trois premiers algorithmes étaient relatifs à un seul nombre entier en entrée. L'algorithme que nous allons considérer maintenant est quant à lui relatif à deux nombres entiers en entrée. Il a pour but de comparer deux nombres entiers a et b.
Soit dès lors l'algorithme ICOMP(ICOMP pour "integer comparison").

s ← ICOMP(a,b) ou si l'on veut ICOMP(a,b,s)

-Spécifications

a et b sont deux nombres entiers.
s est un B-entier.
A la fin de l'exécution de ICOMP, la variable s comprendra la valeur

-1 si a<b
0 si a=b
1 si a>b

L'algorithme ICOMP n'étant plus aussi trivial que l'algorithme ISIGN ou INEG ou encore IABS, en guise d'introduction à la présentation de celui-ci, nous allons présenter deux exemples qui feront apparaître les idées principales de cet algorithme.

Exemple 1 B=10
a=-102
a:--:(-2,0,-1) (*)
b=3810
b:--:(0,1,8,3)

On regarde les premiers chiffres (B-entiers) de la représentation de a et de celle de b. $b_0=0$. Dès lors, on ne connaît pas le signe de b. On retient le signe de a_0(-1).
On inspecte les deux chiffres suivants. On trouve ainsi le signe de b (1).
On peut conclure dès maintenant que a<b.

Exemple 2 B=10
a=-12
a:--:(-2,-1)
b=-800
b:--:(0,0,-8)

On regarde les deux premiers chiffres (B-entiers) de la représentation de a et de celle de b. $b_0=0$ et $b_1=0$. Il est impossible de prendre une décision puisqu'on ne connaît pas le signe de b. Mais la fin de la première suite est atteinte(celle représentant a). Dès lors nous savons que la valeur absolue de b est plus grande que celle de a. Dès lors, nous pourrons prendre une décision lorsque nous connaîtrons le signe de b(car le signe de a-b est le signe de -b). Celui-ci est déterminé par inspection de b_2, soit -8 qui est négatif, -8 étant le premier chiffre non nul rencontré dans la suite représentant b.

(*) Pour rappel, :--: signifie "est représenté par"

-Algorithme

(1) u ← ϕ, v ← ϕ, s ← ϕ (k ← -1) (*)

(2) inspecter successivement a_o, b_o puis $a_1, b_1, \ldots$ jusqu'à ce que la fin d'au moins une des deux suites(celles représentant a et b)soit atteinte. A la kième étape(k=0,1,....), faire

u ← sign(a_k) si $a_k \neq \phi$ (**)

v ← sign(b_k) si $b_k \neq \phi$

et vérifier si u.v=-1

s'il en est ainsi alors s ← u et retourner s (et fin d'exécution de l'algorithme)

sinon s ← sign($a_k - b_k$) si $a_k \neq b_k$

(3) si les deux suites sont finies alors retourner s

(4) sinon si la suite correspondant à a est finie(et donc avant celle de b) s ← -ISIGN((b_{k+1},....)) (***)

sinon s ← ISIGN((a_{k+1},....))

-Preuve de la correction de ICOMP(a,b,s)

Remarquons qu'à tout moment u indique le signe du dernier chiffre non nul rencontré dans la suite $a_o, a_1, \ldots$.(c'est-à-dire en allant des chiffres les moins significatifs aux plus significatifs)étant supposé par convention que si aucun chiffre non nul de la représentation de a n'a encore été rencontré(ce qui inclut le cas où a est représenté par ( )), alors ce signe est ϕ.
La même remarque tient pour v relativement aux chiffres $b_o, b_1, \ldots$ de la représentation de b.

(*) u et v sont deux variables intermédiaires. Elles doivent pouvoir recevoir les valeurs -1,0 et 1.

(**)u ← sign(a_k) signifie : mettre dans u le signe de a_k, c'est-à-dire -1 si $a_k < 0$, ϕ si $a_k = 0$ et 1 sinon.

(***) ISIGN((b_{k+1},....)) signifie "recherche du signe du nombre entier représenté par la suite (b_{k+1},....)"

-Si les deux nombres a et b sont nuls, alors, de par notre convention de représentation du nombre nul(représentation par ( )), l'étape (2) se réduit à néant. De par l'étape (3), on retourne la valeur s qui vaut ϕ de par l'étape (1), ce qui est bien correct puisque a=b.

-Si un seul des nombres a et b est nul, alors, pour la même raison que ci-dessus, l'étape (2) se réduit à néant. L'étape (4) est dès lors exécutée. Tenant compte que k=-1(par convention, puisque l'étape (2) se réduit à néant), on a à la fin de l'étape (4):
si a=ϕ(et donc b$\neq\phi$), alors s égale l'opposé du signe de b ce qui est correct puisue a-b=-b
si b=ϕ(et donc a$\neq\phi$), alors s égale le signe de a ce qui est correct puisque a-b=a.

-Considérons maintenant le dernier cas de figure, c'est-à-dire le cas où ni a ni b n'est nul.

C'est dans cette situation que l'étape (2) a toute sa signification. Considérons être à l'étape k de l'étape (2) pour une certain k=ϕ,1,2,...

Si u.v=-1, de par la signification déjà citée de u et v(voir page 39)respectivement et de par notre structure de représentation des nombres entiers, on peut certifier que a et b sont de signes contraires et que la variable u détermine le signe de a parfaitement tandis que v détermine le signe de b parfaitement. a et b étant de signes contraires, a-b est du signe de a puisque a-b=a+(-b). Comme on vient de le dire, u déterminant parfaitement le signe de a(et donc de a-b), on retourne la valeur de s qui n'est rien d'autre que le contenu de u.

Si u.v$\neq$-1

Nous pouvons affirmer qu'à la fin de l'étape k(k=-1,0,1,...), s comprend le signe de a'(k)-b'(k) où $a'(k) = \sum_{j=0}^{k} a_j B^j$ et $b'(k) = \sum_{j=0}^{k} b_j B^j$ (1)

Ceci se démontre facilement par récurrence sur la valeur de k. En effet, si k=-1, on peut affirmer que a'(k)-b'(k)=0 puisque les sommes ne comprennent aucun terme(et donc par convention sont égales à 0). La fin de l'étape -1 n'est en fait que la fin de l'étape (1) de l'algorithme(étape d'initialisation).

Or, de par cette étape d'initialisation, s comprend bien la valeur ϕ à la fin de l'étape -1.
Supposons maintenant que l'affirmation (1) est vraie à la fin de l'étape k. Montrons qu'elle est encore vraie à la fin de l'étape k+1 (on suppose évidemment que cette étape k+1 existe, c'est-à-dire que ni la fin de la représentation de a ni la fin de la représentation de b n'est atteinte et que u.v$\neq$-1 à la fin de l'étape k).
En effet *si $a_{k+1}=b_{k+1}$, on a que a'(k+1)-b'(k+1)=a'(k)-b'(k). Mais comme on n'a pas modifié le contenu de s, on a toujours que s comprend le signe de a'(k+1)-b'(k+1).
*si $a_{k+1}\neq b_{k+1}$, on a que le signe de a'(k+1)-b'(k+1) est le signe de $a_{k+1}-b_{k+1}$. Dès lors, de par l'instruction "s $\leftarrow$ sign(a_k-b_k)" prévue pour pareil cas, on a que l'affirmation (1) est toujours vérifiée à la fin de l'étape k+1. (*)
Les nombres a et b étant finis et par conséquent les suites les représentant étant finies et vu que nous avons déjà considéré le cas où u.v=-1, nous pouvons supposer une terminaison de l'étape (2) avec u.v$\neq$-1.
*Si les deux suites sont finies, nous pouvons retourner la valeur de s puisque de par l'invariant, s comprend le signe de a'(k)-b'(k) qui est égal au signe de a-b puisque $a=\sum_{j=0}^{k} a_j B^j$ et $b=\sum_{j=0}^{k} b_j B^j$ avec k=n-1. (**)
*Si une des deux suites n'est pas finie (et une seulement), alors
si la suite correspondant à a est finie, alors il est évident que b étant dominant en valeur absolue, le signe de a-b est l'opposé du signe du nombre entier représenté par (b_{k+1},....), sous-suite restante de la représentation de b.
si la suite correspondant à b est finie, alors il est évident que a étant dominant en valeur absolue, le signe de a-b est le signe du nombre entier représenté par (a_{k+1},....), sous-suite restante de la représentation de a.

(*)Nous pouvons donc affirmer que la phrase (1) de la page 39 constitue un invariant (à l'endroit en question dans l'algorithme)
(**)n est la B-longueur de a (et donc de b)

-Remarques,commentaires

Relativement aux différents temps d'exécution de ICOMP(a,b,s), nous pouvons émettre les remarques suivantes. Le mieux que l'on puisse obtenir au niveau du temps d'exécution(c'est-à-dire $t^{-}_{ICOMP(a,b,s)}$) arrive lorsque a et b sont nuls. Mais bien évidemment, cette situation se doit d'être considérée comme "académique". (Elle n'est en effet nullement représentative d'une situation générale). Un cas de terminaison rapide(quasi immédiate même) de l'algorithme ICOMP(a,b,s) intervient lorsque a et b sont de signes contraires et que simultanément le chiffre le moins significatif de la représentation de a en base B et le chiffre le moins significatif de la représentation de b en base B sont non nuls.
A l'opposé, on a une situation tout à fait défavorable, à savoir celle où a(respectivement b) est de B-longueur strictement supérieure à celle de b(respectivement a)et que tous les chiffres a_o, $a_1,\ldots,a_{n-2}$(respectivement $b_o,b_1,\ldots.b_{m-2}$)de la représentation de a en base B(respectivement b)sont nuls.
$t^{+}_{ICOMP(a,b,s)}$est atteint dans pareil cas.
Si nous considérons deux nombres entiers a et b de B-longueurs fixées, nous pouvons dire que dans la moitié des cas, la comparaison des nombres a et b nécessite la considération de seulement a_o et b_o, c'est-à-dire du chiffre le moins significatif de a et de celui de b, ceci arrivant quand a et b sont de signes contraires. Dans l'autre moitié des cas, c'est-à-dire quand a et b sont de mêmes signes, la comparaison de ces deux nombres nécessite l'inspection d'un nombre de chiffres des représentations de a et b égal à $\min(L_B(a),L_B(b))$.
Tout ceci nous permet d'écrire que

$$t^{*}_{ICOMP(a,b,s)} \hat{=} K.\frac{(1+\min(L_B(a),L_B(b)))}{2} \hat{=} K'.\min(L_B(a),L_B(b))$$

Nous arrivons maintenant aux quatre algorithmes considérés comme fondamentaux, à savoir ceux relatifs à l'addition, la soustraction, la multiplication et la division, les quatre algorithmes présentés jusqu'ici, à savoir ISIGN, INEG, IABS et ICOMP n'étant en fait, comme nous allons le voir, que des algorithmes relatifs à des sous-problèmes attachés à la conception de ceux-ci.

Nous arrivons tout naturellement(celui-ci étant nécessaire à la conception des suivants)à devoir commencer par présenter l'algorithme d'addition, soit ISUM(ISUM pour "integer sum").

c ← ISUM(a,b) ou si l'on veut ISUM(a,b,c)

-Spécifications

a et b sont des nombres entiers. c est un nombre entier. A la fin de l'exécution de ISUM, c sera égal à la somme de a et b. (*)

-Algorithme

```
s1 ← ISIGN(a), s2 ← ISIGN(b)                          (**)
si s1=ϕ alors c ← b
sinon si s2=ϕ alors c ← a
      sinon si s1=s2 alors c ← ISUM1(a,b)
            sinon           c ← ISUM2(a,b)
```

où ISUM1 et ISUM2 sont deux sous-algorithmes présentés ci-après.

(*) De manière plus précise, il y a lieu de comprendre qu'à la fin de l'exécution de ISUM, on connaîtra la représentation de c, somme des entiers a et b dont on connaît les représentations respectives.

(**) Remarquons que nous utilisons ici l'algorithme de détermination du signe d'un nombre entier.

-Preuve de la correction de ISUM(a,b,c)

La preuve de la correction de ISUM(a,b,c) est évidente dès que l'on admet que les algorithmes ISUM1 et ISUM2 satisfont à leur spécification, ce qui sera prouvé.

-Remarques, commentaires

Comme l'indique l'algorithme présenté page 42, le problème de l'addition de deux nombres entiers est divisé en plusieurs sous-problèmes.
On détermine avant toute chose les signes de a et b. Remarquons que les variables intermédiaires s1 et s2 doivent pouvoir, conformément aux spécifications de l'algorithme ISIGN être en mesure de recevoir les valeurs -1, ϕ et 1(seule contrainte imposée à ces variables intermédiaires).
Si l'un au moins des deux nombres est nul, la valeur de la somme des deux nombres entiers n'est rien d'autre que l'autre nombre entier(éventuellement nul, lui aussi). Le cas où l'un au moins des deux nombres est nul peut être considéré comme le premier sous-problème du problème de la recherche de la somme de deux nombres entiers. Un deuxième sous-problème, à part entière, est celui de la recherche de la somme de deux nombres entiers non nuls et de mêmes signes, le troisième sous-problème étant celui de la recherche de la somme de deux nombres entiers non nuls et de signes contraires.
La raison de la considération(ou de la distinction)des sous-problèmes deux et trois apparaîtra de manière évidente lors de la présentation des algorithmes ISUM1 et ISUM2 les résolvant.
Quant au sous-problème un, la raison de sa considération(de sa conception) est moins évidente. Il est vrai, en effet, que algorithmiquement parlant, une solution plus élégante eût consisté à inclure la solution du sous-problème un dans celle du sous-problème deux, autrement dit de considérer la solution du sous-problème un comme un cas particulier de la solution du sous-problème deux. Pour des raisons relatives aux temps d'exécution, la technique consistant à considérer le sous-problème un à part entière a été retenue.(Pour s'en convaincre, on se référera à l'algorithme ISUM1)

La raison de ce choix pourrait être critiquée, argumentant par le fait que traitant des grands nombres entiers, la probabilité d'avoir un des deux termes nul est assimilable à zéro. Ne négligeons cependant pas le fait que l'addition de deux nombres entiers peut intervenir comme sous-problème de problèmes plus généraux(par exemple le produit)et que dès lors une telle assimilation n'est plus du tout aussi évidente et pourrait même être franchement contradictoire.

Comme annoncé ci-dessus, présentons les algorithmes ISUM1 et ISUM2 correspondant respectivement aux sous-problèmes deux et trois esquissés ci-avant.(ISUM1 pour "integer sum, first part")
(ISUM2 pour "integer sum, second part")

c ← ISUM1(a,b) ou si l'on veut ISUM1(a,b,c)

-Spécifications

a et b sont des nombres entiers non nuls et de mêmes signes.
c est un nombre entier(non nul et du signe de a(ou b)).
A la fin de l'exécution de ISUM1, c sera égal à la somme de a et b.
On suppose également que initialement, avant exécution de ISUM1(a,b,c), la variable s1 comprend le signe de a(c'est-à-dire aussi de b).

-Algorithme

(1) REPORT $\leftarrow \phi$
si s1=1 alors ADAPTATEUR $\leftarrow$ -B
sinon ADAPTATEUR $\leftarrow$ B

(2) Parcourir successivement a_o, b_o puis $a_1, b_1, \ldots$ jusqu'à ce que la fin d'au moins une des deux suites soit atteinte. A la kième étape(k=0,1,....) faire

$c_k \leftarrow a_k + b_k +$REPORT
si $!c_k! \geq B$ alors $c_k \leftarrow c_k +$ADAPTATEUR, REPORT $\leftarrow$ s1
sinon REPORT $\leftarrow \phi$

(3) Parcourir successivement les chiffres restants de la plus longue entrée(pour fixer les idées, disons a)et pour chacun d'eux faire (on suppose $L_B(a) \geq L_B(b)$)

$c_k \leftarrow a_k +$REPORT
si $!c_k! \geq B$ alors $c_k \leftarrow c_k +$ADAPTATEUR, REPORT $\leftarrow$ s1
sinon REPORT $\leftarrow \phi$, passer à l'étape (4)(et donc arrêt du parcours)

(4) Parcourir successivement les chiffres restants de la plus longue entrée(nous avons dit a) et pour chacun d'eux faire

$c_k \leftarrow a_k$

(5) si REPORT$\neq\phi$ alors k $\leftarrow$ k+1, $c_k \leftarrow$REPORT

Présentons, de manière succincte, l'exécution imagée de l'algorithme sur deux exemples.

Exemple1 B=10

Addition des nombres entiers positifs a et b représentés en base 10 respectivement par 9874501 et 9389.

a:--:(1,ø,5,4,7,8,9)

b:--:(9,8,3,9)

sens d'évolution

⟶

REPORT	ø	1	ø	ø	1	ø
chiffre	1	ø	5	4	7	8	9
de a							
chiffre	9	8	3	9			
de b							
chiffre	ø	9	8	3	8	8	9
du ré-							
sultat							

c:--:(ø,9,8,3,8,8,9)

donc le résultat c=a+b est représenté en base 10 par 9883890.

Exemple2 B=10

Addition des nombres entiers négatifs a et b représentés en base 10 respectivement par -9874501 et -9389.

a:--:(-1,ø,-5,-4,-7,-8,-9)

b:--:(-9,-8,-3,-9)

sens d'évolution

⟶

REPORT	ø	-1	ø	ø	-1	ø
chiffre de a	-1	ø	-5	-4	-7	-8	-9
chiffre de b	-9	-8	-3	-9			
chiffre du	ø	-9	-8	-3	-8	-8	-9
résultat							

c:--:(ø,-9,-8,-3,-8,-8,-9)

donc le résultat est représenté en base 10 par -9883890.

-Preuve de la correction de ISUM1(a,b,c)

Comme signalé au cours de l'algorithme, nous pouvons sans perte de généralité supposer que $L_B(a) \geq L_B(b)$. (*)

Supposons dans un premier temps a et b positifs.

L'étape (1) de l'algorithme présenté page 45 constitue l'étape d'initialisation.

L'étape (2) est en fait une boucle. Nous y considérons successivement les paires de chiffres (a_k, b_k), coefficients dans les décompositions de a et b comme puissances de B de la puissance $B^k (k=0,1,2,....\min(m-1,n-1))$ (**)

Il est facile de montrer qu'à la fin de la kième étape de cette étape (2) de l'algorithme$(k=0,1,....,\min(m-1,n-1))$, la suite $(c_o, c_1, c_2, ..., c_k, \text{REPORT})$ est une représentation sous forme de suite du nombre entier défini par $(\sum_{i=0}^{k} a_i B^i + \sum_{i=0}^{k} b_i B^i)$.(1)

(Attention, il ne s'agit pas nécessairement de la représentation "standard" au sens défini page 23, car nous acceptons ici des éventuels chiffres ϕ non significatifs)

En effet, à la fin de la Oième étape, l'affirmation (1) est respectée. (***)

Pour s'en convaincre, observons le raisonnement suivant. A la fin de la Oième étape, $\sum_{i=0}^{k} a_i B^i + \sum_{i=0}^{k} b_i B^i = a_o + b_o$. Puisque

a et b sont positifs(et donc $a_o + b_o \geq 0$), on a de par l'étape d'initialisation que la variable ADAPTATEUR comprend la valeur -B. De par le corps de la boucle de l'étape (2), si $a_o + b_o \geq B$, sachant que de toute façon $a_o + b_o < 2B$(puisque $a_o \leq B-1$ et $b_o \leq B-1$), on aura à la fin de la Oième étape

$c_o = a_o + b_o - B$(et donc $0 \leq c_o \leq B-1$) et REPORT=1.

(*)Lors de l'implémentation "pratique", il suffira de prévoir un simple "SWITCH" de sorte à substituer aux opérations ne faisant intervenir que les chiffres a_k les opérations ne faisant intervenir que les chiffres b_k au cas où $L_B(a) < L_B(b)$. Ainsi, par exemple, l'opération "$c_k \leftarrow a_k$+REPORT" de l'étape (3) de l'algorithme serait remplacée par "$c_k \leftarrow b_k$+REPORT".

(**)$a = \sum_{k=0}^{m-1} a_k B^k$ et $b = \sum_{k=0}^{n-1} b_k B^k$ avec $a_{m-1} \neq \phi$ et $b_{n-1} \neq \phi$.

(***)De par les spécifications de l'algorithme, nous sommes certains que cette Oième étape existe.

La suite $(c_o, REPORT) = (c_o, 1)$ est donc bien la représentation de $a_o + b_o$. Par contre, si $a_o + b_o < B$, sachant que de toute façon $a_o + b_o \geq 0$, on aura à la fin de la 0ième étape $c_o = a_o + b_o$ (et donc $0 \leq c_o \leq B-1$) et $REPORT = \emptyset$.
La suite $(c_o, REPORT) = (c_o, \emptyset)$ est donc bien une représentation de $a_o + b_o$. (*)
Montrons maintenant que si l'affirmation (1) page 47 est respectée à la fin de la jième étape, elle l'est encore à la fin de la (j+1)ième. $(j \leq \min(m-1, n-1)-1)$
A la fin de la jième étape, on a donc par hypothèse que $(c_o, c_1, c_2, \ldots, c_j, REPORT_{(j)})$ est une représentation du(**) nombre entier défini par $(\sum_{i=0}^{j} a_i B^i + \sum_{i=0}^{j} b_i B^i)$. (***)

De par le corps de la boucle de l'étape (2),
si $a_{j+1} + b_{j+1} + REPORT_{(j)} \geq B$, sachant que de toute façon $a_{j+1} + b_{j+1} + REPORT_{(j)} \leq 2B-1$ (puisque $a_{j+1} \leq B-1$ et $b_{j+1} \leq B-1$ et $REPORT_{(j)} \leq 1$), on aura à la fin de la (j+1)ième étape $c_{j+1} = a_{j+1} + b_{j+1} + REPORT_{(j)} - B$ (et donc $0 \leq c_{j+1} \leq B-1$) et $REPORT_{(j+1)} = 1$.
La suite $(c_o, c_1, c_2, \ldots, c_j, c_{j+1}, REPORT_{(j+1)}) = (c_o, c_1, \ldots, c_j, c_{j+1}, 1)$ est donc bien la représentation de $\sum_{i=0}^{j+1} a_j B^j + \sum_{i=0}^{j+1} b_i B^i$. (****)
Par contre, si $a_{j+1} + b_{j+1} + REPORT_{(j)} < B$, sachant que de toute

(*) Insistons sur le fait qu'il s'agit d'une représentation et non de la représentation standard au sens de la page 23 car le chiffre REPORT, coefficient de la plus haute puissance de B n'est pas non nul.

(**) Par $REPORT_{(j)}$, nous entendons le contenu de REPORT à la fin de l'étape j.

(***) et donc que $(\sum_{i=0}^{j} c_i B^i) + REPORT_{(j)} B^{j+1} = \sum_{i=0}^{j} a_i B^i + \sum_{i=0}^{j} b_i B^i$.

(****) Puisque $\sum_{i=0}^{j+1} c_i B^i + B^{j+2} = \sum_{i=0}^{j} c_i B^i + c_{j+1} B^{j+1} + B^{j+2} =$

$\sum_{i=0}^{j} c_i B^i + (a_{j+1} + b_{j+1} + REPORT_{(j)} - B) B^{j+1} + B^{j+2} = \sum_{i=0}^{j} a_i B^i + \sum_{i=0}^{j} b_i B^i + a_{j+1} B^{j+1} + b_{j+1} B^{j+1}$

$= \sum_{i=0}^{j+1} a_i B^i + \sum_{i=0}^{j+1} b_i B^i$.

façon $a_{j+1}+b_{j+1}+REPORT_{(j)} \geq 0$, on aura à la fin de la (j+1)ième étape $c_{j+1}=a_{j+1}+b_{j+1}+REPORT_{(j)}$ (et donc $0 \leq c_{j+1} \leq B-1$) et REPORT=ϕ. La suite $(c_o,c_1,\ldots,c_j,c_{j+1},REPORT)=(c_o,c_1,\ldots,c_j,c_{j+1},\phi)$ est donc bien une représentation de $\sum_{i=0}^{j+1} a_i B^i + \sum_{i=0}^{j+1} b_i B^i$. (*)

L'étape (3) est également une boucle. Nous y considérons successivement les éventuels chiffres restants de la représentation de a, ceci tant que REPORT$\neq\phi$ (excepté pour l'éventuel premier chiffre restant).
A la fin de l'étape (3), nous avons que $(c_o,c_1,\ldots,c_l,REPORT)$ est une représentation de $\sum_{i=0}^{\min(m-1,n-1)} (a_i B^i + b_i B^i) + \sum_{i=\min(m-1,n-1)+1}^{l} a_i B^i$ (**)

où l est le plus petit entier compris entre min(m-1,n-1)+1 et max(m-1,n-1) pour lequel REPORT=ϕ à la <u>fin</u> d'une étape dans l'étape (3) de l'algorithme.

