Division et racine carrée rapides

Michel Quercia, janvier 2000
quercia@cal.enst.fr

1. Division rapide

1.1. Notations

Soient A un nombre de $2n$ chiffres, B un nombre de n chiffres et Q le quotient de la division de A par B supposé tenir lui aussi sur n chiffres. On note π la base de numération et on écrit :

$$\begin{aligned}
A &= a_0\pi^{2n-1} + a_1\pi^{2n-2} + \ldots + a_{2n-1}\pi^0 \\
B &= b_0\pi^{n-1} + b_1\pi^{n-2} + \ldots + b_{n-1}\pi^0 \\
Q &= q_0\pi^{n-1} + q_1\pi^{n-2} + \ldots + q_{n-1}\pi^0
\end{aligned}$$

où $a_i, b_i, q_i \in [\![0, \pi-1]\!]$ et $b_0 \geqslant \pi/2$. Soit p entier tel que $n \leqslant 2^p$. On pose :

$$B_0 = b_0\pi^{n-1}, \qquad B_j = \sum_{2^{j-1}\leqslant i<2^j} b_i\pi^{n-i-1} \text{ pour } j \geqslant 1, \qquad Q_{j,k} = \sum_{k2^j\leqslant i<(k+1)2^j} q_i\pi^{n-i-1}$$

en convenant que les coefficients b_i et q_i d'indice supérieur ou égal à n valent zéro. On a alors :

$$B = B_0 + B_1 + \ldots + B_p \ ;$$

$$\begin{aligned}
Q &= Q_{0,0} + Q_{0,1} + \ldots + Q_{0,2^p-1} \\
&= Q_{1,0} + Q_{1,1} + \ldots + Q_{1,2^{p-1}-1} \\
&= \ldots \\
&= Q_{p-1,0} + Q_{p-1,1} \ ;
\end{aligned}$$

$$\begin{aligned}
BQ = {} & B_0Q_{0,0} + B_0Q_{0,1} + \ldots + B_0Q_{0,2^p-1} \\
& + B_1Q_{0,0} + B_1Q_{0,1} + \ldots + B_1Q_{0,2^p-1} \\
& + B_2Q_{1,0} + B_2Q_{1,1} + \ldots + B_2Q_{1,2^{p-1}-1} \\
& + \ldots \\
& + B_pQ_{p-1,0} + B_pQ_{p-1,1}.
\end{aligned}$$

On décompose ainsi le produit BQ et produits de la forme $B_jQ_{j-1,k}$ de taille $2^{j-1}\times 2^{j-1}$, la figure 1 illustre la disposition de ces produits. Le temps de calcul de BQ selon cette décomposition en utilisant un algorithme de multiplication de type Karatsuba est :

$$T_p = 2^pM_1 + 2^pM_1 + 2^{p-1}M_2 + \ldots + 2M_{2^{p-1}} = M_{2^p}\left((\tfrac{2}{3})^p + (\tfrac{2}{3})^p + (\tfrac{2}{3})^{p-1} + \ldots + (\tfrac{2}{3})^1\right) \approx 2M_{2^p}$$

où l'on a noté M_i le temps d'une multiplication de taille $i\times i$ et appliqué la règle $M_{2^k} = K3^k$. Si l'on utilise une multiplication de type FFT avec $M_{2^k} = K2