(*) Puisque

$$\sum_{i=0}^{j+1} c_i B^i + \phi B^{j+2} = \sum_{i=0}^{j} c_i B^i + c_{j+1} B^{j+1}$$

$$= \sum_{i=0}^{j} c_i B^i + (a_{j+1}+b_{j+1}+REPORT_{(j)}) B^{j+1}$$

$$= \sum_{i=0}^{j} c_i B^i + REPORT_{(j)} B^{j+1} + a_{j+1} B^{j+1} + b_{j+1} B^{j+1}$$

$$= \sum_{i=0}^{j} a_i B^i + \sum_{i=0}^{j} b_i B^i + a_{j+1} B^{j+1} + b_{j+1} B^{j+1}$$

$$= \sum_{i=0}^{j+1} a_i B^i + \sum_{i=0}^{j+1} b_i B^i$$

(**) Par hypothèse que $L_B(a) \geq L_B(b)$, on a $\min(m-1,n-1)=n-1$. Remarquons qu'un tel l peut ne pas exister auquel cas la seconde somme est nulle. Dans pareil cas, par convention $(c_o,c_1,\ldots,c_l,REPORT)=(c_o,c_1,\ldots,c_{\min(m-1,n-1)},REPORT)$, c'est-à-dire que l'on pose $l=\min(m-1,n-1)$

Cette affirmation se démontre de manière tout à fait analogue à ce qui a été fait pour l'étape (2), après avoir posé artificiellement $b_k=\phi$. $(k=\min(m-1,n-1)+1,\ldots,l)$
(On remarque d'ailleurs qu'avec cette convention, les instructions de l'étape (3) se confondent avec celles de l'étape (2), au point de vue du texte).
L'étape (4) quant à elle inspecte les chiffres restants de la plus longue entrée. Il est facile de montrer qu'à la fin de l'étape (4), la suite $(c_o,c_1,\ldots,c_{\max(m-1,n-1)},\text{REPORT})$ (*) est une représentation de a+b (1). En effet, à la sortie de l'étape (3), nous savons que $(c_o,c_1,\ldots,c_l,\text{REPORT})$ est une représentation de $\sum_{i=0}^{\min(m-1,n-1)}(a_iB^i+b_iB^i)+\sum_{i=\min(m-1,n-1)+1}^{l}a_iB^i$ avec l défini page 49.
Si l'étape (3) se réduit à néant(c'est-à-dire si $L_B(a)=L_B(b)$ et donc $\max(m-1,n-1)=\min(m-1,n-1)$), il est évident que l'étape (4) se réduit aussi à néant et évidemment l'affirmation (1) est vérifiée. Sinon, si à la sortie de l'étape (3), REPORT$\neq\phi$, (**) l'étape (4) se reduit à néant et l'affirmation (1) est encore vérifiée et sinon, un raisonnement analogue mais simplifié à ce qui a été fait pour l'étape (2) peut être fait(démonstration par récurrence sur la valeur de k).
Venons-en enfin à l'étape (5) de l'algorithme. Il est aisé de voir que si REPORT=ϕ à la fin de l'étape (4), nécessairement $c_{\max(m-1,n-1)}\neq\phi$. $(c_o,c_1,\ldots,c_{\max(m-1,n-1)})$ est la (***) représentation de a+b. Sinon, c'est le cas pour
$(c_o,c_1,\ldots,c_{\max(m-1,n-1)},\text{REPORT})=(c_o,c_1,\ldots,c_k)$ (****)

(*)Par hypothèse que $L_B(a)\geq L_B(b)$, on a que $\max(m-1,n-1)=m-1$.
(**)Ceci n'arrive que lorsqu'un seul chiffre de la représentation de a reste à considérer au cours de l'étape (3).

(***)En effet $\sum_{i=0}^{m-1}a_iB^i+\sum_{i=0}^{n-1}b_iB^i=\sum_{i=0}^{\max(m-1,n-1)}c_iB^i+\text{REPORT}.B^{\max(m-1,n-1)+1}$

$\geq B^{\max(m-1,n-1)}$

(****) Ces résultats découlent immédiatement de l'affirmation vraie à la fin de l'étape (4)(voir (1) ci-dessus) et du fait qu'à la fin de l'étape (4), $k=\max(m-1,n-1)$.

Enfin, signalons que si a et b sont négatifs, une preuve strictement similaire peut être établie, étant donné que lors de l'étape d'initialisation, la variable ADAPTATEUR a été mise à la valeur B. Remarquons que dans ce cas, l'instruction "REPORT ← s1" est équivalente à "REPORT ← -1" et que la condition "$!c_k!\geq B$" est équivalente à "$c_k \leq -B$".

-Remarques, commentaires

Remarquons que la variable REPORT doit être capable de recevoir la valeur -1, ϕ ou 1.(Seule contrainte imposée à cette variable) Quant à la variable ADAPTATEUR, elle doit être en mesure de recevoir la valeur -B ou B.(Egalement seule contrainte imposée à cette variable)
Signalons aussi que les étapes (3) et (4) auraient pu être confondues à l'étape (2), à condition de poser judicieusement et artificiellement $b_k=\phi$ ou $a_k=\phi$(pour k=min(m-1,n-1)+1,..,max(m-1,n-1)) suivant le cas. Cette solution n'a pas été retenue pour des raisons évidentes d'amélioration des temps d'exécution, le gain étant important essentiellement lorsque $L_B(a) \ll L_B(b)$ ou $L_B(a) \gg L_B(b)$
En ce qui concerne ces temps d'exécution, remarquons que $t_{ISUM1(a,b,c)} \hat{=} K.\min(L_B(a),L_B(b))$ où K est une constante de proportionnalité, ceci si l'on peut négliger le temps dû aux "copies" réalisées par les instructions "$c_k \leftarrow a_k$"(respectivement "$b_k \leftarrow b_k$")de l'étape (4) de l'algorithme,c'est-à-dire relatives aux chiffres restants de la plus longue entrée et pour lesquels le report a cessé d'intervenir. (*)
Dans le cas où on ne peut pas négliger ce temps, on est forcé d'écrire que $t_{ISUM1(a,b,c)} \hat{=} K.\max(L_B(a),L_B(b))$.

(*) Il n'apparaît pas de manière évidente que ce temps soit négligeable. On trouve peut-être ici et à ce point de vue un des avantages de la technique de la contiguïté physique par rapport à celle des listes. Remarquons aussi que dans pareil cas, nous avons avantage, de ce point de vue, à considérer une base B grande car au plus B est grande, au plus vite(en moyenne)l'influence du report aura cessé, encore que de toute façon, généralement, on puisse assurer que le report ne se propagera pas bien loin au delà du chiffre le plus significatif de la plus petite entrée. C'est d'ailleurs pour cette raison que l'on pouvait écrire $t_{ISUM1(a,b,c)} \hat{=} K.\min(L_B(a),L_B(b))$.

Venons-en à l'algorithme ISUM2.

c ← ISUM2(a,b) ou si l'on veut ISUM2(a,b,c)

-Spécifications

a et b sont des nombres entiers non nuls et de signes contraires. c est un nombre entier.
A la fin de l'exécution de ISUM2, c sera égal à la somme de a et b.

-Algorithme

(1) $u \leftarrow \phi$ et parcourir successivement a_o, b_o puis $a_1, b_1, \ldots$ jusqu'à ce que la fin d'au moins une des deux suites soit atteinte. A la kième étape(k=0,1,...) faire

$c_k \leftarrow a_k + b_k$, $u \leftarrow sign(c_k)$ si $c_k \neq \phi$, $l \leftarrow k$

(2) si $u=\phi$, alors si les deux suites sont finies (*)

alors $c \leftarrow \phi$

sinon parcourir chacun des chiffres restants de la plus longue entrée(pour fixer les idées disons a)et pour chacun d'eux faire(on suppose $L_B(a) \geq L_B(b)$)

$c_k \leftarrow a_k$ $(k=l+1, l+2, \ldots, \max(m-1, n-1))$

(3) sinon (4) si les deux suites sont finies alors $s \leftarrow u$

sinon $s \leftarrow$ ISIGN(a) (on suppose $L_B(a) \geq L_B(b)$)

(5) si $u \neq s$ alors faire $c_{k+1} \leftarrow a_{k+1}, \ldots, c_v \leftarrow a_v$

jusqu'à ce que $a_v \neq \phi$

$l \leftarrow v$

(6) REPORT $\leftarrow \phi$, si s=1 alors ADAPTATEUR $\leftarrow$ B

sinon ADAPTATEUR $\leftarrow$ -B

pour i allant de 0 à l faire

$d \leftarrow c_i$+REPORT

si $d*s<0$ alors $c_i \leftarrow$ d+ADAPTATEUR, REPORT $\leftarrow -s$

sinon REPORT $\leftarrow \phi$, $c_i \leftarrow$ d

(7) pour i allant de l+1 à m-1 faire $c_i \leftarrow a_i$ (**)

(8) supprimer de notre liste représentant c les éventuels zéros non significatifs

(*) c'est-à-dire m-1=n-1 et donc m=n

(où $a=\sum_{k=0}^{m-1} a_k B^k$ *et* $b=\sum_{k=0}^{n-1} b_k B^k$ *avec* $a_{m-1} \neq \phi$ *et* $b_{n-1} \neq \phi$*)*

(**) il est à remarquer que l+1 peut être strictement plus grand que m-1.

Dans un but de familiarisation avec l'algorithme de la page 53 et avant de donner une preuve de sa correction, présentons ce que donnerait l'exécution de ce dernier sur les quelques exemples suivants.(Nous y considérons sans cesse la base B égale à dix)

Exemple1

Addition des nombres entiers a et b représentés par -2095 et 2095 respectivement.

a:--:(-5,-9,0,-2)
b:--:(5,9,0,2)
En exécutant l'étape (1), on obtient (0,0,0,0) et donc u=ϕ et les deux suites sont finies. Par conséquent, on retourne comme résultat la suite représentant ϕ, ce qui est symbolisé par l'instruction "c $\leftarrow \phi$" de l'algorithme(voir étape (2)).

Exemple2

Addition des nombres entiers a et b représentés par -1232095 et 2095 respectivement.

a:--:(-5,-9,0,-2,-3,-2,-1)
b:--:(5,9,0,2)
En exécutant l'étape (1), on obtient (0,0,0,0) et donc u=ϕ et la suite correspondant à b est finie. Comme celle correspondant à a n'est pas finie, on effectue au cours de l'étape (2) une copie des chiffres restants. On obtient ainsi le résultat (0,0,0,0,-3,-2,-1), ce qui est bien la représentation de c=a+b représenté en base 10 par -123000.

Exemple3

Addition des nombres entiers a et b représentés par -1232095 et 2096 respectivement.

a:--:(-5,-9,0,-2,-3,-2,-1)
b:--:(6,9,0,2)

A la fin de l'exécution de l'étape (1), on obtient (1,0,0,0) et donc u=1 et l=3(et k=3).
La suite correspondant à a n'est pas finie, dès lors, après exécution de l'étape (4), on a s=-1. Comme dès lors u≠s, à la fin de l'exécution de l'étape (5), on a la suite (1,0,0,0,-3) et l=4. On exécute ensuite l'étape (6), ce qui donne (-9,-9,-9,-9,-2). Après exécution de l'étape (7), on obtient la suite (-9,-9,-9,-9,-2,-2,-1) ce qui fournit le résultat puisqu'il n'y a pas de ∅ non significatifs à supprimer. Remarquons que cette suite représente bien a+b représenté en base 10 par -1229999.

Exemple4

Addition des nombres entiers a et b représentés par -100000 et 99998 respectivement.

a:--:(0,0,0,0,0,-1)
b:--:(8,9,9,9,9)

A la fin de l'étape (1), on obtient (8,9,9,9,9) et donc u=1 et l=4(et k=4). La suite correspondant à a n'est pas finie, dès lors, après exécution de l'étape (4), on a s=-1.
Comme u≠s, à la fin de l'exécution de l'étape (5), on a la suite (8,9,9,9,9,-1) et l=5. On exécute ensuite l'étape (6), ce qui donne la suite (-2,0,0,0,0,0). L'étape (7) se réduit à néant puisque les chiffres de la plus longue suite ont été considérés. Après suppression des zéros non significatifs (étape (8)), on obtient la suite (-2) qui représente bien le résultat puisque a+b est représenté en base 10 par -2.

Exemple5

Addition des nombres entiers a et b représentés par -1232095 et 2094 respectivement.

a:--:(-5,-9,0,-2,-3,-2,-1)
b:--:(4,9,0,2)
A la fin de l'exécution de l'étape (1), on obtient (-1,0,0,0) et donc u=-1 et l=3(et k=3). La suite correspondant à a n'est pas totalement parcourue, dès lors, après exécution de l'étape (4), on a s=-1. Comme u=s, l'étape (5) se réduit à néant et à la fin de l'exécution de l'étape (6), on a toujours la suite (-1,0,0,0). Après exécution de l'étape (7), on obtient la suite (-1,0,0,0,-3,-2,-1) qui représente bien a+b représenté en base 10 par -1230001 puisqu'aucun chiffre non significatif n'est présent.

Exemple6

Addition des nombres entiers a et b représentés par -2947 et 2939 respectivement.
a:--:(-7,-4,-9,-2)
b:--:(9,3,9,2)

Après exécution de l'étape (1), on obtient la suite (2,-1,0,0) et donc u=-1 et l=3(et k=3). u n'étant pas nul et les deux suites étant terminées, à la fin de l'exécution de l'étape(4), on a s=u=-1. L'étape (5) se réduit à néant. Après exécution de l'étape (6), on obtient la suite (-8,0,0,0). L'étape (7) se réduit aussi à néant. Enfin, de par l'étape (8) de suppression des éventuels zéros non significatifs, on obtient la suite (-8) qui représente a+b représenté en base 10 par -8.

-Preuve de la correction de ISUM2(a,b,c)

Comme signalé au cours de l'algorithme, nous pouvons sans perte de généralité supposer que $L_B(a) \geq L_B(b)$. (*)
Considérons l'étape (1) de l'algorithme présenté page 53.
Il est trivial de voir qu'à la fin de l'exécution de cette étape, $l=\min(m-1,n-1)$. (**)
La variable u quant à elle indique à la fin de l'exécution toujours de cette même étape le signe de $\sum_{k=0}^{l}(a_k+b_k)B^k$. (1)
Montrons cette affirmation par récurrence sur la valeur de k.
Remarquons qu'initialement u comprend la valeur ϕ.
A la fin de l'étape -1, avant tout parcours donc des chiffres des représentations de a et b, l'affirmation (1) est donc vraie puisque $\sum_{k=0}^{-1}(a_k+b_k)B^k=\phi$. (***)
Supposons maintenant que l'affirmation (1) soit vraie à la fin de l'étape k et montrons qu'étant supposé que $k<\min(m-1,n-1)$ elle l'est encore à la fin de l'étape k+1. $k<\min(m-1,n-1)$, dès lors l'étape k+1 existe. Par hypothèse
$u_{(k)}=\text{signe}(\sum_{i=0}^{k}(a_i+b_i)B^i)$. Si $c_{k+1}=a_{k+1}+b_{k+1}=\phi$, alors $u_{(k+1)}=u_{(k)}$ (****)
indique bien le signe de $\sum_{i=0}^{k+1}(a_i+b_i)B^i$ puisque $\sum_{i=0}^{k+1}(a_i+b_i)B^i=\sum_{i=0}^{k}(a_i+b_i)B^i$. Dans le cas contraire, il en est de même puisque $u_{(k+1)}=\text{signe}(a_{k+1}+b_{k+1})=\text{signe}(\sum_{i=0}^{k+1}(a_i+b_i)B^i)$ car

$$\left|(a_{k+1}+b_{k+1})B^{k+1}\right| \geq B^{k+1} > \left|\sum_{i=0}^{k}(a_i+b_i)B^i\right|.$$

(*) L'algorithme proposé est trivialement adaptable pour a et b tels que $L_B(a)$ soit quelconque par rapport à $L_B(b)$.

(**) Pour rappel, $a=\sum_{k=0}^{m-1}a_kB^k$ $(a_{m-1}\neq\phi)$ et $b=\sum_{k=0}^{n-1}b_kB^k$ $(b_{n-1}\neq\phi)$

Remarquons que le minimum dont question ci-dessus existe puisque $a\neq\phi$ et $b\neq\phi$.

(***) Puisqu'aucun terme n'existe dans la somme.

(****)Nous entendons par $u_{(k)}$ le contenu de u à la fin de l'étape k.

Considérons à présent l'étape (2). Si $u=\emptyset$, c'est-à-dire $u_{(1)}=\emptyset$, si les deux suites sont finies, cela veut dire que $a+b=\sum_{i=0}^{l}(a_i+b_i)B^i=\emptyset$ puisque $u=\text{signe}(\sum_{i=0}^{l}(a_i+b_i)B^i)$.

Dans pareil cas, il y a donc lieu de faire "$c \leftarrow \emptyset$".
L'algorithme répond donc dans pareil cas à ses spécifications.
Si, toujours si $u=\emptyset$, une des deux suites n'est pas finie(disons celle représentant a), alors

$$a+b=\sum_{i=0}^{\min(m-1,n-1)}(a_i+b_i)B^i + \sum_{i=\min(m-1,n-1)+1}^{\max(m-1,n-1)} a_iB^i = \sum_{i=\min(m-1,n-1)+1}^{\max(m-1,n-1)} a_iB^i$$

puisque $\sum_{i=0}^{\min(m-1,n-1)}(a_i+b_i)B^i = \emptyset$ car $u_{(1)}=\emptyset$ et vu l'affirmation (1) page 57.
De par les contraintes initiales sur les a_i $(i=l+1,..,\max(m-1,n-1))$, de par l'instruction répétée "$c_k \leftarrow a_k$", l'algorithme répond dans pareil cas à ses spécifications.
L'étape (3) regroupe les étapes à exécuter dans le cas où $u_{(1)}\neq\emptyset$. $u_{(1)}\neq\emptyset$, cela signifie que $\sum_{i=0}^{\min(m-1,n-1)}(a_i+b_i)B^i\neq\emptyset$.

Nous avons d'ailleurs démontré que $u_{(1)}$ indique(à la fin de l'étape (2) ou ce qui revient au même avant exécution de l'étape (3)le signe de $\sum_{i=0}^{\min(m-1,n-1)}(a_i+b_i)B^i$. Tenant compte de ceci et des instructions de l'étape (4), il est facile de montrer qu'à la fin de l'exécution de l'étape (4), toujours dans l'hypothèse où $u_{(1)}\neq\emptyset$, la variable s indique le signe de a+b. En effet, dans le cas où les deux suites sont finies, le résultat est évident de par l'instruction "$s \leftarrow u$" prévue dans pareille situation et de par l'affirmation (1) énoncée page 57. Dans le cas contraire, c'est-à-dire quand une des deux suites n'est pas finie(disons celle correspondant à a), le résultat est tout aussi évident de par l'instruction "$s \leftarrow$ISIGN(a)" et vu le fait que

$$\text{signe}(a+b)=\text{signe}\left(\sum_{i=0}^{\min(m-1,n-1)}(a_i+b_i)B^i + \sum_{i=\min(m-1,n-1)+1}^{\max(m-1,n-1)} a_iB^i\right)$$

$$=\text{signe}(a_{m-1}) \text{ car } \left|a_{m-1}B^{m-1}\right| \geq B^{m-1} > \left|\sum_{i=0}^{\min(m-1,n-1)}(a_i+b_i)B^i\right|$$

$$=\text{signe}(a).$$

A la fin de l'étape (5), toujours dans l'hypothèse où $u_{(1)} \neq \emptyset$, la variable 1 comprend la plus petite valeur de j supérieure ou égale à min(m-1,n-1) telle que $\text{signe}(\sum_{i=0}^{j}(a_i+b_i)B^i)=\text{signe}(a+b)$, étant posé par convention que $b_i=\emptyset$ si $i>\min(m-1,n-1)$.
(Nous supposons toujours $L_B(a)\geq L_B(b)$ et donc $\min(m-1,n-1)=n-1$)
En effet, <u>si u=s</u>, le résultat est évident puisque de par l'étape (1) $1=\min(m-1,n-1)$ et vu que $\text{signe}(\sum_{i=0}^{\min(m-1,n-1)}(a_i+b_i)B^i)=u_{(1)}=u=s=$

signe(a+b).

<u>si u≠s</u>, c'est que nécessairement les deux suites représentant respectivement a et b ne sont pas de même longueur(supposons toujours $L_B(a)>L_B(b)$). Nous savons que dans pareil cas, s=signe(a). Mais nous savons aussi par ce qui précède que $u=u_{(1)}=\text{signe}(\sum_{i=0}^{\min(m-1,n-1)}(a_i+b_i)B^i)$ et que s=signe(a+b).(Ceci avant exécution de l'étape(5))
A la fin de l'exécution de l'étape (5), nous aurons trouvé le plus petit 1 plus grand que min(m-1,n-1) et tel que $a_1 \neq \emptyset$.
(Celui-ci existe bien puisque de toute façon $a_{m-1} \neq \emptyset$)
Remarquons pour terminer notre raisonnement que pour ce 1, on a

$$\text{signe}(\sum_{i=0}^{1}(a_i+b_i)B^i) \underset{(*)}{=} \text{signe}(\sum_{i=0}^{\min(m-1,n-1)}(a_i+b_i)B^i + \sum_{i=\min(m-1,n-1)+1}^{1}a_iB^i)$$

$$\underset{(**)}{=} \text{signe}(\sum_{i=0}^{\min(m-1,n-1)}(a_i+b_i)B^i + a_1B^1)=\text{signe}(a_1B^1)=\text{signe}(a_1)=\text{signe}(a)$$

$$=\text{signe}(a+b).$$

Notons au passage que vu le mode d'affectation de c_i, nous avons $c_i=a_i+b_i (i=0,\ldots,1)$ (***) et donc $\sum_{i=0}^{1}(a_i+b_i)B^i=\sum_{i=0}^{1}c_iB^i$,

ceci à la fin de l'exécution de l'étape (5). Mais il est à noter que tous les c_i ne sont pas nécessairement de même signe.

(*)Voir notre convention ci-dessus

(**)Vu la définition de f

(***)Toujours avec la convention $b_i=\emptyset$ si $i>\min(m-1,n-1)$

L'étape (6) de l'algorithme a pour fonction de récrire $\sum_{i=0}^{l} c_i B^i$ sous la forme $\sum_{i=0}^{l} c'_i B^i$ avec cette fois $c'_i \geq 0$ si $a+b>0$

et $c'_i \leq 0$ si $a+b<0$ $(i=0,\ldots,l)$ (*)

La longue preuve suivante le montre.

Montrons tout d'abord par récurrence sur la valeur de j qu'à la fin de l'étape j de l'étape (6), on a

$$\sum_{i=0}^{j} c_i B^i = \sum_{i=0}^{j} c'_i B^i + REPORT_{(j)} B^{j+1} \quad (**)$$

avec $c'_i \geq 0$ si $a+b>0$ et $c'_i \leq 0$ si $a+b<0$ $(\forall i=0,1,\ldots,j)(\forall j=0,\ldots,l)$ (1)

Supposons dans un premier temps $a+b>0$. De par l'initialisation, on a $s=1$ (=signe(a+b)) et ADAPTATEUR = B. Si $c_o \geq 0$, aucune modification n'est apportée à c_o, c'est-à-dire $c'_o = c_o$. De plus, à la fin de l'étape 0, on a $REPORT_{(o)} = \emptyset$. On a donc bien, dans pareil cas, $c_o B^o = c'_o B^o + REPORT_{(o)} B$ avec $c'_o \geq 0$. Si $c_o < 0$, on a que la variable intermédiaire d est mise à la valeur c_o. Comme $d*s<0$ (puisque $s=1$), alors $c'_o = c_o + B$ et $REPORT_{(o)} = -1$. On a donc bien $c_o B^o = (c'_o - B)B^o = c'_o B^o + REPORT_{(o)} B$ avec $c'_o \geq 0$ puisque $c'_o = c_o + B \geq 0$ puisque $c_o \geq -B$ vu que $c_o = a_o + b_o$, $a_o . b_o \leq 0$, $|a_o| \leq B-1$ et $|b_o| \leq B-1$.

L'affirmation (1) ci-dessus est donc vraie pour j=0.

Montrons maintenant que si cette relation est vraie pour j, elle l'est encore pour j+1, étant supposé que $j<l$. Supposons donc que $\sum_{i=0}^{j} c_i B^i = \sum_{i=0}^{j} c'_i B^i + REPORT_{(j)} B^{j+1}$ avec $c'_i \geq 0$ $\forall i=0,1,2,\ldots,j$ et montrons que $\sum_{i=0}^{j+1} c_i B^i = \sum_{i=0}^{j+1} c'_i B^i + REPORT_{(j+1)} B^{j+2}$ avec $c'_i \geq 0$ $\forall i=0,1,\ldots,j+1$.

Remarquons que

$$\sum_{i=0}^{j+1} c_i B^i = \sum_{i=0}^{j} c_i B^i + c_{j+1} B^{j+1} = \sum_{i=0}^{j} c'_i B^i + REPORT_{(j)} B^{j+1} + c_{j+1} B^{j+1} =$$

$$\sum_{i=0}^{j} c'_i B^i + (REPORT_{(j)} + c_{j+1}) B^{j+1}.$$

(*)Remarquons que nous n'avons pas à considérer le cas où a+b=0. Rappelons aussi que le contenu de s n'est rien d'autre à ce moment que le signe de a+b. (s=-1 ou 1)

(**)Nous notons par c'_i le contenu de c_i après l'étape i. Celui-ci n'est modifié d'ailleurs qu'au cours de l'étape i. Par $REPORT_{(j)}$, il y a lieu d'entendre le contenu de la variable REPORT à la fin de l'étape j.

Exécutons symboliquement l'étape j+1.
La variable d est mise à la valeur $c_{j+1}+REPORT_{(j)}$.
Si $c_{j+1}+REPORT_{(j)} \geq 0$ (ce qui est équivalent à la condition d*s$\geq$0), alors à la fin de l'exécution de l'étape j+1, on a $REPORT_{(j+1)}=\emptyset$ et $c'_{j+1}=c_{j+1}+REPORT_{(j)}$. On a donc bien $\sum_{i=0}^{j+1} c_i B^i = \sum_{i=0}^{j+1} c'_i B^i + REPORT_{(j+1)} B^{j+2}$ avec $c'_i \geq 0$ $\forall i=0,1,\ldots,j+1$.
Si $c_{j+1}+REPORT_{(j)} < 0$ (ce qui est équivalent à la condition d*s<0), alors à la fin de l'exécution de l'étape j+1, on a $c'_{j+1}=c_{j+1}+REPORT_{(j)}+B$ et $REPORT_{(j+1)}=-1$.
On a donc $\sum_{i=0}^{j+1} c_i B^i = \sum_{i=0}^{j} c'_i B^i + (c'_{j+1}-B)B^{j+1} = \sum_{i=0}^{j+1} c'_i B^i - B^{j+2}$

$$= \sum_{i=0}^{j+1} c'_i B^i + REPORT_{(j+1)} B^{j+2}.$$

Remarquons que $c'_{j+1} \geq 0$ puisque $c_{j+1} \geq -B-REPORT_{(j)}$ vu que $c_{j+1} \overset{(*)}{=} a_{i+1}+b_{i+1}$, $a_{i+1} \cdot b_{i+1} \leq 0$, $|a_{i+1}| \leq B-1$ et $REPORT_{(j)}=-1$ ou 0. (**)
$|b_{i+1}| \leq B-1$

Remarquons maintenant qu'une démonstration strictement symétrique peut être présentée dans l'hypothèse a+b<0.

Pour terminer complètement notre preuve entamée page 60, il suffit de montrer que $REPORT_{(l)}=\emptyset$ puisqu'alors $\sum_{i=0}^{l} c_i B^i = \sum_{i=0}^{l} c'_i B^i + \emptyset . B^{l+1}$. Puisque de manière strictement symétrique un raisonnement analogue dans le cas où a+b<0 pourrait être établi, nous nous contenterons d'apporter la preuve dans le cas où a+b est positif.
Supposons par l'absurde que $REPORT_{(l)}=-1$. (***)
On aurait alors par ce qui précède

$$\sum_{i=0}^{l} (a_i+b_i)B^i = \sum_{i=0}^{l} c_i B^i = \sum_{i=0}^{l} c'_i B^i + REPORT_{(l)} B^{l+1} = \sum_{i=0}^{l} c'_i B^i + (-1)B^{l+1} < 0,$$

ce qui contredit la définition de l donnée au début de la page 59 puisque a+b>0.

(*)Avec toujours la convention habituelle $b_{i+1}=\emptyset$ si i+1>min(m-1,n-1)
(**) Car nous avons supposé a+b>0.
(***)seule autre valeur possible pour $REPORT_{(l)}$ dans pareil cas.

Nous allons établir maintenant et de manière fort heureuse d'ailleurs que les c_i' dont question à la page 60 sont tels que $!c_i'! \leq B-1$, $\forall i=0,1,\ldots,l$, autrement dit que $c_i' \leq B-1$ si $a+b>0$ et $c_i' \geq -B+1$ si $a+b<0$, $\forall i=0,1,\ldots,l$.
Comme précédemment, nous nous contenterons de la démonstration dans l'hypothèse où $a+b>0$, celle correspondant à l'hypothèse où $a+b<0$ étant symétrique. Pour cela, rappelons qu'avant transformation $c_i=a_i+b_i$ et observons les instructions à l'étape i (*) $(i=0,\ldots,l)$. Nous savons que $!a_i! \leq B-1$ et que $!b_i! \leq B-1$. Comme de plus $a_i * b_i \leq 0$ (puisque a et b sont de signes contraires), nous pouvons écrire que $!a_i+b_i! \leq B-1$. Si $a_i+b_i+REPORT_{(i-1)}<0$, on aura $c_i'=a_i+b_i+REPORT_{(i-1)}+B \leq -1+B=B-1$. (**)
Si $a_i+b_i+REPORT_{(i-1)} \geq 0$, on aura $c_i'=a_i+b_i+REPORT_{(i-1)} \leq a_i+b_i$ puisque $REPORT_{(i-1)}=-1$ ou ϕ. Mais comme dans pareil cas $a_i+b_i=!a_i+b_i! \leq B-1$, on a bien $c_i' \leq B-1$.
Pour conclure, remarquons que les étapes (7) et (8) font que, toujours dans l'hypothèse où $u \neq \phi$, l'algorithme répond bien aussi dans pareil cas à ses spécifications.
Ceci termine la preuve de la correction de ISUM2(a,b,c).

(*) Avec l'éternelle convention $b_i=\phi$ si $i>min(m-1,n-1)$

(**) Par convention, on posera $REPORT_{(-1)}=\phi$

-Remarques, commentaires

Remarquons que les variables u et REPORT doivent être capables de recevoir la valeur -1, ϕ ou 1.(Seule contrainte imposée à ces variables)
Quant à la variable s, elle doit être en mesure de recevoir la valeur -1 ou 1.(Egalement seule contrainte imposée à cette variable)
Enfin, la variable ADAPTATEUR doit être en mesure de recevoir la valeur -B ou B.

En ce qui concerne le temps d'exécution de ISUM2(a,b,c), nous pouvons émettre des remarques semblables à celles dressées relativement à ISUM1(a,b,c).
Nous pouvons donc écrire que $t_{ISUM2(a,b,c)} \hat{=} K.\min(L_B(a),L_B(b))$ où K est une constante de proportionnalité, ceci si l'on peut négliger le temps d'exécution de l'étape (7) qui consiste en une "copie" des composantes restantes de la suite représentant la plus longue entrée.(Respectivement de l'étape (2)).

Nous arrivons tout naturellement(celui-ci étant le successeur logique de celui correspondant à l'addition)à l'algorithme de soustraction, soit IDIF(IDIF pour "integer difference").

c ← IDIF(a,b) ou si l'on veut IDIF(a,b,c)

-Spécifications

a et b sont des nombres entiers. c est un nombre entier.
A la fin de l'exécution de IDIF, c sera égal à la différence de a et b. (*)

-Algorithme

d ← INEG(b) (**)
c ← ISUM(a,d)

-Preuve de la correction de IDIF(a,b,c)

La preuve de la correction de IDIF(a,b,c) est évidente dès que l'on admet que les algorithmes INEG et ISUM répondent bien à leurs spécifications,(voir pour cela ce qui précède) non sans avoir observé que a-b = a+(-b).

-Remarques, commentaires

Conceptuellement parlant, l'algorithme de soustraction présenté ci-dessus est le mieux que l'on puisse espérer. Son extrême simplicité en est la preuve. Mais que l'on ne s'y trompe pas, cet algorithme n'est pas ce que l'on peut espérer de mieux en ce qui concerne les temps d'exécution. En effet, la recherche de l'opposé du nombre entier b demande le "parcours" de la suite $(b_0, b_1, \ldots, b_{n-1})$ représentant b.

(*) De manière plus précise, il y a lieu de comprendre qu'à la fin de l'exécution de IDIF, on connaîtra la représentation de c, différence des nombres entiers a et b dont on connaît les représentations respectives.
(**)d est une variable intermédiaire du même type que celui de b.

Lorsque nous additionnons a avec d, nous effectuons un parcours de la suite $(d_o, d_1, \ldots, d_{n-1})$ représentant d. Il y a là une redondance certaine et qui peut être évitée.
Ainsi, une meilleure solution à ce point de vue consiste à reprendre intégralement l'algorithme d'addition ISUM et à chaque fois qu'il est question dans le texte d'un chiffre b_i de la représentation de b de lui substituer l'élément de texte $-b_i$.
Notons aussi que le gain s'étend au niveau de la taille mémoire, notre dernière solution algorithmique, contrairement à la première, ne nécessitant aucune place mémoire supplémentaire.

Les problèmes de l'addition et de la soustraction de deux nombres entiers étant réglés, nous pouvons maintenant aborder le problème de la multiplication de deux nombres entiers. En guise d'introduction à la solution algorithmique proposée, nous allons traiter un exemple,"manuellement", mettant en évidence les grandes idées, c'est-à-dire les sous-problèmes qu'il y a lieu de résoudre.

Exemple

Considérons les nombres entiers a et b représentés en base 10 respectivement par -2347 et 340070. Nous désirons donc obtenir le produit de a par b.
(a est donc le multiplicande et b le multiplicateur)
Sous forme de suites, on a que
a:--:(-7,-4,-3,-2)
b:--:(0,7,0,0,4,3)
-Le premier chiffre du multiplicateur est 0, par conséquent il suffit d'incrémenter de 1 un compteur de décalage qu'on aura eu soin d'initialiser à la valeur ϕ. Soit CD ce compteur de décalage.(Dès lors CD=1)
-Le deuxième chiffre est 7. Effectuons le produit du multiplicande par 7. On obtient ainsi la suite (-9,-2,-4,-6,-1) représentant le produit de -2347 par 7. (*)
Puisqu'à ce moment le compteur de décalage vaut 1, il suffit d'introduire dans la présente suite un zéro.(**)
Nous obtenons ainsi la suite (0,-9,-2,-4,-6,-1). (1)
Incrémentons le compteur de décalage de 1 en vue de considérer le chiffre suivant du multiplicateur. Dès lors CD=2.
-Le troisième chiffre est 0. Incrémentons le compteur de décalage de 1. Dès lors CD=3.
-Le quatrième chiffre est 0. Incrémentons le compteur de décalage de 1. Dès lors CD=4.

(*)Nous voyons qu'un premier sous-problème consiste à pouvoir multiplier un nombre par un chiffre(B-entier).

(**)Ce compteur de décalage indique en fait qu'il s'agit d'une multiplication par 70 et non par 7.

-Le cinquième chiffre est 4. Effectuons le produit du multiplicande par 4. (*)
On obtient donc la suite (-8,-8,-3,-9) représentant le produit de -2347 par 4.
Le compteur de décalage étant égal à 4, il y a lieu d'introduire dans la suite 4 zéros. Nous obtenons ainsi la suite (0,0,0,0,-8,-8,-3,-9). (2)
Il y a lieu maintenant d'effectuer l'addition des nombres correspondant aux suites (1) de la page 66 et (2) de cette page 67, c'est-à-dire de -164290 et -93880000. (**)
On obtient ainsi la suite (0,-9,-2,-4,-4,0,-4,-9) (3) qui correspond au nombre entier représenté en base 10 par -94044290.
Ajoutons 1 au compteur de décalage en vue de considérer le chiffre suivant du multiplicateur. Dès lors CD=5.
-Le sixième chiffre est 3. Effectuons le produit du multiplicande par 3. On obtient donc la suite (-1,-4,0,-7) représentant le produit de -2347 par 3. Le compteur de décalage étant égal à 5, il y a lieu d'introduire dans la suite 5 zéros. On obtient ainsi la suite (0,0,0,0,0,-1,-4,0,-7).(4)
Il y a lieu maintenant d'effectuer l'addition des nombres correspondant aux suites (3) et (4), c'est-à-dire des nombres représentés en base 10 par -94044290 et -704100000. (**)
On obtient ainsi la suite (0,-9,-2,-4,-4,-1,-8,-9,-7) qui correspond au nombre représenté en base 10 par -798144290 qui n'est rien d'autre que le résultat, c'est-à-dire le produit des nombres entiers représentés en base 10 par -2347 et 340070.

(*) La même remarque qu'à la page 66 tient.

(**) Nous pouvons pour celà utiliser l'algorithme d'addition présenté auparavant. Néanmoins, pour une raison liée au temps d'exécution, nous présenterons une façon plus adéquate, tenant compte de la présence des zéros d'extrême gauche "ajoutés" aux suites. Contrairement à ce qui est présenté dans cet exemple, nous éviterons aussi par la même occasion l'adjonction réelle de ces zéros.

L'exemple introductif met en évidence deux sous-problèmes qu'il suffirait de résoudre pour confectionner l'algorithme de multiplication, à savoir

1)Effectuer le produit d'un nombre entier non nul par un chiffre (B-entier)non nul.

2)Additionner à un nombre entier(éventuellement nul)le produit d'un nombre entier par une puissance entière de B, soit B^n où n est un nombre entier positif ou nul.

En effet, ces deux sous-problèmes résolus, nous obtiendrions l'algorithme IPROD(IPROD pour "integer product") présenté ci-dessous.

c ← IPROD(a,b) ou si l'on veut IPROD(a,b,c).

-Spécifications

a et b sont des nombres entiers. c est un nombre entier.
A la fin de l'exécution de IPROD, c sera égal au produit de a par b. (*)

(*) a est appelé le multiplicande et b le multiplicateur

-Algorithme

(1) si a=ϕ ou b=ϕ alors c $\leftarrow \phi$
(2) sinon (3) CD $\leftarrow \phi$, c $\leftarrow \phi$
(4) pour i allant de 0 à n-1 faire (*)
(5) si $b_i \neq \phi$ alors (a) effectuer le produit de a par b_i et mettre le résultat dans P, c'est-à-dire
P $\leftarrow$ IPRODDI(a,b_i) ($*_1$)
(b) additionner c avec le produit de P par B^{CD} et mettre le résultat dans d, c'est-à-dire
d $\leftarrow$ ISUMPO(c,P,CD) ($*_2$)
(c) effectuer une copie de d dans c, c'est-à-dire
c $\leftarrow$ d
(6) CD $\leftarrow$ CD+1

(*) Pour rappel $b=\sum_{i=0}^{n-1} b_i B^i$ *avec* $b_{n-1} \neq \phi$.

($*_1$) On retrouve ici le premier sous-problème dont question ci-avant(voir page 68). Pour des spécifications précises et l'algorithme relatifs à IPRODDI, on se référera ci-après.

($*_2$) On retrouve ici le deuxième sous-problème dont question ci-avant(voir page 68). Pour des spécifications précises et l'algorithme relatifs à ISUMPO, on se référera ci-après.

-Preuve de la correction de IPROD(a,b,c)

Dans le cas où l'un au moins des deux nombres est nul, la preuve est évidente de par l'instruction à l'étape (1) et le fait que dans pareil cas, le produit est nul.
Il nous reste à fournir la preuve de la correction de IPROD(a,b,c) dans le cas où ni a ni b ne sont nuls.
Montrons qu'à la fin de la kième étape de l'étape (4) ($k=-1,0,1,\ldots,n-1$) $c=(\sum_{i=0}^{k} b_i B^i).a$ et $CD=k+1$, ceci par récurrence sur la valeur de k. (*)
A la fin de l'étape -1, on a que $CD=\emptyset$ et $c=\emptyset$ de par l'étape (3)(étape d'initialisation). Comme par convention $\sum_{i=0}^{-1} b_i B^i=\emptyset$, on a bien que $c=\emptyset=(\sum_{i=0}^{k} b_i B^i).a$. De plus, à la fin de cette même étape -1, on a $CD=k+1$ puisque $CD=\emptyset=-1+1$.
Montrons que si $c_{(k)}=(\sum_{i=0}^{k} b_i B^i).a$ et $CD_{(k)}=k+1$ et supposant (**) que $k<n-1$ (et donc que l'étape k+1 existe), alors $c_{(k+1)}=(\sum_{i=0}^{k+1} b_i B^i).a$ et $CD_{(k+1)}=k+2$.
Par hypothèse $c_{(k)}=(\sum_{i=0}^{k} b_i B^i).a$. Par hypothèse aussi, l'étape k+1 existe. Effectuons symboliquement l'exécution de cette étape k+1. Si $b_{k+1}\neq\emptyset$, de par les instructions de l'étape (5) et de l'étape (6), on a à la fin de l'étape k+1 que $c_{(k+1)}=c_{(k)}+b_{(k+1)}B^{CD(k)}=(\sum_{i=0}^{k} b_i B^i).a+b_{k+1}B^{k+1}=(\sum_{i=0}^{k+1} b_i B^i).a$ et

$CD_{(k+1)}=CD_{(k)}+1=(k+1)+1=k+2$.

Si $b_{k+1}=\emptyset$, l'étape (5) se réduit à néant et on peut de toute façon écrire que $c_{(k+1)}=c_{(k)}+\emptyset=c_{(k)}+b_{(k+1)}B^{CD(k)}=$

$$(\sum_{i=0}^{k} b_i B^i).a+b_{(k+1)}B^{k+1}=\sum_{i=0}^{k+1} b_i B^i \text{ et}$$

$$CD_{(k+1)}=CD_{(k)}+1=(k+1)+1=k+2.$$

Remarquons maintenant qu'à la fin de l'exécution de l'algorithme, on a $c=c_{(n-1)}=(\sum_{i=0}^{n-1} b_i B^i).a=b.a$. L'algorithme est donc bien conforme à ses spécifications.

(*) Nous supposerons comme d'habitude dans pareil cas l'existence d'une étape -1(artificiellement), la fin de l'étape -1 signifiant en fait la fin de l'étape (3) de l'algorithme.
(**)Nous entendons par $c_{(k)}$, la valeur de c à la fin de l'étape k. De même par $CD_{(k)}$, nous entendons le contenu de CD à la fin de l'étape k.

-Remarques, commentaires

La variable CD est un compteur de décalage. Elle doit être capable de recevoir à tout moment un nombre entier positif ou nul. Ce compteur de décalage prend en fait comme valeur maximum la valeur de n où n est défini par la décomposition de b suivant les puissances de B(pour rappel, $b=\sum_{i=0}^{n-1} b_i B^i$, $b_{n-1}\neq\phi$). A moins de tomber dans le gigantisme, frôlant d'ailleurs le ridicule, on peut estimer raisonnablement que cette valeur de n ne dépassera pas la valeur maximum qui peut être contenue dans une variable de type entier relative à la machine et/ou au langage utilisés.

A l'étape (4), nous écrivons l'expression "pour i allant de 0 à n-1". Il n'y a pas lieu de croire qu'il est nécessaire pour ce faire de connaître la valeur de n. Le lecteur perturbé par cette façon de faire lira de manière équivalente l'expression "pour chaque chiffre b_i de la représentation de b, i=0,1,...". Lors de l'implémentation pratique, c qui en fait est assimilable à un accumulateur, ainsi que P, sont représentés par des suites de B-entiers. Nous faisons d'ailleurs ici peut-être trop abstraction de ce fait. Le lecteur se convaincra qu'il n'y a là derrière aucune astuce cachée.

La preuve de la correction de IPROD n'est valable que dans la mesure où nous supposons que les algorithmes IPRODDI et ISUMPO répondent bien à leurs spécifications. Comme promis, nous y reviendrons dans un instant. Remarquons à ce propos que la variable intermédiaire d, de type suite, est nécessaire de par la construction de l'algorithme ISUMPO(pour s'en convaincre, voir cette construction).

En ce qui concerne le temps d'exécution de IPROD(a,b,c), nous en parlerons après analyse des temps d'exécution relatifs à IPRODDI et ISUMPO.

Présentons à présent les algorithmes relatifs aux deux sous-problèmes mis en évidence lors de la mise au point de l'algorithme de multiplication de deux nombres entiers, à savoir IPRODDI et ISUMPO.
Commençons par l'algorithme IPRODDI(IPRODDI pour "integer product by a digit").

c ← IPRODDI(a,di) ou si l'on veut IPRODDI(a,di,c)

-Spécifications

a est un nombre entier non nul. di est un B-entier non nul (et donc 0<!di!<B). c est un nombre entier. A la fin de l'exécution de IPRODDI, c sera égal au produit de a par di. (*)

-Algorithme

(1) REPORT ← ∅
    i ← ∅
(2) jusqu'à ce que la suite correspondant à a soit terminée, faire
        d ← (a_i*di)+REPORT
        c_i ← d mod B (**)
        REPORT ← d div B (***)
        i ← i+1
(3) si REPORT≠∅ alors c_m ← REPORT

(*) De manière plus précise, il y a lieu de comprendre qu'à la fin de l'exécution de IPRODDI, on connaîtra la représentation de c, produit de a par di, étant supposée connue la représentation de a.
(**) Par d mod B, nous entendons le reste de la division entière de d par B. Ainsi, en base B=10, on a :
-82 mod B=-2 puisque -82=10(-8)+(-2)*
*82 mod B=2 puisque 82=10*8+2*
(***) Par d div B, nous entendons le quotient de la division entière de d par B. Ainsi, en base 10, on a :
-82 div B=-8 et 82 div B=8(voir à nouveau les relations ci-dessus)

-Preuve de la correction de IPRODDI(a,b,c)

L'étape (1) de l'algorithme constitue l'étape d'initialisation. L'étape (2) est en fait une boucle comprenant m itérations, m étant défini par la relation $a=\sum_{i=0}^{m-1} a_i B^i$, $a_{m-1}\neq\phi$.

Nous allons montrer par récurrence sur la valeur de k qu'à la fin de la kième itération(k=-1,0,1,...,m-1), la suite $(c_o,c_1,c_2,\ldots,c_k,REPORT_{(k)})$ est telle que (*)

$\sum_{i=0}^{k} c_i B^i+REPORT_{(k)}B^{k+1}=(\sum_{i=0}^{k} a_i B^i)di$ avec les c_i(i=0,1,....,k) et $REPORT_{(k)}$ nuls ou du signe de a*di et de valeurs absolues inférieures à B.(Nous appellerons cet état de fait l'invariant)

A la fin de l'itération -1 l'invariant est vérifié. En effet, la suite $(c_o,c_1,\ldots,c_k,REPORT_{(k)})=(REPORT_{(-1)})=(\phi)$(en fait aucun c_in'existe)($REPORT_{(-1)}=\phi$ de par l'étape (1) d'initialisation)

Supposons maintenant qu'à la fin de l'itération k l'invariant soit vérifié et supposant que l'itération k+1 existe(c'est-à-dire k<m-1), montrons qu'à la fin de celle-ci l'invariant est encore vérifié. Nous supposons donc qu'à la fin de l'itération k, la suite $(c_o,c_1,c_2,\ldots,c_k,REPORT_{(k)})$ est telle que $\sum_{i=0}^{k} c_i B^i+REPORT_{(k)}B^k=(\sum_{i=0}^{k} a_i B^i)di$ avec les c_i(i=0,...,k) et $REPORT_{(k)}$ nuls ou du signe de a*di et de valeurs absolues inférieures à B. L'itération k+1 existe. Exécutons symboliquement celle-ci. A la fin de cette itération, on a

$c_{k+1}=((a_{k+1}*di)+REPORT_{(k)})modB$ et $REPORT_{(k+1)}=$ $((a_{k+1}*di)+REPORT_{(k)})divB$. Remarquons que

$(\sum_{i=0}^{k+1} a_i B^i)di=(\sum_{i=0}^{k} a_i B^i)di+a_{k+1}B^{k+1}di=\sum_{i=0}^{k} c_i B^i+REPORT_{(k)}B^{k+1}+$

$a_{k+1}B^{k+1}di=\sum_{i=0}^{k} c_i B^i+(REPORT_{(k)}+(a_{k+1}di))B^{k+1}$

(*) Par $REPORT_{(k)}$, nous entendons la valeur de REPORT à la fin de la kième itération.

Comme à l'habitude dans pareil cas, nous supposerons artificiellement la présence de l'itération -1, la fin de l'itération -1 signifiant la fin de l'étape (1) de l'algorithme. Et donc $REPORT_{(-1)}=\phi$.

$$= \sum_{i=0}^{k} c_i B^i + (((REPORT_{(k)} + (a_{k+1} di)) divB) B + (REPORT_{(k)} + (a_{k+1} di)) modB) B^{k+1}$$

$$= \sum_{i=0}^{k} c_i B^i + (REPORT_{(k+1)} B + c_{k+1}) B^{k+1} = \sum_{i=0}^{k+1} c_i B^i + REPORT_{(k+1)} B^{k+2}$$

Les $c_o, c_1, \ldots, c_k$ étant inchangés au cours de l'itération k, il suffit pour terminer notre preuve(preuve qu'à la fin de l'itération k+1 l'invariant est toujours vérifié)de montrer que c_{k+1} et $REPORT_{(k+1)}$ sont nuls ou du signe de a*di et de valeurs absolues inférieures à B. Le fait que c_{k+1} soit de valeur absolue inférieure à B est évident puisque $c_{k+1} = ((a_{k+1} di) + REPORT_{(k)}) modB$ et vu les caractéristiques de l'opération mod. Remarquons aussi que

$$|REPORT_{(k+1)}| = |((a_{k+1} di) + REPORT_{(k)}) divB|$$

$$= |(a_{k+1} di) + REPORT_{(k)}| divB \leq ((|a_{k+1}| . |di|) + |REPORT_{(k)}|) divB$$

$$\underset{(*)}{\leq} (((B-1).(B-1)) + (B-1)) divB = (B^2 - B) divB = B-1$$

Rappelons encore que $((a_{k+1} di) + REPORT_{(k)}) modB = c_{k+1}$. Comme $a_{k+1} di$ est nul ou du signe de a*di et que par hypothèse $REPORT_{(k)}$ est nul ou du signe de a*di, il découle immédiatement que c_{k+1} est nul ou du signe de a*di puisque le reste d'une division entière est nul ou du signe du dividende. Enfin, puisque $REPORT_{(k+1)} = ((a_{k+1} di) + REPORT_{(k)}) divB$ et pour les mêmes raisons que ci-dessus ajoutées au fait que B est positif, on déduit que $REPORT_{(k+1)}$ est nul ou du signe de a*di.
L'invariant dont question à la page 74 étant démontré, on peut affirmer maintenant qu'à la fin de l'étape (2) de l'algorithme, qui coïncide avec la fin de l'itération m-1 de cette étape (2), on a que la suite $(c_o, c_1, c_2, \ldots, c_{m-1}, REPORT_{(m-1)})$ est telle que $\sum_{i=0}^{m-1} c_i B^i + REPORT_{(m-1)} B^m = (\sum_{i=0}^{m-1} a_i B^i).di$ avec les $c_i (i=0,1,..,m-1)$ et $REPORT_{(m-1)}$ nuls ou du signe de a*di et de valeurs absolues inférieures à B. Si $REPORT_{(m-1)} \neq \emptyset$, de par l'étape (3) et par ce qui vient d'être énoncé, on a que $(c_o, c_1, \ldots, c_m)$ est bien la représentation de a*di.

(*) a_{k+1} et di sont effectivement des B-entiers.
— $|REPORT_{(k)}| \leq B-1$ de par notre hypothèse.

Si $REPORT_{(m-1)}=\phi$, l'étape (3) se réduisant à néant, aussi par ce qui vient d'être énoncé et à la condition de montrer que dans pareil cas $c_{m-1}\neq\phi$, on a que $(c_o,c_1,\ldots,c_{m-1})$ est <u>la</u> représentation de a*di. Montrons donc pourterminer que $c_{m-1}\neq\phi$ si $REPORT_{(m-1)}=\phi$. Supposons par l'absurde que $c_{m-1}=\phi$. On aurait que $a*di=(\sum_{i=0}^{m-1} a_i B^i)di=\sum_{i=0}^{m-1} c_i B^i=\sum_{i=0}^{m-2} c_i B^i$. Donc on pourrait conclure que $|a*di|<B^{m-1}$, ce qui est absurde puisque $|a*di|=|(\sum_{i=0}^{m-1} a_i B^i)di|\geq B^{m-1}$ puisque $a_{m-1}\neq\phi$ et $di\neq\phi$.

-<u>Remarques, commentaires</u>

La variable REPORT doit être une variable capable de recevoir un B-entier. Ceci est la seule contrainte imposée à cette variable puisque nous avons prouvé ci-dessus que le contenu de REPORT est de valeur absolue inférieure à B.
La variable d quant à elle doit être capable de recevoir des valeurs entières dont la valeur absolue est inférieure ou égale à B(B-1). En effet, la plus grande valeur de d à considérer est celle correspondant à $a_i=B-1$ et di=B-1 et REPORT=B-1 (respectivement à $a_i=-(B-1)$ et di=-(B-1) et REPORT=(B-1)), c'est-à-dire à (B-1)(B-1)+(B-1)=(B-1)B. La plus petite valeur de d à considérer est celle correspondant à $a_i=-(B-1)$ et di=B-1 et REPORT=-(B-1)(respectivement à $a_i=(B-1)$ et di=-(B-1) et REPORT=-(B-1)), c'est-à-dire -(B-1)(B-1)-(B-1)=-(B-1)B.(*)
La raison de la considération de la variable intermédiaire d est que nous voulons éviter un double calcul de (a_i*di)+REPORT (voir étape (2) de l'algorithme).
Enfin signalons que le temps d'exécution $t_{IPRODDI(a,di,c)}$ est tel que $t_{IPRODDI(a,di,c)} \hat{} K.L_B(a)$ où K est une constante de proportionnalité. Ceci découle de manière évidente de l'algorithme IPRODDI.

(*) Cette remarque est importante car elle nous montre que nous sommes limités supérieurement pour le choix de B. Ainsi, si nous travaillons sur une machine de type IBM avec la représentation des nombres entiers dans des mots de 4 bytes selon la technique du complément, on a comme contrainte que $(B-1)B\leq 2^{31}-1$ *et* $-(B-1)B\geq -2^{31}$, *ce qui se résume à* $(B-1)B\leq 2^{31}-1$.

Continuons par l'algorithme ISUMPO(ISUMPO pour "integer sum after a multiplication of the second term by a power of B").

c ← ISUMPO(a,b,e) ou si l'on veut ISUMPO(a,b,e,c)

-Spécifications

a est un nombre entier(qui peut être nul). b est un nombre entier non nul. Si a n'est pas nul, b est du même signe que a. e est un nombre entier positif ou nul. A la fin de l'exécution de ISUMPO, c sera égal à la somme de a avec le produit de b par B élevé à la puissance e, c'est-à-dire $c=a+b.B^e$.

-Algorithme

(1) (a) i ← ϕ

(b) tant que la fin de la suite représentant a n'est pas atteinte et que i≠e faire

$c_i \leftarrow a_i$

i ← i+1

(2) si la fin de la suite représentant a n'est pas atteinte et i=e faire

(a) s1 ← ISIGN(a)

(b) SUITEINT ← ISUM1($(a_e,a_{e+1},\ldots,a_{m-1}),(b_o,b_1,\ldots,b_{n-1})$)

(c) concaténer les suites $(c_o,c_1,\ldots,c_{e-1})$ et SUITEINT donnant ainsi la suite représentant le résultat c

(3) sinon (a) tant que i≠e faire

$c_i \leftarrow \phi$

i ← i+1

(b) tant que la fin de la suite représentant b n'est pas atteinte faire

$c_i \leftarrow b_{i-e}$

i ← i+1

-Preuve de la correction de ISUMP0(a,b,e,c)

Remarquons que l'algorithme répond à ses spécifications dans le cas où a est nul. En effet, le nombre a nul étant représenté par la suite vide, l'étape (1)(a) est exécutée et l'étape (1)(b) se réduit à néant. L'étape (2) se réduit à néant puisque évidemment la fin de la représentation de a est atteinte. Exécutons symboliquement l'étape (3). A la fin de celle-ci, on a que c est représenté par la suite

$$(c_o, c_1, \ldots, c_{e-1}, c_e, c_{e+1}, \ldots, c_{e+n-1}) \qquad (*)$$
$$\phi \quad \phi \qquad \phi \quad b_o \quad b_1 \qquad b_{n-1}$$

qui est bien la représentation de $a+bB^e$ puisque $a+bB^e=bB^e$

$$=\left(\sum_{i=0}^{n-1} b_i B^i\right)B^e=\sum_{i=0}^{n-1} b_i B^{i+e}=\sum_{i=0}^{e-1} c_i B^i+\sum_{i=e}^{e+n-1} c_i B^i=\sum_{i=0}^{e+n-1} c_i B^i \qquad (**)$$

et vu les contraintes initiales sur les b_i qui restent toujours d'actualité. (***)

Venons-en maintenant à la preuve de la correction dans le cas où ni a ni b ne sont nuls. a et b sont donc non nuls et de mêmes signes. Il est évident de voir qu'à la fin de l'étape (1), $i=\min(m,e)$ et $(c_o, c_1, \ldots, c_{i-1})=(a_o, a_1, \ldots, a_{i-1})$ et donc (****) $\sum_{j=0}^{i-1} c_j B^j=\sum_{j=0}^{i-1} a_j B^j$. A la fin de l'étape (1), 3 situations sont possibles. Ou bien la fin de la suite représentant a n'est pas atteinte et i=e ou bien cette fin est atteinte et $i\neq e$ ou bien encore cette fin est atteinte et i=e(ceci découle de la condition de terminaison de l'étape (1)).

(*) La suite $(c_o, c_1, \ldots, c_{e-1})$ est la suite vide si $e=\phi$.

(**) Avec $c_i=\phi$ pour $i=0,1,\ldots,e-1$ et $c_i=b_{i-e}$ pour $i=e,\ldots,e+n-1$.

(***) Remarquons d'ailleurs que vu les spécifications, $b_{n-1}=c_{e+n-1}\neq\phi$ puisque $b\neq\phi$.

(****) m est défini par $a=\sum_{i=0}^{m-1} a_i B^i, a_{m-1}\neq\phi$.
Les suites $(c_o, c_1, \ldots, c_{e-1})$ et $(a_o, a_1, \ldots, a_{e-1})$ sont vides si $e=\phi$.

Envisageons dans un premier temps la première situation, c'est-à-dire i<m et i=e et fournissons la preuve de la correction dans pareil cas. Exécutons symboliquement l'étape (2). Remarquons que puisque e=i<m, la suite $(a_e,a_{e+1},\ldots,a_{m-1})$ n'est pas vide. Remarquons aussi que la suite $(b_o,b_1,\ldots,b_{n-1})$ n'est pas vide puisque $b\neq\phi$. De plus $a_{m-1}\neq\phi$ et $b_{n-1}\neq\phi$. De par l'instruction (2)(a), s1 comprend bien le signe de a. Nous sommes donc dans les hypothèses pour pouvoir appliquer l'algorithme ISUM1. De par les spécifications relatives à ISUM1, nous sommes certains qu'à la fin de l'exécution de l'étape (2)(b), la suite SUITEINT représentera bien la somme des nombres entiers représentés respectivement par $(a_e,a_{e+1},\ldots,a_{m-1})$ et $(b_o,b_1,\ldots,b_{n-1})$. Appelons l la B-longueur de SUITEINT. On a donc $\sum_{i=0}^{l-1}SUITEINT_iB^i=\sum_{i=0}^{m-1-e}a_{e+i}B^i+\sum_{i=0}^{n-1}b_iB^i$.

A la fin de l'étape (2)(c) de concaténation, on a donc que c est représenté par $(c_o,c_1,\ldots,c_{e-1},c_e,c_{e+1},\ldots,c_{e+l-1})$ où $c_{e+i}=SUITEINT_i(i=0,\ldots,l-1)$.

Nous pouvons donc écrire que $\sum_{i=0}^{e+l-1}c_iB^i=(\sum_{i=0}^{e-1}c_iB^i)+(\sum_{i=e}^{e+l-1}c_iB^i)$

$=\sum_{i=0}^{e-1}a_iB^i+(\sum_{i=e}^{e+l-1}c_iB^{i-e})B^e=\sum_{i=0}^{e-1}a_iB^i+(\sum_{i=e}^{e+l-1}SUITEINT_{i-e}B^{i-e})B^e$

$=\sum_{i=0}^{e-1}a_iB^i+(\sum_{i=0}^{l-1}SUITEINT_i.B^i)B^e=\sum_{i=0}^{e-1}a_iB^i+(\sum_{i=0}^{m-1-e}a_{e+i}B^i+\sum_{i=0}^{n-1}b_iB^i)B^e$

$=\sum_{i=0}^{e-1}a_iB^i+\sum_{i=0}^{m-1-e}a_{e+i}B^{e+i}+(\sum_{i=0}^{n-1}b_iB^i)B^e=\sum_{i=0}^{m-1}a_iB^i+(\sum_{i=0}^{n-1}b_iB^i)B^e$

$=a+bB^e$. Il est de plus évident que chaque $c_i(i=0,\ldots,e+l-1)$ est nul ou du signe de a, de valeur absolue inférieure à B et que $c_{e+l-1}\neq\phi$. (*)

La suite $(c_o,c_1,\ldots,c_e,c_{e+1},\ldots,c_{e+l-1})$ est donc bien la représentation de $a+bB^e$.

(*) De par les contraintes sur les $a_i(i=0,\ldots,e-1)$ et de par les spécifications relatives à ISUM1.

Il nous reste à fournir la preuve de la correction dans le cas des deux dernières situations, c'est-à-dire, pour rappel, si à la fin de l'étape (1), la fin de la suite représentant a est atteinte et i=e ou si cette fin est atteinte et i≠e(et donc i<e). Il est donné suite à ces deux situations à l'étape (3).
En effet, la fin de la représentation de a étant atteinte, nous pouvons certifier que i=m. Donc, à la fin de l'exécution de l'étape (1), on a $\sum_{i=0}^{m-1} c_i B^i = \sum_{i=0}^{m-1} a_i B^i$. Exécutons symboliquement l'étape (3). A la fin de celle-ci, on a i=e+n (*)
et $(c_0,c_1,\ldots,c_{m-1},c_m,c_{m+1},\ldots,c_{e-1},c_e,c_{e+1},\ldots,c_{e+n-1})=$ (**)

$(c_0,c_1,\ldots,c_{m-1},\underline{\phi,\phi,\ldots,\phi},b_0,b_1,\ldots,b_{n-1})=$

$(a_0,a_1,\ldots,a_{m-1},\underline{\phi,\phi,\ldots,\phi},b_0,b_1,\ldots,b_{n-1})$

e-m zéros si m<e et 0 zéro sinon(c'est-à-dire si m=e)

et donc $\sum_{i=0}^{e+n-1} c_i B^i = \sum_{i=0}^{m-1} a_i B^i + \sum_{i=0}^{n-1} b_i B^{i+e} = \sum_{i=0}^{m-1} a_i B^i + (\sum_{i=0}^{n-1} b_i B^i) B^e$

$=a+bB^e$. Comme de plus chaque c_i (i=0,...,e+n-1) est nul ou du signe de a, de valeur absolue inférieure à B et que $c_{e+n-1} = b_{n-1} \neq \phi$, on a bien que la suite $(c_0,c_1,\ldots,c_{m-1},c_m,c_{m+1},\ldots,c_{e-1},c_e,c_{e+1},\ldots,c_{e+n-1})$ est la représentation de $a+bB^e$.

Enfin remarquons que la démonstration particulière présentée au tout début pour prouver la correction de ISUMPO dans le cas où a est nul est superflue car la démonstration que nous venons de terminer convient aussi pour le cas où a est nul, à condition de poser dans pareil cas m=ϕ et de prendre comme convention que $(c_0,c_1,\ldots,c_{m-1})$ et $(a_0,a_1,\ldots,a_{m-1})$ sont vides.

(*) n est la B-longueur de b.
(**) La sous-suite $(c_m,c_{m+1},\ldots,c_{e-1})$ est vide si m≥e, c'est-à-dire si m=e puisque à la fin de l'étape (1), on a m=i≤e.

-Remarques, commentaires

Nos remarques porteront essentiellement sur le temps d'exécution de ISUMPO(a,b,e,c) et sur les conséquences des temps d'exécution de IPRODDI(a,di,c) et ISUMPO(a,b,e,c) sur celui de IPROD(a,b,c).

Remarquons que mise à part l'incrémentation de i, l'étape (1) se réduit à une instruction répétée de type "copie"("$c_i \leftarrow a_i$"). Le même genre de remarque tient pour l'étape (3). Remarquons aussi que les étapes (2) et (3) sont mutuellement exclusives. Par conséquent, si l'étape (2) se réduit à néant(ceci arrivera d'autant plus vite que $L_B(a)$ est petite et que l'exposant e est grand), l'exécution de l'algorithme se réduit à un parcours des suites et à des instructions de type "copie", aucune instruction de type "calcul" n'étant présente. Nous gagnons donc là inévitablement au niveau du temps d'exécution. Par contre, si l'étape (2)(et non l'étape (3))est exécutée(ceci arrive d'autant plus que $L_B(a)$ est grande et que l'exposant e est petit), il peut apparaître une part importante de temps d'exécution de type "calcul".

Faisons exceptionnellement un retour en arrière. Nous avions en effet promis de reparler du temps d'exécution de l'algorithme IPROD. Pour cela, regardons une nouvelle fois l'algorithme IPROD présenté page 69 et plus particulièrement la boucle portant sur les étapes (5) et (6) et que nous avons baptisée étape (4), seule partie intéressante à considérer pour le temps d'exécution. De par la remarque qui précède relative au temps d'exécution de ISUMPO, nous pouvons croire que le temps nécessaire à l'étape (5)(b)(addition de c avec le produit de P par B^{CD} et mise du résultat dans d)diminuera au fur et à mesure que nous avançons dans la suite représentant b(puisque le contenu de CD, compteur de décalage augmente)malgré l'augmentation de $L_B(c)$, c jouant le rôle d'accumulateur. Si ce temps restait constant, nous pourrions écrire que

$t_{IPROD(a,b,c)} \hat{=} K.L_B(a).L_B(b)$ si nous supposons tous les b_i non nuls puisque $t_{IPRODDI(a,b_i,P)} \hat{=} K'L_B(a)$ et que l'on pourrait (*)

(*) K' est une constante de proportionnalité.

écrire que $t_{ISUMPO(c,P,CD,d)} \hat{=} K'' \min(L_B(c),L_B(P)) \underset{(*)}{=} K'' L_B(P)$.

$\underset{(**)}{=} K''' L_B(a)$.

Nous pouvons donc écrire, vu les remarques qui précèdent, renforcées encore par le fait que certains chiffres b_i de la représentation de b sont nuls, que (***)

$t_{IPROD(a,b,c)} \hat{=} f(L_B(a),L_B(b)) \leq K.L_B(a).L_B(b)$, K étant une constante de proportionnalité. (****)

(*) Le fait que $L_B(P) \leq L_B(c)$ découle du fait que c joue le rôle d'accumulateur.

(**) En fait, P représentant le produit de a par b_i, B-entier, $L_B(P)=L_B(a)$ ou $L_B(P)=L_B(a)+1$.

(***) L'influence favorable des b_i nuls est d'autant plus grande que B est petite.

(****) Par $f(L_B(a),L_B(b))$, il y a lieu de lire "une fonction de $L_B(a)$ et $L_B(b)$".

Nous allons à présent traiter le problème de la division d'un nombre entier par un autre nombre entier, supposant que ce dernier est non nul.
Contrairement à ce qui a été fait pour les autres problèmes, nous ne présenterons pas directement un algorithme de solution en fournissant ultérieurement une preuve de sa correction, mais nous mettrons d'abord en évidence toute une série de problèmes à résoudre, de résultats particuliers acquis et de remarques intéressantes qui amènent à la conception et à la justification d'un algorithme de solution. (Comme nous allons bien vite le comprendre, présenter directement l'algorithme proposé s'avérerait antipédagogique, voire suicidaire)
Bien évidemment, il y a lieu de présenter, comme à l'habitude avant toute autre chose, les spécifications précises du problème à résoudre (c'est-à-dire de l'algorithme de division), rappelant par la même occasion les caractéristiques précises d'une division d'un nombre entier par un autre nombre entier non nul.
L'algorithme de division sera baptisé IQR (IQR pour "integer quotient and remainder").

(q,r) ← IQR(a,b) ou si l'on veut IQR(a,b,q,r)

-Spécifications

a est un nombre entier. b est un nombre entier non nul.
q et r sont des nombres entiers. A la fin de l'exécution de IQR, q sera égal au quotient et r au reste de la division de a par b. (A titre de rappel, nous trouverons ci-dessous la définition du quotient et du reste de la division du nombre entier a par le nombre entier b non nul)

Rappel Soient $a,b \in \mathbb{Z}, b \neq 0$. Le quotient et le reste r de la division de a par b sont les uniques nombres entiers q et r définis par les relations

(1) $a = b.q + r$
(2) $0 \leq r < |b|$ si $a \geq 0$
(3) $-|b| < r \leq 0$ si $a < 0$

Avant de poursuivre, montrons que la définition du quotient et du reste donnée dans le rappel a bien un sens, entendons par là qu'étant donnés a et b deux nombres entiers, b étant non nul, l'existence et l'unicité de q et r satisfaisant aux contraintes (1,(2) et (3) ne font aucun doute.
Commençons par l'unicité. Supposons dans un premier temps $a \geq 0$. Supposons donc l'existence de q et r, nombres entiers tels que $a=b.q+r$ et $0 \leq r < |b|$ et l'existence de q' et r', nombres entiers tels que $a=b.q'+r'$ et $0 \leq r' < |b|$. Montrons que $q=q'$ et $r=r'$. Pour cela, observons que $0=a-a=(q-q')b+(r-r')$ et que $|r-r'| < |b|$ (puisque $0 \leq r < |b|$ et $-|b| < -r' \leq 0$ impliquent $-|b| < r-r' < |b|$). $(q-q')b+r-r'=0$, dès lors $(q-q')b=r'-r$ et donc $|q-q'||b|=|r-r'|$. Si on suppose $q \neq q'$, on aboutit à une contradiction puisqu'on peut en déduire $\underbrace{|q-q'|}_{\geq 1}|b|=|r-r'| \geq |b|$.
Dès lors, on peut certifier que $q=q'$. Mais $(q-q')b+r-r'=0$, dès lors on en déduit que $r=r'$.
Un raisonnement parfaitement symétrique peut être fourni dans le cas où $a<0$.
Poursuivons par l'existence. Remarquons tout d'abord que si $a=0$, alors $q=0$ et $r=0$ puisque $0=0.b+0$ (on a bien $0 \leq 0 < |b|$). L'existence est bien prouvée dans ce cas. Si $a>0$ et $b>0$, nous pouvons écrire que $a=\lfloor a/b \rfloor .b+(a-\lfloor a/b \rfloor .b)$. Il suffit donc de (*) poser $q=\lfloor a/b \rfloor$ et $r=a-\lfloor a/b \rfloor .b$ pour prouver l'existence dans pareil cas puisque $0 \leq a-\lfloor a/b \rfloor .b < b=|b|$.
Si $a>0$ et $b<0$, la preuve de l'existence se déduit du cas précédent. En effet. Soient q et r le quotient et le reste de la division de a par -b. On a donc $a=q.(-b)+r$ avec $0 \leq r < |-b|=|b|$. On a donc $a=(-q).b+r$ avec $0 \leq r < |b|$. En posant $q'=-q$ et $r'=r$, ceci prouve l'existence du quotient et du reste dans le cas où $a>0$ et $b<0$.

(*) $\lfloor a/b \rfloor$ désigne le plus petit nombre entier inférieur ou égal à a/b.

Si $a<0$ et $b>0$, la preuve de l'existence se déduit d'une manière analogue. En effet. Soient q et r le quotient et le reste de la division de -a par b. On a donc $(-a)=q.b+r$ avec $0\leq r<|b|$. On a donc $a=(-q).b+(-r)$ avec $-|b|<-r\leq 0$. En posant $q'=-q$ et $r'=-r$, ceci prouve l'existence du quotient et du reste dans le cas où $a<0$ et $b>0$.

Enfin, si $a<0$ et $b<0$, effectuons un raisonnement analogue. Soient q et r le quotient et le reste de la division de -a par -b. On a donc $-a=q.(-b)+r$ avec $0\leq r<|-b|=|b|$. On a donc $a=q.b+(-r)$ avec $-|b|<-r\leq 0$. En posant $q'=q$ et $r'=-r$, ceci prouve l'existence du quotient et du reste dans le cas où $a<0$ et $b<0$.

Nous venons donc de prouver l'existence et l'unicité du quotient et du reste de la division d'un nombre entier a par un nombre entier b non nul. En fait, nous venons de prouver bien plus que cela. Nous venons en plus de prouver que le problème de la recherche du quotient et du reste de la division d'un nombre entier a par un nombre entier b non nul se réduit au problème de la recherche du quotient et du reste d'un nombre a positif par un nombre entier b positif. (*)

Nous mettrons cette constatation à profit lors de l'élaboration de l'algorithme IQR.

En fait, le problème de la recherche du quotient et du reste de la division d'un nombre entier a positif par un nombre entier b positif se réduit lui-même au problème de la recherche du quotient et du reste de la division d'un nombre entier a positif par un nombre entier b positif tels que $L_B(a)\geq L_B(b)$ puisque si $L_B(a)<L_B(b)$, alors $a<b$ et donc $a=0.b+a$ (Il suffit donc de poser $q=\emptyset$ et $r=a$).

(*) Par l'expression "réduction du problème 2 au problème 1", il y a lieu de comprendre que si nous disposons d'un algorithme permettant de résoudre le problème 1, il suffit de l'adapter trivialement pour en obtenir un résolvant le problème 2.

Nous allons présenter maintenant une propriété qui explique le comportement du quotient et du reste d'une division entière d'un nombre entier positif a par un nombre positif b lorsque simultanément le dividende a et le diviseur b sont multipliés par un nombre positif d.

Propriété 1

Soient $a,b,d \in \mathbb{N}_o$. Soient q et r le quotient et le reste de la division de a par b. Soient $\bar{a}=a.d$ et $\bar{b}=b.d$. Soient $\bar{q}$ et $\bar{r}$ le quotient et le reste de la division de $\bar{a}$ par $\bar{b}$. On a $\bar{q}=q$ et $\bar{r}=r.d$.

Preuve

Par hypothèse, on a que $a=q.b+r$ avec $0 \leq r < b$.

Dès lors $\bar{a}=a.d=q.b.d+r.d=q.\bar{b}+r.d$.

Observons que $0 \leq r.d < b.d=\bar{b}$.

On en conclut que $\bar{q}=q$ et $\bar{r}=r.d$.

Un peu dans le même ordre d'idée, nous allons définir ce que nous entendons par la normalisation des nombres entiers positifs a et b et énoncer une propriété qui y est relative. (*)

Définition

Soient a et b deux nombres entiers positifs. Soient aussi $(a_o,a_1,\ldots,a_{m-2},a_{m-1})$ et $(b_o,b_1,\ldots,b_{n-2},b_{n-1})$ les suites représentant ces deux nombres. $(m=L_B(a), n=L_B(b))$

Normaliser a et b signifie multiplier a et b par $d=\lfloor B/(b_{n-1}+1) \rfloor$ (**) donnant ainsi a' et b'. ((a',b') est appelé la normalisation de (a,b), d est le facteur de normalisation)

Propriété 2

Supposons adoptées les mêmes notations que dans la définition ci-dessus. On a que $L_B(a')=m+1$ ou $L_B(a')=m$ et $L_B(b')=n$ (c'est-à-dire a' est représenté par la suite $(a'_o,a'_1,\ldots,a'_{m-1},a'_m)$ ou $(a'_o,a'_1,\ldots,a'_{m-1})$ et b' est représenté par la suite $(b'_o,b'_1,\ldots,b'_{n-1})$. De plus $b'_{n-1} \geq \lfloor B/2 \rfloor$. Autrement dit $\lfloor B/2 \rfloor B^{n-1} \leq b' < B^n$.

(*) Nous verrons que le concept de normalisation joue un rôle important dans la conception de l'algorithme IQR.

(**) $\lfloor x \rfloor$ signifie "le plus grand entier inférieur ou égal à x".

Preuve

La première partie de la propriété, celle concernant a' est presque évidente. En effet. $a=\sum_{i=0}^{m-1} a_i B^i$ $(a_{m-1}\neq\emptyset)$ et donc $B^{m-1}\leq a<B^m$. De plus $1\leq d<B$ puisque $d=\lfloor B/(b_{n-1}+1)\rfloor$ et que $1\leq b_{n-1}<B$. Par conséquent

$B^{m-1}\leq a.d=a'<B^{m+1}$, ce qui signifie que la représentation de a' nécessite au plus m+1 chiffres mais au moins m.

Venons-en à la deuxième partie, c'est-à-dire à celle concernant b'. Montrons donc que $\lfloor B/2\rfloor B^{n-1}\leq b'<B^n$.
(b) (a)

(a) $b=b_{n-1}B^{n-1}+b_{n-2}B^{n-2}+\ldots+b_1B+b_o<(b_{n-1}+1)B^{n-1}$.

$<B^{n-1}$

Donc $b'=d.b=\lfloor B/(b_{n-1}+1)\rfloor . b\leq \lfloor B/(b_{n-1}+1)\rfloor .(b_{n-1}+1).B^{n-1}$

$\leq (B/(b_{n-1}+1)).(b_{n-1}+1).B^{n-1}$

$= B.B^{n-1}=B^n$

(b) Montrons donc que $\lfloor B/2\rfloor .B^{n-1}\leq b'$, c'est-à-dire que

$$\lfloor B/2\rfloor .B^{n-1}\leq \lfloor B/(b_{n-1}+1)\rfloor .(b_{n-1}B^{n-1}+b_{n-2}B^{n-2}+\ldots+b_1B+b_o)$$

Remarquons que puisque $b_{n-2}B^{n-1}+b_{n-2}B^{n-2}+\ldots+b_1B+b_o\geq 0$, il suffirait de montrer que
$\lfloor B/2\rfloor .B^{n-1}\leq \lfloor B/(b_{n-1}+1)\rfloor .b_{n-1}B^{n-1}$, c'est-à-dire que

$\lfloor B/2\rfloor \leq \lfloor B/(b_{n-1}+1)\rfloor b_{n-1}$. Or cette inégalité est vraie. Nous allons le prouver en deux étapes, à savoir si $b_{n-1}\geq\lfloor B/2\rfloor$ puis si $1\leq b_{n-1}<\lfloor B/2\rfloor$.

(1) Si $b_{n-1}\geq\lfloor B/2\rfloor$

$b_{n-1}\geq\lfloor B/2\rfloor$. Par conséquent $b_{n-1}+1>B/2$. (*)

Dès lors $B/(b_{n-1}+1)<B/(B/2)$, c'est-à-dire $b/(b_{n-1}+1)<2$.

(*) En effet. Raisonnons pour cela par contraposée.

$b_{n-1}+1\leq B/2 \Rightarrow b_{n-1}+1\leq\lfloor B/2\rfloor \Rightarrow b_{n-1}<\lfloor B/2\rfloor$

car $b_{n-1}+1$ est entier

Mais puisque $1 \leq B/(b_{n-1}+1)$, on peut en déduire que $\lfloor B/(b_{n-1}+1) \rfloor = 1$. (*)

Dès lors $\lfloor B/(b_{n-1}+1) \rfloor \cdot b_{n-1} = b_{n-1} \geq \lfloor B/2 \rfloor$, par hypothèse.

(2) Si $1 \leq b_{n-1} < \lfloor B/2 \rfloor$

Remarquons que $b_{n-1} \cdot \lfloor B/(b_{n-1}+1) \rfloor > b_{n-1}((B/(b_{n-1}+1))-1)$ (1)

puisque $b_{n-1} > 0$. Nous pouvons montrer de plus que $b_{n-1}((B/(b_{n-1}+1))-1) \geq (B/2)-1$ (2). En effet

$b_{n-1}((B/(b_{n-1}+1))-1)-(B/2)+1 = (B/2-b_{n-1}-1)(b_{n-1}-1)/(b_{n-1}+1) \geq 0$. (**) Pour s'en convaincre, observons que $b_{n-1} < \lfloor B/2 \rfloor$ et donc que $b_{n-1}+1 \leq \lfloor B/2 \rfloor \leq B/2$, ce qui implique que $B/2-b_{n-1}-1 \geq 0$. De plus $b_{n-1}-1 \geq 0$ et $b_{n-1}+1 \geq 0$ puisque par hypothèse $b_{n-1} \geq 1$.

Puisque de toute façon $(B/2)-1 \geq \lfloor B/2 \rfloor - 1$, nous pouvons déduire de la relation (2) que $b_{n-1}((B/(b_{n-1}+1))-1) \geq \lfloor B/2 \rfloor - 1$. En combinant ceci avec la relation (1), on obtient $b_{n-1} \lfloor B/(b_{n-1}+1) \rfloor > \lfloor B/2 \rfloor - 1$ et donc $b_{n-1} \lfloor B/(b_{n-1}+1) \rfloor \geq \lfloor B/2 \rfloor$, ce qu'on voulait démontrer.

(*) car $B \geq b_{n-1}+1$ *car* $b_{n-1} < B$

(**) $b_{n-1}((B/(b_{n-1}+1))-1)-(B/2)+1 = \dfrac{b_{n-1}(B-(b_{n-1}+1))-((B/2)-1)(b_{n-1}+1)}{b_{n-1}+1}$

$$= \frac{b_{n-1}(B/2-b_{n-1}-1)+B/2 \cdot b_{n-1}-B/2 \cdot b_{n-1}-B/2+b_{n-1}+1}{b_{n-1}+1}$$

$$= \frac{b_{n-1}(B/2-b_{n-1}-1)-(B/2-b_{n-1}-1)}{b_{n-1}+1}$$

$$= \frac{(B/2-b_{n-1}-1)(b_{n-1}-1)}{b_{n-1}+1}$$

Poursuivons en émettant les remarques suivantes.
-Le calcul du facteur de normalisation $d=\lfloor B/(b_{n-1}+1)\rfloor$ ne pose aucun problème sur les machines habituelles. En effet. b_{n-1} étant un B-entier non nul, on a que $2\leq b_{n-1}+1\leq B$. Le calcul de d se ramène à un calcul du plus grand entier inférieur ou égal au quotient de B par un nombre compris entre 2 et B, ce qui ne fait intervenir que des opérations classiques sur toute machine.(*)
-L'opération de normalisation des entiers positifs a et b ne pose aucun problème non plus. En effet, nous avons mis en évidence lors de la preuve de la propriété 2 que $1\leq d=\lfloor B/(b_{n-1}+1)\rfloor < B$, c'est-à-dire que $d=\lfloor B/(b_{n-1}+1)\rfloor$ est un B-entier. Or nous disposons déjà d'un algorithme(voir IPRODDI page 72) permettant de multiplier un nombre entier non nul par un B-entier(non nul).
-Si nous parvenions à déterminer le quotient q' et le reste r' de la division de a'=a.d par b'=b.d ($d=\lfloor B/(b_{n-1}+1)\rfloor$), nous pourrions à peu de frais en déduire le quotient q et le reste r de la division de a par b, puisque la propriété 1(page 85) nous signale que q=q' et vu que r=a-q.b.(Il est préférable, indispensable même de calculer r par la formule r=a-q.b plutôt que par la formule r=r'/d car cette dernière fait intervenir une division (pour laquelle nous ne disposons pas encore d'un algorithme puisque nous cherchons justement à en établir un)alors que la première ne fait intervenir qu'une multiplication et une soustraction pour lesquelles nous disposons déjà d'un algorithme) (**)

(*) Nous mettons ici en évidence <u>une</u> contrainte relative à la valeur de B. Celle-ci doit être inférieure ou égale à la plus grande valeur entière représentable dans la machine. Ainsi, sur une machine de type IBM avec les entiers représentables dans des mots de 4 bytes, selon la technique du complément, la contrainte s'écrirait $B\leq 2^{31}-1$. En fait, cette contrainte n'en n'est plus une, puisqu'on a déjà mis en évidence des contraintes plus draconiennes concernant le choix de B. Voir (*) page 75)
(**) Il peut sembler à priori étrange pour calculer le quotient et le reste de la division du nombre entier positif a par le nombre positif b de passer par l'intermédiaire du calcul du quotient et du reste de la division du nombre entier a' par le nombre entier b'(obtenus par normalisation). Il n'en est rien. Nous montrerons en effet que la détermination des chiffres du quotient q est rendue "aisée" par l'opération de normalisation.

Nous fixerons donc maintenant toute notre attention sur le dernier "gros" problème à résoudre en vue de l'obtention d'un algorithme de division, à savoir celui de la détermination des chiffres du quotient de la division de a' par b', a' et b' étant obtenus par normalisation de a et b(on suppose a et b positifs et tels que $L_B(a) \geq L_B(b)$, vu la remarque de la page 84). Nous comprendrons ici, enfin, pourquoi l'opération de normalisation est fondamentale.
Nous basant sur les caractéristiques des représentations de a et b(et par voie de conséquence sur celles de a' et b'), mettons en évidence quelques caractéristiques de la représentation de q(quotient de la division de a par b, c'est-à-dire de celle de a' par b').

$a :--: (a_0, a_1, \ldots, a_{m-1})$, $a_{m-1} > 0$.

$b :--: (b_0, b_1, \ldots, b_{n-1})$, $b_{n-1} > 0$.

Nous supposons de plus $m \geq n$, c'est-à-dire $L_B(a) \geq L_B(b)$.

$a' :--: (a'_0, a'_1, \ldots, a'_{m-1}, a'_m)$ ou $a' :--: (a'_0, a'_1, \ldots, a'_{m-1})$ si $a'_m = \emptyset$.

$b' :--: (b'_0, b'_1, \ldots, b'_{n-1})$.

Nous savons donc que $B^{m-1} \leq a < B^m$ et $B^{n-1} \leq b < B^n$.
Appelons a/b le quotient de a par b au sens réel.
$a < B^m$ et $B^{n-1} \leq b$. Dès lors $a/b < B^m / B^{n-1} = B^{m-n+1}$ (1)

$B^{m-1} \leq a$ et $b < B^n$. Dès lors $B^{m-n-1} = B^{m-1}/B^n < a/b$ (2)

Combinant les relations (1) et (2), on obtient

$$B^{-1} \leq B^{m-n-1} < a/b < B^{m-n+1} \quad (3)$$

Comme nous savons que $q = \lfloor a/b \rfloor$ et de par la relation (3), nous pouvons certifier que q peut s'écrire sous la forme

$$q = \sum_{j=0}^{m-n} q_j B^j \text{ avec } q_j \geq 0,\ j = 0, 1, \ldots, m-n \quad (4)$$

(Et par conséquent, la représentation de q comportera au plus m-n+1 chiffres. Remarquons d'ailleurs que cette représentation peut être la suite vide puisque q peut être nul. Ceci arrive si $a < b$, ce qui n'est pas en contradiction avec le fait que $L_B(a) \geq L_B(b)$. Remarquons aussi que si $m-n \geq 1$ et que $q_{m-n} = \emptyset$, alors $q_{m-n-1} \neq \emptyset$ puisqu'alors B^{m-n-1} est un nombre entier et donc que $B^{m-n-1} \leq \lfloor a/b \rfloor$, ceci découlant de la relation (3))

Le reste du travail pour arriver à "notre" algorithme de division consiste à mettre en évidence une technique pour déterminer successivement(et rapidement si possible)les chiffres q_{m-n-j}, $j=0,1,\ldots,m-n$ correspondant à la décomposition de q sous la forme $\sum_{j=0}^{m-n} q_{m-n-j} B^{m-n-j}$ (c'est-à-dire du plus significatif au moins significatif), nous permettant ainsi d'obtenir la représentation sous forme de suite de q.
La technique adoptée et que nous appellerons technique de "division-correction" consiste à trouver une approximation (*) $\bar{q}_{m-n-j}$ de q_{m-n-j} et de laquelle nous déduirons la valeur q_{m-n-j}, ceci pour chaque j allant de 0 à m-n. (**)
En vue de présenter cette technique, considérons à nouveau les nombres a' et b' considérés avant, obtenus par normalisation de a et b, a et b étant deux nombres positifs et tels que $L_B(a) \geq L_B(b)$. Nous avons mis en évidence que la représentation de a' est $(a'_o,a'_1,\ldots,a'_{m-1},a'_m)$ ou $(a'_o,a'_1,\ldots,a'_{m-1})$ suivant que la normalisation a nécessité ou non un chiffre supplémentaire, et que la représentation de b' est $(b'_o,b'_1,\ldots,b'_{n-1})$ $(b'_{n-1} \geq \lfloor B/2 \rfloor)$.
De manière à ne traiter qu'une seule forme pour la représentation de a', nous dirons que a' est représenté par $(a'_o,a'_1,\ldots,a'_{m-1},a'_m)$, acceptant exceptionnellement que a'_m puisse être nul. (***)
(mais en imposant que si tel est le cas, alors a'_{m-1} existe et est non nul). Nous considérons donc une représentation de a' à m+1 chiffres. Les chiffres du quotient q de a par b(c'est-à-dire de a' par b')vont être obtenus via les représentations de a' et b' de la façon suivante.

(*) Nous verrons ultérieurement la raison de cette appellation.
(**)Comme nous le verrons, $\bar{q}_{m-n-j}$ est un nombre entier et ne différera guère(parfois même pas du tout) de q_{m-n-j}. C'est d'ailleurs ce qui fera tout l'intérêt de cette méthode de "division-correction".
(***) Nous ne pouvons plus parler de la représentation au sens strict de a' puisque nous acceptons que a'_m soit nul.

Supposons vouloir déterminer q_{m-n-j}, supposant avoir déterminé déjà les chiffres plus significatifs précédents du quotient, c'est-à-dire $q_{m-n}, q_{m-n-1}, \ldots, q_{m-n-j+1}$. (*)
Supposons avoir déterminé déjà aussi le reste partiel $\mathcal{S}(m-n-j)=a'-b'\sum_{i=m-n-j+1}^{m-n} q_i B^i$ (nous supposons j tel que (*)

$0 \leq j \leq m-n$), $\mathcal{S}(m-n-j)$ satisfaisant à l'inégalité

$0 \leq \mathcal{S}(m-n-j) < b' B^{m-n-j+1} < B^{m-j+1}$ (1) (**)

Remarquons que $B^{m-j+1} \leq B^{m+1}$ puisque $B^{-j} \leq 1$ $(j=0,1,\ldots,m-n)$.
Cette remarque est importante car elle met en évidence que le nombre entier $\mathcal{S}(m-n-j)$ peut être représenté au moyen de m+1 B-entiers au plus. Considérons d'ailleurs la représentation sur une B-longueur de m+1 chiffres de ce reste partiel $\mathcal{S}(m-n-j)$. (Par représentation sur une B-longueur de m+1 chiffres de ce reste partiel $\mathcal{S}(m-n-j)$, nous entendons la représentation au sens habituel concaténée par la droite avec une suite(éventuellement vide)de B-entiers nuls jusqu'à obtenir un nombre de chiffres égal à m+1)
La représentation de $\mathcal{S}(m-n-j)$ sur une B-longueur de m+1 chiffres s'écrit donc $(\mathcal{S}(m-n-j)_0, \mathcal{S}(m-n-j)_1, \ldots, \mathcal{S}(m-n-j)_m)$ avec
$B-1 \geq \mathcal{S}(m-n-j)_i \geq 0 \quad \forall i=0,1,\ldots,m$.
Définissons maintenant $\mathcal{S}^*(m-n-j)=\mathcal{S}(m-n-j)_{m-j} \cdot B + \mathcal{S}(m-n-j)_{m-j-1}$, le nombre obtenu en considérant les deux chiffres les plus significatifs de la représentation sur une B-longueur de m+1 chiffres de $\mathcal{S}(m-n-j)$ susceptibles de ne pas être nuls. (***)
$(j=0,1,\ldots,m-n)$
(Remarquons que ceci a un sens puisque $m-j-1 \geq m-(m-n)-1=n-1 \geq 0$ car $n \geq 1$ et vu que $m-j \leq m-0=m$)

(*) Lors du "démarrage"(c'est-à-dire j=0), par convention la suite $(q_{m-n}, q_{m-n-1}, \ldots, q_{m-n-j+1})$ est vide. De plus, par convention aussi, si j=0,
$\sum_{i=m-n-j+1}^{m-n} q_i B^i = \sum_{i=m-n+1}^{m-n} q_i B^i = \emptyset$.

(**) Cette relation peut se démontrer facilement. En effet.
$\mathcal{S}(m-n-j)=a'-b'\sum_{i=m-n-j+1}^{m-n} q_i B^i = b'\sum_{i=0}^{m-n} q_i B^i + r' - b'\sum_{i=m-n-j+1}^{m-n} q_i B^i =$

$b'\sum_{i=0}^{m-n-j} q_i B^i + r' \geq 0$

puisque $b'>0$, $\sum_{i=0}^{m-n-j} q_i B^i \geq 0$ et $r' \geq 0$ puisque r' représente le reste de la division de a' par b'. ⟶ suite page 92

suite de la page 91.

Montrons maintenant que $\beta(m-n-j) < b' B^{m-n-j+1}$.

Nous venons de voir que $\beta(m-n-j) = b' \sum_{i=0}^{m-n-j} q_i B^i + r'$, *r' étant le reste de la division de a' par b'. Comme* $r' < b'$, *on a que* $\beta(m-n-j) < b' \sum_{i=0}^{m-n-j} q_i B^i + b' = b' (\sum_{i=0}^{m-n-j} q_i B^i + 1) \leq b' B^{m-n-j+1}$.

Remarquons enfin que puisque $b' < B^n$ *(voir propriété 2 page 85), on a que* $b' B^{m-n-j+1} < B^{m-j+1}$

(***) La relation $\beta(m-n-j) < b' B^{m-n-j+1} < B^{m-j+1}$ *montre en effet que la suite* $(\beta(m-n-j)_{m-j+1}, \ldots, \beta(m-n-j)_m)$ *n'est composée que de zéros.*

Enfin, définissons $q^*_{m-n-j} = \left\lfloor \frac{\mathcal{S}^*(m-n-j)}{b'_{n-1}} \right\rfloor = \left\lfloor \frac{\mathcal{S}(m-n-j)_{m-j} B + \mathcal{S}(m-n-j)_{m-j-1}}{b'_{n-1}} \right\rfloor$ (*)

et $\bar{q}_{m-n-j} = \min(q^*_{m-n-j}, B-1) \quad (j=0,1,\ldots,m-n)$

(Il est donc garanti que $\bar{q}_{m-n-j}$ est un B-entier)

Nous allons montrer que $\bar{q}_{m-n-j}$ est une approximation précise de q_{m-n-j} et plus précisément même que $-2 \leq q_{m-n-j} - \bar{q}_{m-n-j} \leq 0$, à la condition que $3 < B/2 \leq b'_{n-1}$, autrement dit que $\bar{q}_{m-n-j}$ est toujours plus grand ou égal à q_{m-n-j} et l'est au plus de 2 unités.
La preuve de l'affirmation ci-dessus sera obtenue par une suite d'inégalités que nous démontrerons chacune.
Par définition de $\mathcal{S}^*(m-n-j)$, on a

$$0 \leq \mathcal{S}(m-n-j) - \mathcal{S}^*(m-n-j) B^{m-j-1} = \sum_{i=0}^{m-j-2} \mathcal{S}(m-n-j)_i B^i < B^{m-j-1} \qquad (1)$$

Aussi par définition de b'_{n-1}, on a

$$b'_{n-1} B^{n-1} \leq b' < (b'_{n-1}+1) B^{n-1} \qquad (2) \quad (\text{car } B^{n-1} \geq 1 \text{ car } n \geq 1)$$

Remarquons que les inégalités (1) et (2) mettent respectivement en évidence les rapports entre $\mathcal{S}(m-n-j)$ et $\mathcal{S}^*(m-n-j)$ et entre b' et b'_{n-1}.

(*) Nous savons que $\mathcal{S}(m-n-j)_{m-j}$ et $\mathcal{S}(m-n-j)_{m-j-1}$ sont des B-entiers positifs ou nuls. Pour être sûr que le calcul de q^*_{m-n-j} (et donc de $\bar{q}_{m-n-j}$) soit réalisable, il y a lieu d'imposer une contrainte limitant supérieurement le choix de B. Ainsi, si nous travaillons sur une machine de type IBM avec la représentation des nombres dans des mots de 4 bytes selon la technique du complément, on a comme contrainte que $(B-1)B+(B-1) = B^2-1 \leq 2^{31}-1$, c'est-à-dire que $B^2 \leq 2^{31}$.

Utilisons les inégalités (1) et (2) pour donner un encadrement de $\frac{\mathcal{S}^*(m-n-j)}{b'_{n-1}}$.

De par la relation (1), nous pouvons dire que $\frac{\mathcal{S}(m-n-j)-B^{m-j-1}}{B^{m-j-1}} < \mathcal{S}^*(m-n-j)$, c'est-à-dire que

$$\mathcal{S}(m-n-j)B^{j-m+1}-1<\mathcal{S}^*(m-n-j) \qquad (3a)$$

De la relation (2), nous pouvons tirer que $b'_{n-1}B^{n-1} \leq b'$, c'est-à-dire que $b'_{n-1} \leq b'B^{-n+1}$ (3b)

Des relations (3a) et (3b), on tire la relation

$$\frac{\mathcal{S}(m-n-j)B^{j-m+1}-1}{b'B^{-n+1}} < \frac{\mathcal{S}^*(m-n-j)}{b'_{n-1}} \qquad (3c)$$

(Nous venons donc de minorer $\frac{\mathcal{S}^*(m-n-j)}{b'_{n-1}}$)

De la relation (1) encore, nous pouvons tirer que $\mathcal{S}^*(m-n-j) \leq \frac{\mathcal{S}(m-n-j)}{B^{m-j-1}}$, c'est-à-dire que $\mathcal{S}^*(m-n-j) \leq \mathcal{S}(m-n-j)B^{-m+j+1}$ (3d)

De la relation (2), nous pouvons tirer que $b'_{n-1} > b'B^{-n+1}-1$ (3e)

Des relations (3d) et (3e), on déduit la relation

$$\frac{\mathcal{S}^*(m-n-j)}{b'_{n-1}} \leq \frac{\mathcal{S}(m-n-j)B^{-m+j+1}}{b'B^{-n+1}-1} \qquad (3f) \quad (*)$$

(Nous venons donc de majorer $\frac{\mathcal{S}^*(m-n-j)}{b'_{n-1}}$)

Rassemblant les relations (3c) et (3f), on obtient

$$\frac{\mathcal{S}(m-n-j)B^{j-m+1}-1}{b'B^{-n+1}} < \frac{\mathcal{S}^*(m-n-j)}{b'_{n-1}} \leq \frac{\mathcal{S}(m-n-j)B^{-m+j+1}}{b'B^{-n+1}-1} \qquad (3)$$

(Nous venons donc de trouver l'encadrement recherché de $\frac{\mathcal{S}^*(m-n-j)}{b'_{n-1}}$)

(*) La relation (3f)(et donc la partie droite de la relation(3)) n'est valable que si $b' \neq B^{n-1}$.

Distinguons maintenant la situation $b' \neq B^{n-1}$ de celle où $b' = B^{n-1}$.

-Si $b' \neq B^{n-1}$ Appliquons maintenant la relation

$$\frac{\rho^*(m-n-j)}{b'_{n-1}} - 1 < \underbrace{\left\lfloor \frac{\rho^*(m-n-j)}{b'_{n-1}} \right\rfloor}_{=q^*_{m-n-j}} \leqslant \frac{\rho^*(m-n-j)}{b'_{n-1}}$$

que nous combinons avec la relation (3). On obtient ainsi

$$\frac{\rho(m-n-j)B^{j-m+1}-1}{b'B^{-n+1}} - 1 < q^*_{m-n-j} \leqslant \frac{\rho(m-n-j)B^{-m+j+1}}{b'B^{-n+1}-1} \qquad (4)$$

(Cette formule fournit un encadrement de q^*_{m-n-j})

Remarquons que

$$q_{m-n-j}B^{m-n-j}b' \leqslant \rho(m-n-j) < (q_{m-n-j}+1)B^{m-n-j}b' \qquad (*)$$

c'est-à-dire que

$$q_{m-n-j}B^{m-n-j} \leqslant \frac{\rho(m-n-j)}{b'} < (q_{m-n-j}+1)B^{m-n-j} \qquad (5)$$

(*) En effet. $\rho(m-n-j) = a' - b' \sum_{i=m-n-j+1}^{m-n} q_i B^i = b'.q + r' - b' \sum_{i=m-n-j+1}^{m-n} q_i B^i$

$$= b' \sum_{i=0}^{m-n} q_i B^i + r' - b' \sum_{i=m-n-j+1}^{m-n} q_i B^i$$

$$= b' \sum_{i=0}^{m-n-j} q_i B^i + r' \quad (r' \text{ est le reste de la division de } a' \text{ par } b' \text{ et donc } 0 \leqslant r' < b')$$

Dès lors $\rho(m-n-j) \geqslant b' q_{m-n-j} B^{m-n-j}$

De plus $\rho(m-n-j) = b' \sum_{i=0}^{m-n-j} q_i B^i + r' < b' \left(\sum_{i=0}^{m-n-j} q_i B^i + 1 \right) \leqslant b' (q_{m-n-j}+1) B^{m-n-j}$

Combinons maintenant les relations (4) et (5) afin d'obtenir un encadrement de $q_{m-n-j} - q^{*}_{m-n-j}$.
De la relation (4), on déduit que

$$\frac{-\varsigma(m-n-j)B^{-m+j+1}}{b'B^{-n+1}-1} \leq -q^{*}_{m-n-j} < \frac{-\varsigma(m-n-jB^{j-m+1}+1}{b'B^{-n+1}} + 1 \qquad (6a)$$

De la relation (5), on peut déduire que

$$q_{m-n-j} \leq \frac{\varsigma(m-n-j)B^{-m+n+j}}{b'} \qquad (6b)$$

et que $$\frac{\varsigma(m-n-j)B^{-m+n+j}}{b'} - 1 < q_{m-n-j} \qquad (6c)$$

En combinant la relation (6a) avec les relations (6b) et (6c), on a

$$\frac{\varsigma(m-n-j)B^{-m+n+j}}{b'} - 1 - \frac{\varsigma(m-n-j)B^{-m+j+1}}{b'B^{-n+1}-1} < q_{m-n-j} - q^{*}_{m-n-j}$$

$$< \frac{\varsigma(m-n-j)B^{-m+n+j}}{b'} + \frac{-\varsigma(m-n-j)B^{j-m+1}+1}{b'B^{-n+1}} + 1$$

qui se réduit après simplification (*) à la relation

$$\frac{-\varsigma(m-n-j)B^{-m+2n+j-1}}{b'(b'-B^{n-1})} - 1 < q_{m-n-j} - q^{*}_{m-n-j} < 1 + \frac{B^{n-1}}{b'}$$

Mais comme nous sommes dans le cas $b' \neq B^{n-1}$ et donc $b' > B^{n-1}$, nous pouvons écrire la relation

$$\frac{-\varsigma(m-n-j)B^{-m+2n+j-1}}{b'(b'-B^{n-1})} - 1 < q_{m-n-j} - q^{*}_{m-n-j} < 1 + \frac{B^{n-1}}{b'} < 2 \qquad (6)$$

(*) La relation

$$\frac{\rho(m-n-j)B^{-m+n+j}}{b'} - 1 - \frac{\rho(m-n-j)B^{-m+j+1}}{b'B^{-n+1}-1} < q_{m-n-j} - q^{*}_{m-n-j} <$$

$$\frac{\rho(m-n-j)B^{-m+n+j}}{b'} + \frac{-\rho(m-n-j)B^{j-m+1}+1}{b'B^{-n+1}} + 1 \quad \text{est équivalente à}$$

(par réduction au même dénominateur)

$$\frac{\rho(m-n-j)B^{-m+n+j}(b'B^{-n+1}-1)-b'(b'B^{-n+1}-1)-\rho(m-n-j)B^{-m+j+1}b'}{b'(b'B^{-n+1}-1)} <$$

$$q_{m-n-j} - q^{*}_{m-n-j} < \frac{\rho(m-n-j)B^{-m+n+j}B^{-n+1}-\rho(m-n-j)B^{j-m+1}+1+b'B^{-n+1}}{b'B^{-n+1}}$$

elle-même équivalente à

$$\frac{\cancel{\rho(m-n-j)b'B^{-m+j+1}}-\rho(m-n-j)B^{-m+n+j}-b'^{2}B^{-n+1}+b'-\cancel{\rho(m-n-j)B^{-m+j+1}}b'}{b'(b'B^{-n+1}-1)} <$$

$$q_{m-n-j} - q^{*}_{m-n-j} < \frac{1+b'B^{-n+1}}{b'B^{-n+1}} \quad \text{encore équivalente à}$$

$$\frac{-\rho(m-n-j)B^{-m+n+j}-b'(b'B^{-n+1}-1)}{b'(b'B^{-n+1}-1)} < q_{m-n-j} - q^{*}_{m-n-j} < 1 + \frac{B^{n-1}}{b'}$$

encore équivalente à

$$\frac{-\rho(m-n-j)B^{-m+n+j}}{b'(b'B^{-n+1}-1)} - 1 < q_{m-n-j} - q^{*}_{m-n-j} < 1 + \frac{B^{n-1}}{b'}$$

enfin équivalente à

$$\frac{-\rho(m-n-j)B^{-m+2n+j-1}}{b'(b'-B^{n-1})} - 1 < q_{m-n-j} - q^{*}_{m-n-j} < 1 + \frac{B^{n-1}}{b'}$$

-Si $b'=B^{n-1}$

Alors $q^{*}_{m-n-j}=q_{m-n-j}$ $\left\lfloor \frac{\mathcal{S}^{*}(m-n-j)}{b'_{n-1}} \right\rfloor = \left\lfloor \mathcal{S}^{*}(m-n-j) \right\rfloor = \mathcal{S}^{*}(m-n-j)$

En effet. $q^{*}_{m-n-j} = \left\lfloor \frac{\mathcal{S}^{*}(m-n-j)}{b'_{n-1}} \right\rfloor = \left\lfloor \mathcal{S}^{*}(m-n-j) \right\rfloor$

$$= \mathcal{S}(m-n-j)_{m-j} + \mathcal{S}(m-n-j)_{m-j-1}$$

Or

$$q_{m-n-j} \underset{(*)}{=} \left\lfloor \frac{\mathcal{S}(m-n-j)}{b'B^{m-n-j}} \right\rfloor = \left\lfloor \frac{\mathcal{S}(m-n-j)}{B^{m-j-1}} \right\rfloor$$

$$= \left\lfloor \frac{\mathcal{S}(m-n-j)_{m-j}B^{m-j} + \mathcal{S}(m-n-j)_{m-j-1}B^{m-j-1} + Z}{B^{m-j-1}} \right\rfloor$$

$$\underset{(**)}{=} \left\lfloor \mathcal{S}(m-n-j)_{m-j}B + \mathcal{S}(m-n-j)_{m-j-1} + \frac{Z}{B^{m-j-1}} \right\rfloor$$

$$= \mathcal{S}(m-n-j)_{m-j}B + \mathcal{S}(m-n-j)_{m-j-1}$$

$$= q^{*}_{m-n-j}$$

Dans pareil cas, bien évidemment, on a $q_{m-n-j}-q^{*}_{m-n-j}=0$.

(*) Pour s'en convaincre, il suffit d'effectuer le raisonnement suivant.

$a'=b'(\sum_{i=0}^{m-n} q_i B^i)+r'$ *où* $0 \leq r' < b'$

c'est-à-dire $a'=b' \sum_{i=m-n+j+1}^{m-n} q_i B^i + b' \sum_{i=0}^{m-n-j} q_i B^i + r'$ *où* $0 \leq r' < b'$

ce qui est équivalent à

$\mathcal{S}(m-n-j)=b' \sum_{i=0}^{m-n-j} q_i B^i + r'$ *où* $0 \leq r' < b'$.

On en déduit que $\left\lfloor \frac{\mathcal{S}(m-n-j)}{b'B^{m-n-j}} \right\rfloor = q_{m-n-j} + \left\lfloor \frac{b' \sum_{i=0}^{m-n-j-1} q_i B^i + r'}{b'B^{m-n-j}} \right\rfloor$

Mais $0 \leq \frac{b' \sum_{i=0}^{m-n-j-1} q_i B^i + r'}{b'B^{m-n-j}} \leq \frac{b'(\sum_{i=0}^{m-n-j-1} q_i B^i + 1)}{b'B^{m-n-j}} \leq \frac{b'B^{m-n-j}}{b'B^{m-n-j}} = 1$

On en déduit finalement que

$\left\lfloor \frac{\mathcal{S}(m-n-j)}{b'B^{m-n-j}} \right\rfloor = q_{m-n-j}$ *puisque* $\left\lfloor \frac{b' \sum_{i=0}^{m-n-j-1} q_i B^i + r'}{b'B^{m-n-j}} \right\rfloor = 0$

(**) Z est une quantité telle que $0 \leq Z < B^{m-j-1}$ *(et donc* $\frac{Z}{B^{m-j-1}} < 1$*)*

Enonçons et démontrons le lemme suivant, toujours dans le but de fournir à plus long terme la preuve que $-2 \leq q_{m-n-j} - \bar{q}_{m-n-j} \leq 0$ à la condition que $3 < B/2 \leq b'_{n-1}$.

Lemme Si $3 < B/2 \leq b'_{n-1}$, alors $-3 \leq q_{m-n-j} - q^{*}_{m-n-j} \leq 0$

Preuve du lemme

Remarquons que la preuve ne doit être fournie que dans le cas où $b' \neq B^{n-1}$ puisque sinon $q_{m-n-j} = q^{*}_{m-n-j}$.
L'inégalité dans l'hypothèse implique $b' \geq B^n/2$ puisque $b' = \sum_{i=0}^{n-1} b'_i B^i$.
Dès lors

$$\frac{\mathcal{S}(m-n-j)B^{-m+2n+j-1}}{b'(b'-B^{n-1})} = \frac{\mathcal{S}(m-n-j)B^{-m+2n+j-1}}{b'b'(1-\frac{B^{n-1}}{b'})} \leq \frac{\mathcal{S}(m-n-j)B^{-m+2n+j-1}}{b'\,\frac{B^n}{2}\,(1-\frac{B^{n-1}}{b'})}$$

$$\leq \frac{\mathcal{S}(m-n-j)B^{-m+2n+j-1}}{b'\frac{B^n}{2}\,(1-\frac{B^{n-1}}{B^n})} \qquad (\text{car } b' \geq B^n/2)$$

$$= \frac{\mathcal{S}(m-n-j)B^{-m+n+j-1}}{b'.\frac{1}{2}.(1-2B^{-1})} < \frac{3\mathcal{S}(m-n-j)B^{-m+n+j-1}}{b'} \qquad (*)$$

Mais $\frac{\mathcal{S}(m-n-j)}{b'} < (q_{m-n-j}+1)B^{m-n-j} \leq B^{m-n-j+1}$ vu la relation (5)

page 95. Tout ceci nous permet de déduire de la relation (6) page 96 que $-4 < q_{m-n-j} - q^{*}_{m-n-j} < 2$

En effet, le fait que $q_{m-n-j} - q^{*}_{m-n-j} < 2$ est inclus dans la relation (6) page 96. Il suffit donc de montrer que $-4 < q_{m-n-j} - q^{*}_{m-n-j}$

(*) Cette inégalité découle du fait que $\frac{1}{\frac{1}{2}(1-2B^{-1})} < 3$ *qui elle-même découle du fait que* $B > 6$.
En effet. $B > 6 \Leftrightarrow 1/B < 1/6 \Leftrightarrow 6B^{-1} - 1 < 0 \Leftrightarrow 2 < 3(1-2B^{-1})$

$$\Leftrightarrow \frac{1}{\frac{1}{2}(1-2B^{-1})} < 3$$

La relation (6) page 96 permet d'écrire que

$$\frac{-\mathcal{S}(m-n-j)B^{-m+2n+j-1}}{b'(b'-B^{n-1})} -1 < q_{m-n-j} - q^{*}_{m-n-j}$$

Mais nous venons d'établir que $\frac{\mathcal{S}(m-n-j)B^{-m+2n+j-1}}{b'(b'-B^{n-1})} < \frac{3\mathcal{S}(m-n-j)B^{-m+n+j-1}}{b'}$

Nous déduisons alors que

$$\frac{-3\mathcal{S}(m-n-j)B^{-m+n+j-1}}{b'} -1 < q_{m-n-j} - q^{*}_{m-n-j}$$

Mais nous venons aussi de mettre en évidence que

$$\frac{\mathcal{S}(m-n-j)}{b'} \leq B^{m-n-j+1}$$

Par conséquent, nous pouvons établir que
$-4 < q_{m-n-j} - q^{*}_{m-n-j}$, ce que nous nous étions promis de démontrer.
Pour poursuivre, remarquons que puisque q_{m-n-j} et q^{*}_{m-n-j} sont des entiers et que $-4 < q_{m-n-j} - q^{*}_{m-n-j} < 2$, nécessairement on a

$$-3 \leq q_{m-n-j} - q^{*}_{m-n-j} \leq 1 \qquad (7)$$

Pour terminer la preuve du lemme, il suffit de montrer qu'il n'est pas possible que $q_{m-n-j} = q^{*}_{m-n-j}$. Pour cela, effectuons le raisonnement suivant.
Par définition de $\mathcal{S}^{*}(m-n-j)$ et q^{*}_{m-n-j} (voir pages 91 et 93), on a que $\mathcal{S}^{*}(m-n-j) = q^{*}_{m-n-j} \cdot b'_{n-1} + r^{*}$, le reste r^{*} satisfaisant la relation $0 \leq r^{*} < b'_{n-1}$.
Supposons par l'absurde que $q_{m-n-j} - q^{*}_{m-n-j} = 1$. Nous aurions donc

$$r^{*} = \mathcal{S}^{*}(m-n-j) - q_{m-n-j} b'_{n-1} + b'_{n-1} \qquad (8)$$

Mais nous savons que

$$q_{m-n-j} b'_{n-1} B^{m-j-1} \underset{(*)}{\leq} q_{m-n-j} b' B^{m-n-j} \underset{(**)}{\leq} \mathcal{S}(m-n-j) \qquad (9)$$

(*) Car $b' = b'_{n-1} B^{n-1} + \ldots \geq b'_{n-1} B^{n-1}$

(**) Voir première partie de la relation (5) page 95

Considérons maintenant les relations (1) page 93 et (8) et (9) page 100.
La relation (1) nous permet d'écrire que
$\varsigma(m-n-j) < B^{m-j-1}(\varsigma^*(m-n-j)+1)$. En combinant ceci avec la relation (9), on obtient la relation
$q_{m-n-j} b'_{n-1} B^{m-j-1} < B^{m-j-1}(\varsigma^*(m-n-j)+1)$
équivalente à $q_{m-n-j} b'_{n-1} < \varsigma^*(m-n-j)+1$.
Par conséquent $\varsigma^*(m-n-j) - q_{m-n-j} b'_{n-1} > -1$, ce qui combiné avec la relation (8) permet d'écrire $r^* > b'_{n-1} - 1$, c'est-à-dire $r^* \geq b'_{n-1}$.
Or ceci est en contradiction avec la définition de r^* qui satisfait la relation $0 \leq r^* < b'_{n-1}$.
Nous sommes donc bien forçés d'admettre que l'égalité $q_{m-n-j} - q^*_{m-n-j} = 1$ est impossible.
Ceci termine la preuve de la démonstration du lemme.

Nous sommes maintenant en mesure de terminer la preuve de l'affirmation énoncée page 93, c'est-à-dire que si $3 < B/2 \leq b'_{n-1}$, alors $-2 \leq q_{m-n-j} - \bar{q}_{m-n-j} \leq 0 \ (j=0,1,\ldots,m-n)$.
Montrons tout d'abord que l'inégalité du lemme portant sur q^*_{m-n-j} est encore vérifiée par $\bar{q}_{m-n-j}$, c'est-à-dire que $-3 \leq q_{m-n-j} - \bar{q}_{m-n-j} \leq 0$.

Si $0 \leq q^*_{m-n-j} \leq B-1$, alors $\bar{q}_{m-n-j} = q^*_{m-n-j}$. Le résultat est évident dans pareil cas.
Si $q^*_{m-n-j} > B-1$, alors $\bar{q}_{m-n-j} = B-1$ (par définition de $\bar{q}_{m-n-j}$)

Donc $q_{m-n-j} - \bar{q}_{m-n-j} = q_{m-n-j} - (B-1) \leq 0$ (puisque $0 \leq q_{m-n-j} \leq B-1$)
Nous savons aussi que $q_{m-n-j} - q^*_{m-n-j} \geq -3$
Or, par hypothèse $q^*_{m-n-j} \geq B$ et $\bar{q}_{m-n-j} = B-1$
Dès lors $q_{m-n-j} - \bar{q}_{m-n-j} \geq q_{m-n-j} - q^*_{m-n-j} \geq -3$ et donc
$q_{m-n-j} - \bar{q}_{m-n-j} \geq -3$.

Remarquons maintenant qu'un raisonnement strictement analogue permet d'écrire $-2 \leq q_{m-n-j} - q^{*}_{m-n-j} \leq 0 \Rightarrow -2 \leq q_{m-n-j} - \bar{q}_{m-n-j} \leq 0$.

Ceci permet d'affirmer que pour montrer qu'on a bien $-2 \leq q_{m-n-j} - \bar{q}_{m-n-j} \leq 0$, il suffit de montrer que si $q_{m-n-j} - q^{*}_{m-n-j} = -3$, alors $-2 \leq q_{m-n-j} - \bar{q}_{m-n-j} \leq 0$.

En fait, nous allons même montrer que dans pareil cas, $q_{m-n-j} - \bar{q}_{m-n-j} = -1$ ou ϕ.

Supposons donc $q_{m-n-j} - q^{*}_{m-n-j} = -3$. Nous pouvons certifier dès lors que $b' \neq B^{n-1}$ (car si $b' = B^{n-1}$, alors $q^{*}_{m-n-j} = q_{m-n-j}$. Voir pour cela le raisonnement page 98.) Nous pouvons donc appliquer la formule (6) page 96 et écrire

$$\frac{-\varsigma(m-n-j)B^{-m+2n+j-1}}{b'(b'-B^{n-1})} - 1 < q_{m-n-j} - q^{*}_{m-n-j} = -3$$

Dès lors $$\frac{-\varsigma(m-n-j)B^{-m+2n+j-1}}{b'(b'-B^{n-1})} < -2$$

c'est-à-dire $$\frac{\varsigma(m-n-j)B^{-m+2n+j-1}}{b'} > 2(b'-B^{n-1})$$

ou encore

$$\frac{\varsigma(m-n-j)B^{-m+n+j}}{b'} > 2(b'-B^{n-1})B^{-n+1} = 2(b'B^{-n+1}-1)$$

et donc $$q_{m-n-j} \underset{(*)}{=} \left\lfloor \frac{\varsigma(m-n-j)B^{-m+n+j}}{b'} \right\rfloor \geqslant 2(b'B^{-n+1}-1) \underset{(**)}{\geqslant} 2(b'_{n-1}-1)$$

(*) Voir la remarque (*) page 98

(**) Car $b'B^{-n+1} - 1 = b'_{n-1} + Z - 1$ où $0 \leq Z$

Mais nous savons par hypothèse que $b'_{n-1} \geq B/2$.
Dès lors $q_{m-n-j} \geq B-2$. Mais comme $q^*_{m-n-j} = q_{m-n-j} + 3$, on peut certifier que $q^*_{m-n-j} \geq B+1$. Dès lors $\bar{q}_{m-n-j} = B-1$. Mais puisque $q_{m-n-j} = B-2$ ou $q_{m-n-j} = B-1$, on en conclut que $q_{m-n-j} - \bar{q}_{m-n-j} = -1$ ou 0, ce que nous voulions démontrer.

Nous sommes à présent à même de déterminer q_{m-n-j}. En effet. Nous connaissons $\bar{q}_{m-n-j}$ et nous savons que $\bar{q}_{m-n-j}$ est égal à l'une des trois valeurs suivantes, à savoir

q_{m-n-j}, $q_{m-n-j}+1$ ou $q_{m-n-j}+2$. (*)

Il suffit donc de travailler à présent par essai et correction éventuelle, entendons par là de procéder de la manière qui suit.
Nous savons que $q_{m-n-j} = \left\lfloor \frac{\varsigma(m-n-j)}{b'B^{m-n-j}} \right\rfloor$.
Calculons alors la quantité $re = \varsigma(m-n-j) - \bar{q}_{m-n-j} b' B^{m-n-j}$.
Si re n'est pas strictement négatif, cela signifie que $\bar{q}_{m-n-j} = q_{m-n-j}$. Nous avons ainsi déterminé q_{m-n-j}.
Si re est strictement négatif, cela signifie que $\bar{q}_{m-n-j} = q_{m-n-j}+1$ ou $q_{m-n-j}+2$. On calcule alors $(re - \bar{q}_{m-n-j} b' B^{m-n-j}) + b' B^{m-n-j}$, donnant ainsi le nouveau re, soit re'. Si re' n'est pas strictement négatif, celà signifie que $\bar{q}_{m-n-j} = q_{m-n-j}+1$, c'est-à-dire que $q_{m-n-j} = \bar{q}_{m-n-j} - 1$. Nous avons ainsi déterminé q_{m-n-j}. Si re' est strictement négatif, cela signifie que $\bar{q}_{m-n-j} = q_{m-n-j}+2$, c'est-à-dire $q_{m-n-j} = \bar{q}_{m-n-j} - 2$.
(Nous sommes assurés que $re'' = re' + b' B^{m-n-j}$ n'est pas strictement négatif)

(*) <u>Attention</u> Cette affirmation est valable à la condition que B soit pair et strictement plus grand que 6. En effet, cette affirmation est écrite à la condition que $3 < B/2 \leq b'_{n-1}$. Or nous savons que $\lfloor B/2 \rfloor \leq b'_{n-1}$. (Voir page 85, propriété 2) Pour que ($\lfloor B/2 \rfloor \leq b'_{n-1}$) implique ($B/2 \leq b'_{n-1}$), nous sommes obligés de considérer B pair, faute de quoi nous ne sommes en mesure de rien affirmer. Bien que effectivement nous réduisions de l'ordre de moitié le nombre de B acceptables, ceci ne peut être considéré comme un handicap sérieux, et à peine comme un manque de généralité. En effet, ne perdons pas de vue que notre problème initial était le traitement des grands nombres entiers, traitement que nous nous proposions d'effectuer via le <u>choix</u>

suite page 104

suite de la page 103
d'une base B intéressante.

Remarquons d'ailleurs que si nous trouvons une autre façon de multiplier a et b(dividende et diviseur) de manière à obtenir a' et b' tels que $L_B(a')=L_B(a)$ *ou* $L_B(a)+1$ *et* $L_B(b')=L_B(b)$ *et* $B/2 \leq b'_{n-1}$, *tout ce qui précède, ainsi que l'algorithme de division qui suivra restent valables, les B impairs étant alors tolérés.*

Remarquons que l'intérêt de cette méthode de "division-correction"(division car le calcul de $\overline{q}_{m-n-j}$ nécessite le calcul de $\left\lfloor \frac{\beta(m-n-j)_{m-j}B+\beta(m-n-j)_{m-j-1}}{b'_{n-1}} \right\rfloor$ qui nécessite une division, et correction car le calcul de q_{m-n-j} s'obtient par correction de la valeur de $\overline{q}_{m-n-j}$)réside d'une part dans le fait que le calcul de $\overline{q}_{m-n-j}$ est simple et rapide, et d'autre part dans le fait que q_{m-n-j} s'éloigne inférieurement d'au plus deux unités de $\overline{q}_{m-n-j}$, ce qui signifie que l'étape de correction est exécutée au plus deux fois.

Nous sommes maintenant prêts à présenter l'algorithme de division IQR dont les spécifications ont été présentées page 82. Cet algorithme, comme nous l'avions déjà signalé est presque intégralement basé sur les raisonnements et résultats précédents qui constituent d'ailleurs presque complètement une preuve de sa correction. Nous présenterons néanmoins un complément de preuve, celui-ci étant nécessité par les légères adaptations "techniques" qui ont été faites.
Rappelons une fois encore que cet algorithme est garanti correct sous la réserve de prendre la base B satisfaisant aux contraintes $B \geq 6$ et B pair.

-Algorithme

(1) $m \leftarrow L_B(a)$
$n \leftarrow L_B(b)$
(2) si $m<n$ alors $q \leftarrow \emptyset$, $r \leftarrow a$
(3) sinon (3a) $s1 \leftarrow ISIGN(a)$, $s2 \leftarrow ISIGN(b)$
$d \leftarrow \lfloor B/(|b_{n-1}|+1) \rfloor$ (d est le facteur de
$d1 \leftarrow s1*d$, $d2 \leftarrow s2*d$ normalisation)
$a' \leftarrow IPRODDI(a,d1)$, $b' \leftarrow IPRODDI(b,d2)$
$RO(m+1) \leftarrow a'$ (entendons par là : créer une suite RO qui contient la représentation sur une B-longueur de m+1 chiffres de a')
(3b) pour j allant de 0 à m-n, faire
(3b1) $\bar{q} \leftarrow \min(B-1, \lfloor (RO_{m-j}*B+RO_{m-j-1}) / b'_{n-1} \rfloor)$
(3b2)(3b21) si $\bar{q} \neq \emptyset$ alors $N1 \leftarrow IPRODDI(b',\bar{q})$
sinon $N1 \leftarrow \emptyset$
(3b22) $N2$(sans zéro non signif.) $\leftarrow (RO_{m-n-j},\ldots,RO_{m-j})$
(entendons par là : créer la représentation au sens habituel du nombre dont la représentation sur une B-longueur de n+1 chiffres est $(RO_{m-n-j},\ldots,RO_{m-j})$)
(3b23) $TEST \leftarrow IDIF(N2,N1)$
(3b24) tant que $ISIGN(TEST)=-1$ faire
$\bar{q} \leftarrow \bar{q}-1$
$INT \leftarrow ISUM(TEST,b')$
$TEST \leftarrow INT$
$INT1 \leftarrow IDIF(N1,b')$
$N1 \leftarrow INT1$

(3b3)(3b31) $q_{m-n-j} \leftarrow \bar{q}$

(3b32) concaténer la suite N1 par la gauche avec une suite constituée de m-n-j chiffres ϕ donnant ainsi la nouvelle suite N1

(3b33) INT2(sans zéro non ← RO significatif)

(entendons par là : créer la représentation au sens habituel du nombre dont la représentation sur une B-longueur de m+1 chiffres est $(RO_o, RO_1, \ldots, RO_m)$)

(3b34) INT3 ← IDIF(INT2,N1)

(3b35) RO(m+1) ← INT3

(3c)(3c1) si $q_{m-n}=\phi$ alors supprimer q_{m-n} de la suite représentant q

(3c2) si s1*s2=-1 alors INT4 ←INEG(q)
q ← INT4

(3c3) INT5 ←IPROD(b,q)
r ← IDIF(a,INT5)

-Complément de preuve et remarques et commentaires

-L'étape (1) de l'algorithme consiste à calculer $L_B(a)$ et $L_B(b)$. Nous n'avons cependant pas mis au point un algorithme permettant de calculer la B-longueur d'un nombre entier(dont on connaît la représentation en base B). Mais la construction d'un tel algorithme étant triviale, nous avons assimilé cet algorithme à une instruction de base.

-Considérons à présent l'étape (2). Si $m<n$, c'est-à-dire si $L_B(a)<L_B(b)$, alors nous sommes assurés que $|a|<|b|$. Mais bien évidemment $a=0.b+a$. Si $a<0$, on a bien $-|b|<a\leq 0$ puisque $-|b|<a$ puisque $|b|>|a|=-a$. Si $a\geq 0$, on a bien $0\leq a<|b|$ puisque $a=|a|<|b|$. Dans pareil cas donc, ϕ et a sont bien le quotient et le reste de la division de a par b.

-Dans le cas contraire, c'est-à-dire si $L_B(a)\geq L_B(b)$, nous appliquons les raisonnements effectués auparavant qui nous garantissent la correction de l'algorithme dans pareil cas. - Effectuons néanmoins toutes les remarques suivantes.

L'étape (3a) est une étape d'initialisation. Il est clair qu'à la fin de celle-ci (a',b') est la normalisation de $(|a|,|b|)$. (Pour rappel, voir la définition page 85)

Remarquons que d1 et d2 sont des B-entiers non nuls et dès lors qu'il est légitime d'appliquer IPRODDI.(Revoir à cet effet les spécifications de IPRODDI page 72)

L'instruction RO(m+1) ← a' n'est que la traduction de "donner la représentation sur une B-longueur de m+1 chiffres de $\S(m-n-\phi)$"(Relire à cet effet la page 91).

L'étape (3b) consiste à rechercher q_{m-n-j}, $j=0,1,\ldots,m-n$(nous avons donc une boucle sur la valeur de j). Pour un j particulier, nous avons à considérer les étapes (3b1),(3b2) et (3b3).

L'étape (3b1) est la parfaite traduction de la formule relative à la définition de $\overline{q}_{m-n-j}$(voir page 93).

La détermination de RO_{m-j} et RO_{m-j-1} consiste à prendre la jème et la (j+1)ième composante, en commençant par la fin, de la suite RO(m+1).(Opération tout à fait élémentaire)

L'étape (3b2) consiste à déterminer q_{m-n-j} de la connaissance de $\overline{q}_{m-n-j}$. Voyons plus en détail cette étape.
A la fin de l'étape (3b21), la suite N1 représente le produit de b' par $\overline{q}$, c'est-à-dire de b' par $\overline{q}_{m-n-j}$. Le fait de distinguer la situation $\overline{q}=\phi$ de $\overline{q}\neq\phi$ est nécessité par les spécifications de IPRODDI(voir page 72). A la fin de l'étape (3b22), la suite N2 est la représentation au sens habituel du nombre dont la représentation sur une B-longueur de n+1 chiffres est $(RO_{m-n-j},\ldots,RO_{m-j})$. Cette opération est nécessitée par le fait qu'à l'étape (3b23), on applique l'algorithme IDIF qui de par ses spécifications nécessite des représentations initiales des nombres entiers traités au sens strict.
Mais pourquoi considérer $(RO_{m-n-j},\ldots,RO_{m-j})$? La réponse n'est pas tout à fait triviale. Celle-ci est fournie par le raisonnement suivant.
Nous savons que q_{m-n-j} est obtenue par réduction éventuelle de $\overline{q}_{m-n-j}$(réduction éventuellement répétée).(Revoir à ce propos la technique présentée page 103)
Brièvement rappelé, nous retirons 1 de $\overline{q}_{m-n-j}(\overline{q})$ aussi longtemps que $\mathcal{S}(m-n-j)-\overline{q}b'B^{m-n-j}$ est strictement négatif. Mais remarquons que la condition $\mathcal{S}(m-n-j)-\overline{q}b'B^{m-n-j}<0$ est équivalente à une condition ne portant plus que sur unepartie de la suite de B-longueur m+1 représentant $\mathcal{S}(m-n-j)$(et non plus sur la totalité). Pour cela, effectuons le raisonnement suivant.

$$\mathcal{S}(m-n-j)\underset{(*)}{=}\mathcal{S}(m-n-j)_{m-j}B^{m-j}+\ldots+\mathcal{S}(m-n-j)_{m-n-j}B^{m-n-j}+\underbrace{\mathcal{S}(m-n-j)_{m-n-j-1}B^{m-n-j-1}+\ldots+\mathcal{S}(m-n-j)_{o}}_{(**)}$$

(*) Voir à ce propos la remarque (***) page 92.

(**) Cette somme est vide si m-n<j+1

La condition $\varsigma(m-n-j)-\bar{q}b'B^{m-n-j}<0$ est donc équivalente à la condition

$$\underbrace{\varsigma(m-n-j)_{m-j}B^{m-j}+\ldots+\varsigma(m-n-j)_{m-n-j}B^{m-n-j}-\bar{q}B^{m-n-j}b'+\varsigma(m-n-j)_{m-n-j-1}B^{m-n-j-1}+\ldots+\varsigma(m-n-j)_{o}}_{Z\quad(*)}<0$$

elle-même équivalente à la condition (**)

$$\varsigma(m-n-j)_{m-j}B^{n}+\ldots+\varsigma(m-n-j)_{m-n-j}+\underbrace{\varsigma(m-n-j)_{m-n-j-1}B^{-1}+\ldots+\varsigma(m-n-j)_{o}B^{-m+n+j}}_{Z'\quad(***)}<0$$

elle-même équivalente à la condition

$\varsigma(m-n-j)_{m-j}B^{n}+\ldots+\varsigma(m-n-j)_{m-n-j}-\bar{q}b'<0$ puisque le membre de gauche de cette inégalité est un nombre entier.

(fin du raisonnement)

(*) Cette quantité Z est telle que $0\leq Z<B^{m-n-j}$.

(**) On divise chaque membre de l'inégalité par $B^{m-n-j}(>0)$.

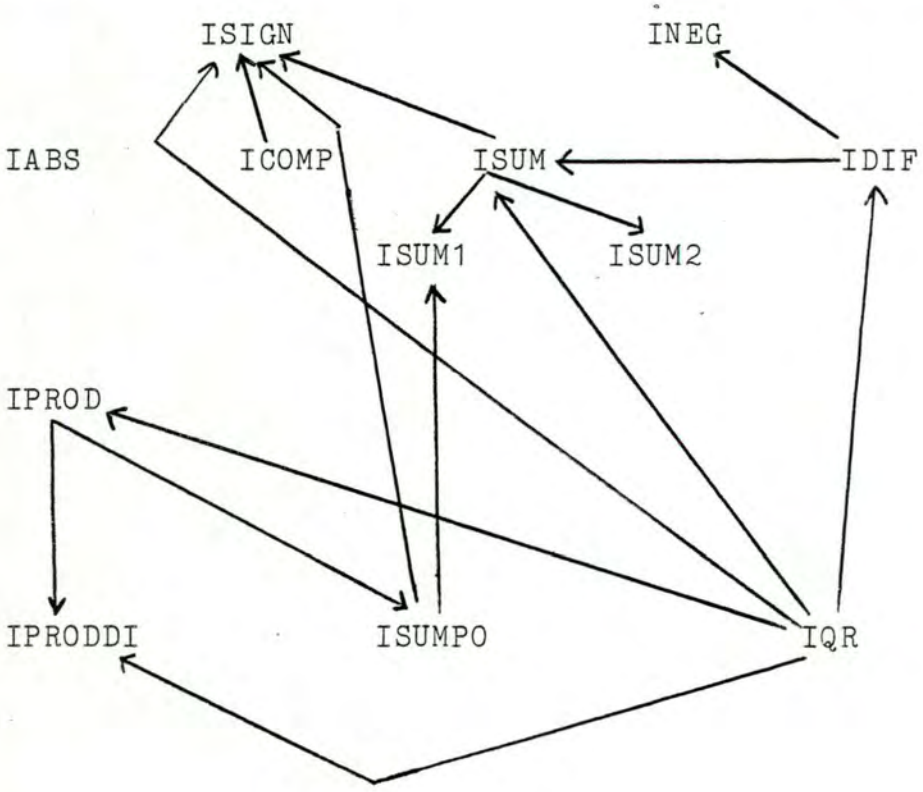
(***) Cette quantité Z' est telle que $0\leq Z'<1$.

Remarquons qu'à la fin de l'étape (3b23), la suite TEST représente bien $\S(m-n-j)_{m-j}B^n+\ldots+\S(m-n-j)_{m-n-j}-\overline{q}b'$.
Le lecteur n'aura aucune difficulté à se convaincre qu'à la fin de l'étape (3b24) $\overline{q}=q_{m-n-j}$ et que la suite N1 représente $\overline{q}b'=q_{m-n-j}b'$.
L'étape (3b31) ne pose donc aucun problème de compréhension.
L'étape (3b32) nous garantit qu'à la fin de celle-ci, la suite N1 représente bien $q_{m-n-j}B^{m-n-j}b'$. A la fin des étapes (3b33) et (3b34) et (3b35), la suite RO est la représentation sur une B-longueur de m+1 chiffres de $\S(m-n-j)-q_{m-n-j}B^{m-n-j}b'$, c'est-à-dire, si $j<m-n$, de $\S(m-n-j-1)$. (Revoir la page 91)
(On prépare ainsi la détermination de $q_{m-n-j-1}$ au passage suivant si celui-ci existe)
L'étape (3c) est l'étape de terminaison. Si q_{m-n} est nul, il y a lieu de supprimer ce chiffre de la suite représentant q pour obtenir la représentation "standard" de q, quotient recherché. (Puisqu'alors q_{m-n-1} est non nul, si celui-ci existe. Voir à ce propos la remarque au bas de la page 89)
C'est le rôle de l'instruction de l'étape (3c1).
Mais ne nous y trompons pas. q est le quotient de la division de |a| par |b|, ceci avant exécution de l'étape (3c2). A la fin de l'exécution de l'étape (3c2), q est le quotient de la division de a par b. (Revoir à ce propos les pages 83 et 84)
L'étape (3c3) quant à elle garantit qu'à la fin de celle-ci, r est égal au reste de la division de a par b, puisque a=b.q+r et donc r=a-b.q.

Au niveau du temps d'exécution de IQR(a,b,q,r), signalons que $t_{IQR(a,b,q,r)} \hat{=} K.L_B(b).(L_B(a)-L_B(b)+1)$, K étant une constante de proportionnalité.

3.3 Synthèse concernant l'utilisation des algorithmes les uns par les autres

En guise de synthèse à tous les algorithmes qui viennent d'être présentés, nous présentons ci-dessous un graphe représentant l'utilisation des algorithmes les uns par les autres.



(*)

(*) *A ⟶ B signifie A utilise B*

Chapitre 4. CHOIX D'UNE REPRESENTATION SOUS FORME DE LISTES DES SUITES REPRESENTANT LES NOMBRES ENTIERS ET MISE EN EVIDENCE DES PRIMITIVES UTILES, VOIRE NECESSAIRES, A METTRE EN OEUVRE POUR LA MISE SUR PIED PRATIQUE DES ALGORITHMES PRESENTES AUPARAVANT(PARALLELEMENT A L'INTERPRETATION DE CES DERNIERS)

Nous l'avons signalé dès le début, les algorithmes qui ont été présentés travaillent à partir de représentations supérieures, ou encore appelées logiques. Nous les avions qualifiés dans ce sens de théoriques.(Voir à ce propos la page 28)
Nous y avons représenté les nombres entiers sous forme de suites de B-entiers. A titre illustratif mais tout de même intéressant, nous allons concrétiser quelque peu cette notion (*) de suite en représentant une suite quelconque(de B-entiers)par une liste(de B-entiers). (**) Nous montrerons alors comment interpréter les algorithmes présentés auparavant et nous exhiberons les primitives relatives aux listes qu'il serait intéressant ou nécessaire de programmer pour exécuter réellement ces algorithmes. Nous mettons dès maintenant le lecteur en garde contre le fait qu'un mauvaise interprétation des algorithmes conduirait inévitablement(et ce indépendamment de l'effort qui a été mis en oeuvre pour fournir les preuves de correction de ces derniers)à des erreurs d'exécution. Autrement dit, malgré le fait que ces algorithmes sont théoriquement corrects, ils ne le seraient plus pratiquement.
Dans le souci d'être clair et logique, nous allons considérer les algorithmes un par un.

(*) Ceci est fait à titre purement illustratif et non par nécessité. Le lecteur se convaincra en effet que d'autres formes de concrétisation de ces suites sont possibles et compatibles avec les algorithmes présentés. Ainsi, par exemple, une suite peut être représentée par un vecteur dont les composantes sont les B-entiers composant la suite (principe de la contiguïté physique). Il est certain que les listes présentent des facilités de manipulation que ne présentent pas les vecteurs(mais aussi certains désavantages).

(**) Le mot liste est à prendre dans le sens de langages comme PASCAL.

1.Algorithme ISIGN

L'algorithme présenté page 29, de par son étape (2), fait apparaître(ou mieux suggère ou encore évoque) les primitives suivantes.

-Déterminer si une liste est vide.

-Prendre le premier élément d'uneliste non vide(tête de liste).

-Déterminer si le dernier élément d'une liste non vide est atteint(sortie de liste)(primitive totalement superflue).

-Prendre l'élément suivant d'un élément d'une liste dont le dernier élément n'est pas atteint.

-Déterminer si la fin d'une liste est atteinte(la fin d'une liste vide est d'office atteinte).

-Prendre l'élément suivant dans une liste dont la fin n'est pas atteinte(l'élément suivant au moment du "démarrage" est la tête)

(Le lecteur se convaincra que ces primitives suffisent de loin à mettre en oeuvre effectivement l'agorithme ISIGN. Les deux dernières d'ailleurs suffisent déjà) (*) page 115.

2. Algorithme INEG

L'algorithme présenté page 34 suggère les primitives suivantes.

-De par l'étape (1)

  -Déterminer si une liste est vide
   (l'interprétation de la condition "a=ϕ"étant "la liste correspondant à a est vide")

  -Créer une liste vide
   (l'interprétation de l'instruction "b $\leftarrow \phi$" étant "créer une liste vide(correspondant à b)")

-De par l'étape (2)

  -Les mêmes primitives que celles présentées pour l'algorithme ISIGN.

  -Ajouter à la droite d'une liste un élément d'un contenu donné (c'est-à-dire ajouter un élément suivant d'une liste, de contenu donné)
   (l'interprétation de l'instruction "$b_i \leftarrow -a_i$" étant "ajouter à la droite de la liste b déjà créée un élément dont on forcera la valeur à $-a_i$". Il est à noter que l'instruction "$b_o \leftarrow -a_o$" nécessite en plus d'avoir créé initialement une liste vide)

Remarquons que pour mettre en oeuvre une telle primitive, nous devons supposer l'existence d'une liste non vide (que nous appellerons RESERVE) de "cellules mémoires" disponibles et dans laquelle nous puisons chaque fois que cela est nécessaire une ou plusieurs "cellules mémoires". Il y a donc "décrochage" d'une cellule mémoire de la liste RESERVE et "accrochage" de celle-ci à la liste correspondant à b (avec affectation à la valeur $-a_i$ courante).
Il est bien évident que pour puiser une "cellule mémoire" dans la liste RESERVE, il faut au préalable s'être assuré que cette dernière n'est pas vide. Nous avons déjà mis en évidence (pour l'algorithme ISIGN) l'utilité d'une primitive effectuant le test de liste vide.

3. Algorithme IABS

L'algorithme présenté page 35 ne met pas en évidence la nécessité d'autres primitives que celles présentées jusqu'à présent. Au niveau de l'interprétation de cet algorithme, remarquons que nous avons des considérations semblables à celles émises pour l'algorithme ISIGN. Ainsi, les instructions telles que "$b_i \leftarrow \emptyset$", "$b_i \leftarrow a_i$", "$b_i \leftarrow -a_i$" de l'étape (2) sont à interpréter de manière semblable et dans le sens qu'elles nécessitent l'ajout de "cellules mémoires" à décrocher au préalable de la liste RESERVE. Par contre, l'instruction "$b \leftarrow \emptyset$" de l'étape (1) est à interpréter dans le sens de la création d'une liste vide. De même pour la comparaison "$a=\emptyset$" de cette même étape, il y a lieu de l'interpréter dans le sens de l'égalité à une liste vide.

(*) Nous parlerons plus souvent de primitives utiles que de primitives nécessaires. Le lecteur se rendra d'ailleurs compte au fur et à mesure du passage en revue des différents algorithmes que certaines primitives présentées pour un même algorithme pourtant ne sont pas indépendantes l'une de l'autre et que l'on pourrait très bien se passer de certaines d'entre elles. Nous travaillerons donc de manière suffisante et non pas de manière nécessaire. Ainsi, pour l'interprétation de l'étape (2) de l'algorithme ISIGN, nous voyons que nous gardons une certaine liberté dans le choix des primitives et de leur combinaison qui réalisera l'étape (2).

4. Algorithme ICOMP

Le lecteur se convaincra du fait que l'algorithme ICOMP présenté page 38 ne nécessite aucune autre primitive que celles déjà mises en évidence jusqu'à maintenant.

5. Algorithme ISUM

L'algorithme présenté page 42 nécessite quelques explications quant à l'interprétation de deux de ses instructions. Ainsi, l'instruction "c ← b"(de même que l'instruction "c ← a") est à interpréter dans le sens de la création d'une liste (correspondant à c) et qui sera une copie conforme de la liste correspondant à b. Ceci suggère la mise au point d'une primitive nouvelle, à savoir une primitive qui permettra de -copier une liste donnée.
Remarquons que pour la mise au point d'une telle primitive, il suffira de combiner judicieusement certaines des primitives déjà présentées auparavant telles que -créer une liste vide, -ajouter à la droite d'une liste un élément de contenu donné,etc..

6. Algorithme ISUM1

L'algorithme présenté page 45 ne nécessite aucune primitive nouvelle. Au niveau de l'interprétation de celui-ci, signalons qúe les instructions "$c_k \leftarrow a_k+b_k$+REPORT"(étape (2)) et "$c_k \leftarrow a_k$+REPORT"(étape (3)) ainsi que "$c_k \leftarrow a_k$"(étape(4)) impliquent l'ajout d'un élément(et donc nécessité de puiser une "cellule mémoire" dans la liste RESERVE)contrairement aux instructions "$c_k \leftarrow c_k$+ADAPTATEUR" des étapes (2) et (3) qui n'exécutent que des modifications du contenu de l'élément en cours(c_k)déjà existant. (*)
Enfin, notons que la combinaison des instructions "k ← k+1" et "c_k ← REPORT" de l'étape (5) équivaut à l'ajout à la liste représentant c d'un élément de contenu égal à celui de REPORT.

(*) Il est à noter que l'interprétation des algorithmes nécessite un minimum de bons sens de la part de l'utilisateur puisqu'à des instructions de formes semblables correspondent des significations parfois différentes. Un utilisateur qui ne comprendrait pas un minimum la "philosophie" des algorithmes présentés et qui verserait dans une programmation aveugle verrait son travail automatiquement voué à l'échec.

7. Algorithme ISUM2

Considérons l'algorithme présenté page 53. Des remarques semblables à celles qui précèdent peuvent être émises en ce qui concerne les instructions "$c_k \leftarrow a_k+b_k$"(étape(1)), "$c_k \leftarrow a_k$"(étape(2)), "$c_{k+1} \leftarrow a_{k+1}$", "$c_v \leftarrow a_v$"(étape(5)) et "$c_i \leftarrow a_i$"(étape (7)).(Nécessité de puiser une "cellule mémoire" dans la liste RESERVE et ajout à la liste en cours avec affectation à la valeur adéquate). Les instructions "$c_i \leftarrow$ d+ADAPTATEUR" et "$c_i \leftarrow$ d" de l'étape (6) équivalent à de simples modifications de contenu. Elles ne requièrent donc aucune "cellule mémoire" supplémentaire. L'instruction "$c \leftarrow \emptyset$" de l'étape (2) est à interpréter, comme d'habitude, dans le sens de la création d'une liste vide.
Au niveau des primitives, remarquons qu'il y a intérêt à mettre au point une primitive de -suppression des éléments de fin d'une liste et d'un contenu donné(utile à l'étape (8)). Celle-ci se baserait sur des primitives nouvelles telles que
-prendre le dernier élément d'une liste non vide,
-déterminer si le premier élément d'un liste non vide est atteint,
-prendre l'élément précédent d'un élément d'une liste dont le premier élément n'est pas atteint, (*)
mais aussi sur une primitive nouvelle de
-libération de "cellules mémoires" correspondant à des éléments d'une liste devenus inutiles.(Entendons par là le "décrochage" des éléments devenus inutiles de la liste considérée et "l'accrochage" à la liste RESERVE).
En fait, même si cela n'apparaît pas dans les algorithmes (comme si cela était superflu), il est capital que lors de l'interprétation des algorithmes théoriques présentés dans ce travail, l'utilisateur ait à tout moment à l'esprit la phrase "N'y a-t-il pas maintenant moyen de "récupérer" dans la zone RESERVE certaines "cellules mémoires" qui seraient devenues "inutiles""et qu'il agisse en conséquence. En effet, toute négligence de ce côté conduirait à un gaspillage de place mémoire mais pire encore, à court terme, à un arrêt

(*) Remarquons que nous mettons ici en évidence la nécessité de pouvoir parcourir les listes de la droite vers la gauche, c'est-à-dire vers l'arrière.

obligatoire du programme exécuté, par le fait que la liste RESERVE serait vide(situation équivalente à l'absence de place mémoire disponible).
Cette primitive de libération-récupération serait aussi utile à l'étape (2) de l'algorithme(voir cette étape). En effet, dans le cas où u est nul et que les deux suites sont finies, alors, en même temps que de créer une liste vide qui donnera le résultat c(instruction "c ← ϕ"), il y a lieu de "récupérer" les "cellules mémoires" qui peuvent être libérées puisque devenues inutiles(et qui avaient été créées par l'instruction "$c_k \leftarrow a_k+b_k$" de l'étape (1).
Enfin, remarquons que les instructions "l ← k"(étape(1)) et "l ← v"(étape(5)) symbolisent le fait qu'il y a lieu de garder, temporairement du moins, l'adresse de l'élément considéré à l'instant dans la liste.

8. Algorithme IDIF

Il n'y a rien de particulier à signaler ni au niveau de l'interprétation ni au niveau de la mise au point de primitives nouvelles relativement à l'algorithme IDIF présenté page 64, si ce n'est le fait qu'il y a lieu juste avant la fin de l'exécution de l'algorithme, de prévoir la libération-récupération de toutes les "cellules mémoires" monopolisées par les éléments de la liste représentant d, elle-même étant devenue totalement inutile.(d jouait le rôle d'une liste intermédiaire)

9. Algorithme IPROD

Aucune nécessité de primitive nouvelle n'apparaît dans l'algorithme présenté page 69.
En ce qui concerne l'interprétation de l'étape (1), signalons que les comparaisons "a=ϕ" et "b=ϕ" équivalent à déterminer si les listes correspondantes sont vides et que l'affectation "c ← ϕ" est équivalente à la création d'une liste vide correspondant à c(même remarque pour l'instruction "c ← ϕ" de l'étape (3)).
Il est important de remarquer qu'à la fin de l'étape (5)(b), tous les éléments correspondant à la liste P peuvent être récupérés et qu'il en est de même pour les éléments de la liste intermédiaire d à la fin de l'étape (5)(c).
Enfin, signalons que l'instruction "c ← d" de l'étape (5)(c)

est à interpréter dans le sens de la création d'une liste, copie conforme de celle correspondant à d, et que l'expression "pour i allant de 0 à n-1" de l'étape (4) est équivalente à l'expression "tant que la fin de la liste correspondant à b n'est pas atteinte".

10. Algorithme IPRODDI

Les primitives déjà présentées suffisent à mettre en oeuvre l'algorithme IPRODDI présenté page 72. Côté interprétation, signalons que les instructions "$c_i \leftarrow$ d mod B" de l'étape (2) et "$c_m \leftarrow$ REPORT" de l'étape (3) impliquent l'ajout d'une "cellule mémoire". L'instruction "i $\leftarrow$ i+1" est équivalente à "considérer l'élément suivant dans la liste(s'il existe)".

11. Algorithme ISUMPO

Considérons l'algorithme présenté page 76. Les instructions "$c_i \leftarrow a_i$"(étape(1)), "$c_i \leftarrow \emptyset$"(étape(3)(a)) et "$c_i \leftarrow b_{i-e}$" (étape(3)(b)) impliquent l'ajout d'un élément à la liste correspondant à c. Les instructions "i $\leftarrow$ i+1" dans ces mêmes étapes équivalent à "considérer l'élément suivant(s'il existe)". Remarquons que l'étape (2)(c) met en évidence l'utilité d'une primitive permettant de -concaténer deux listes.

12. Algorithme IQR

Enfin, pour terminer, considérons l'algorithme de division présenté pages 106 et 107.
L'instruction "q $\leftarrow \emptyset$" de l'étape (2) est équivalente à une instruction de création de liste vide correspondant à q. L'instruction "r $\leftarrow$ a" de cette même étape est équivalente à la création d'une liste, copie conforme de celle correspondant à a.
L'instruction "RO(m+1) $\leftarrow$ a'" peut être réalisée au moyen de la primitive déjà présentée de copie d'une liste donnée et de la primitive elle aussi déjà évoquée d'ajout à la droite d'une liste d'un élément de contenu donné.
L'étape (3b22) est réalisable au moyen des primitives déjà évoquées permettant de copier une liste donnée et de supprimer les éléments de fin d'une liste et d'un contenu donné.

Il est important de remarquer qu'à la fin de l'étape (3b23), les éléments de la liste N2 doivent être récupérés dans la liste prévue à cet effet, soit RESERVE. Il en est de même pour les éléments de chacune des listes telles que TEST,INT,INT1, INT2, etc... dès que celles-ci ont cessé d'être utiles.
(On laissera au lecteur le soin de les retrouver toutes et de voir à quel moment cela arrive)
Remarquons que les instructions (3b1) et (3b22) évoquent une primitive nouvelle permettant de -trouver le jème élément d'une liste(comptage à partir de la droite de la liste et en débutant à ϕ). Enfin, signalons que l'instruction (3b31) implique la création d'un nouvel élément et évoque une primitive nouvelle qui permet -d'ajouter à la gauche d'une liste un élément d'un contenu donné, et que l'étape (3c1) peut être réalisée au moyen de la primitive déjà évoquée de suppression des éléments de fin d'une liste et d'un contenu donné.

Nous venons de mettre en évidence un certain nombre de primitives relatives au traitement des listes et qu'il serait bon, voire nécessaire, de programmer pour appliquer les algorithmes théoriques présentés auparavant. Certes, avoir relevé ces primitives est important, mais ce qui l'est bien plus, c'est de se rendre compte que ces algorithmes ne font appel pour pouvoir être mis pratiquement en oeuvre via la notion de listes qu'à des primitives classiques et simples concernant le traitement des listes. (*)
C'est finalement à l'utilisateur de tels algorithmes qu'il sera laissé le soin de détailler de manière minutieuse(bien mieux que cela n'a été fait ci-dessus)les primitives dont il a vraiment besoin.

(*) Le lecteur habitué de manipuler les listes dans n'importe quel langage de programmation sera facilement persuadé de la chose.

Chapitre 5. DISCUSSION CONCERNANT LE CHOIX DE LA BASE B

Lors de la présentation des différents algorithmes, nous avons à l'occasion signalé quelques contraintes concernant la base B. Ces contraintes étaient essentiellement des obligations pour la base B de ne pas dépasser telle ou telle valeur(limitation supérieure de B).
Nous allons à présent reprendre ces différentes contraintes et les compléter en passant en revue algorithme par algorithme et essayer finalement de dégager parmi les valeurs acceptables de B une ou plusieurs valeurs particulières et intéressantes.
Appelons MAX, la valeur maximum représentable pour une nombre entier dans le langage utilisé, sur ma machine utilisée. (*)
(Le plus petit nombre entier représentable étant alors généralement -MAX-1)
Il est évident qu'une première contrainte relative à B est $2 \leq B \leq MAX+1$ (1).(Cette contrainte découle du fait que les B-entiers doivent être représentables comme des nombres entiers)
L'algorithme ISIGN présenté page 29 pas plus que l'algorithme INEG présenté page 34 ou que l'algorithme IABS présenté page 35 ne met en évidence d'autres contraintes que la contrainte (1) ci-dessus. Considérons alors l'algorithme ICOMP présenté page 38. Une nouvelle contrainte relative au choix de B apparaît puisqu'à l'étape (2) de l'algorithme on trouve l'instruction "s ← sign(a_k-b_k)" qui nécessite le calcul de a_k-b_k.
Nous savons que $-(B-1) \leq a_k \leq B-1$ et $-(B-1) \leq b_k \leq B-1$. De là, on déduit que $-2(B-1) \leq a_k - b_k \leq 2(B-1)$ et que les valeurs $-2(B-1)$ et $2(B-1)$ peuvent être effectivement atteintes. D'où l'on tire la contrainte $2(B-1) \leq MAX$ (2).
Si l'algorithme ISUM présenté page 42 fait apparaître une nouvelle contrainte, ce ne sera, vu sa structure, que via les algorithmes ISUM1 et ISUM2 présentés respectivement pages 45 et 53.

(*) Cette valeur MAX dépend effectivement du langage utilisé et de la machine sur laquelle on travaille, bien que MAX ne prenne en pratique que quelques valeurs habituelles. Ainsi, sur une machine de type IBM pour laquelle les nombres entiers sont représentés dans des mots de 4 bytes par la technique du complément, cette valeur MAX est égale à $2^{31}-1$.

Considérons en premier lieu l'algorithme ISUM1. La variable ADAPTATEUR dont question à l'étape (1) pouvant recevoir les valeurs -B et B, on obtient la contrainte $B \leq MAX$ (3).
L'instruction "$c_k \leftarrow a_k+b_k$+REPORT" de l'étape (2) implique la contrainte $2B-1 \leq MAX$ (4) puisque $-(B-1) \leq a_k \leq B-1$ et $-(B-1) \leq b_k \leq B-1$ et $|REPORT| \leq 1$. Aucune autre contrainte n'y apparaît. (*)
En ce qui concerne l'algorithme ISUM2, remarquons que de par les instructions "ADAPTATEUR ← B" et "ADAPTATEUR ← -B" de l'étape (6), on retrouve la contrainte $B \leq MAX$ déjà citée.
L'instruction "d ← c_i+REPORT" de cette même étape fournit la même contrainte $B \leq MAX$. Le lecteur se convaincra que les instructions "$c_k \leftarrow a_k+b_k$" et "c_i ← d+ADAPTATEUR" des étapes (1) et (6) ne donnent lieu à aucune contrainte.
L'algorithme IDIF page 64 ne met en évidence aucune nouvelle contrainte.
L'algorithme IPROD présenté page 69 imposera de nouvelles contraintes mais seulement via les algorithmes IPRODDI et ISUMPO.(Voir à ce propos la remarque page 71 concernant le compteur de décalage CD relativement à l'étape (6) de l'algorithme)
L'algorithme IPRODDI présenté page 72 fait apparaître la nouvelle contrainte $B(B-1) \leq MAX$ (5). (Voir à ce propos les remarques et commentaires relatifs à l'algorithme IPRODDI page 75)
Quant à l'algorithme ISUMPO présenté page 76, il est évident de voir que celui-ci ne met en évidence aucune nouvelle contrainte. Enfin, l'algorithme de division IQR présenté pages 106 et 107 a mis en évidence l'intérêt de considérer $B > 6$ et B pair (6) mais aussi la nécessité de prendre B tel que $B^2-1 \leq MAX$ (7)(voir à ce propos la remarque (*) page 93)(la contrainte énoncée page 88 est moins forte et donc superflue).

(*) L'instruction "$c_k \leftarrow c_k$+ADAPTATEUR" des étapes (2) et (3) ne fait apparaître aucune nouvelle contrainte car ADAPTATEUR est du signe contraire à celui de a et b(voir étape(1) d'initialisation)

Si nous synthétisons les contraintes relatives à B, on a donc le système

$$\begin{cases} 2 \leq B \leq MAX+1 \\ 2(B-1) \leq MAX \\ B \leq MAX \\ 2B-1 \leq MAX \\ B(B-1) \leq MAX \\ B > 6 \text{ et } B \text{ pair} \\ B^2-1 \leq MAX \end{cases}$$

équivalent au système

$$\begin{cases} B > 6 \text{ et } B \text{ pair} \\ B^2 \leq MAX+1 \end{cases}$$

Avouons que ces contraintes sont finalement extrêmement faibles, ce qui a l'avantage de nous laisser une grande liberté dans le choix de B.
Un travail intéressant consisterait à mesurer réellement les temps d'exécution des différents algorithmes pour des données fixées mais avec différentes valeurs de B. On ose croire à priori que les valeurs de B seraient d'autant plus intéressantes qu'elles seraient grandes, ne serait-ce d'ailleurs que parce que plus B est grand, moins $L_B(a)$ est grand(pour a nombre entier fixé)et donc moins le nombre de pointeurs(on suppose une représentation des nombres sous forme de listes)est élevé. Le raisonnement n'est plus aussi marquant cependant si on travaille avec une représentation par "contiguïté physique" comme tend à le prouver la propriété page 6(voir aussi à ce propos la page 21). Mais seuls des résultats de simulation seraient à même de nous confirmer ces impressions(ou de les infirmer).
Mais, il y a un point important et dont nous n'avons encore jamais parlé dans ce travail et dont pourtant il faut tenir compte pour le choix de B. Il est évident que les nombres entiers que nous manipulons via les différents algorithmes(quelle que soit la base B choisie) ont dû être tôt ou tard "entrés" dans la machine(disons "lus"). On peut sans perte de généralité (vu notre habitude de travailler dans le système de position décimal)considérer que ces nombres sont présentés sous une forme "caractères" proche du système de position décimal.

Il faut donc entre-temps prévoir un algorithme de conversion des nombres entiers exprimés sous forme "caractères" en les nombres entiers exprimés en base B. De même, il y a lieu de prévoir un algorithme de conversion inverse permettant d'obtenir l'expression d'un nombre entier sous forme "caractères" à partir de l'expression de ce même nombre entier en base B.
On peut mettre au point de tels algorithmes, sans grande difficulté(valables quelle que soit la base B). Mais il est important de remarquer que si nous prenons des bases B de la forme 10^n($n \in N$ et $n \geqslant 1$), les algorithmes de conversion et de conversion inverse s'en trouvent beaucoup simplifiés et leur temps d'exécution réduits.
Il semble donc intéressant de considérer des bases B de la forme 10^n($n \in N$ et $n \geqslant 1$) satisfaisant à la contrainte $10^{2n} \leqslant MAX+1$. (Les autres contraintes $B > 6$ et B pair étant d'office satisfaites)
A titre d'exemple, essayons de voir la base B du type 10^n, maximum acceptable en supposant que $MAX=2^{31}-1$. La contrainte s'écrit alors $10^{2n} \leqslant 2^{31}$, c'est-à-dire $10^{2n} \leqslant 2\ 147\ 483\ 648$. Le n maximum acceptable est donc 4, ce qui donne la base B de type 10^n maximum acceptable égale à 10^4, c'est-à-dire 10 000.
L'algorithme de conversion utiliserait dans pareil cas des groupements des caractères représentant les chiffres de la représentation dans le système de position décimal des nombres entiers considérés par paquets de quatre en partant des chiffres les moins significatifs vers les plus significatifs, ceci pour obtenir les B-entiers de la représentation en base B de ces nombres. L'algorithme de conversion inverse ferait le travail réciproque.
Signalons aussi pour terminer sur ce point qu'il serait intéressant de considérer des bases du type 2^n. Une base de ce type maximum serait égale à 2^{15} puisque ces bases doivent satisfaire la contrainte $2^{2n} \leqslant 2^{31}$. On pourrait réaliser des simulations se basant sur B du type 2^n où n varie entre 3 et 15.

CONCLUSION

Nous nous bornerons dans cette conclusion à exprimer les extensions possibles de ce travail.
Il est bien évident que le travail qui vient d'être présenté peut être complété. Complété tout d'abord au niveau des problèmes d'arithmétique des grands nombres entiers traités. Ainsi, nous l'avons déjà signalé dans le dernier chapitre, il y aurait lieu d'écrire un algorithme de détermination de la représentation d'un nombre entier exprimé en base B en base $\mathcal{B}$.(Algorithme de changement de base). Il y aurait lieu aussi d'écrire des algorithmes relatifs aux opérations d'exponentiation, de détermination du plus grand commun diviseur ou du plus petit commun multiple. Quand je dis écrire, je pense bien évidemment aussi aux preuves de correction accompagnantes.
Complété aussi et surtout dans le sens qu'il y aurait lieu d'écrire effectivement des programmes rendus exécutables dans un langage particulier(tel que PASCAL ou C)sur une machine particulière et correspondant aux algorithmes généraux présentés. Une fois ces programmes mis au point, il y aurait lieu de faire des comparaisons au niveau des temps d'exécution correspondant à différentes bases B.(Il est évident que la base B de travail devrait être un paramètre de toutes les procédures)(voir à ce propos les indications dans le dernier chapitre)
Signalons aussi qu'il serait intéressant d'étendre le sujet aux nombres rationnels(les fractions)en les considérant comme des couples de nombres entiers. On pourrait alors mettre au point, de par les algorithmes dont nous disposons et relatifs aux nombres entiers, des algorithmes relatifs aux opérations sur les fractions incluant d'ailleurs la mise sous forme irréductible de ces dernières(simplification de fractions).
Comme nous le voyons, le problème étudié ainsi que les techniques de solution adoptées ouvrent les portes à des considérations bien plus larges que celles se limitant aux grands nombres entiers. Nous entrons d'ailleurs dans l'univers des manipulations symboliques.

REFERENCES

(1) G.E. COLLINS & R. LOOS, Computing Supplementum 4 Computer Algebra Symbolic and Algebraic Computation, Springer-Verlag Wien New York, 1982

(2) D.E.KNUTH, The Art of Computer Programming, Vol. 2, Addison-Wesley, 1971

(3) A. ROM, Rational Arithmetic with Integers of Indefinite Length, Pgr Quarterly Newsletter, 15-19, Summer 1969

(4) D.A.POPE, M.L. STEIN, Multiple Precision Arithmetic, Communications of the ACM, 652-654, 1960

(5) G.E. COLLINS, D.R. MUSSER, Analysis of the Pope-Stein Division Algorithm, Information Processing Letters 6, 151-155, 1977

(6) A. KARATSUBA, YU OFMAN, Dokl. Akad. SSSR 145, 293-294(1962) (English translation : Multiplication of Multidigit Numbers on automata. Sov. Phys., Dokl. 7, 595-596(1963))

(7) IBM Assembler Language Coding, Independant Study Program, 1974

(8) C.COUVREUR & J.J. QUISQUATER, An Introduction to fast generation of Large Prime Numbers, Philips J. Res. 37, 231-264, 1982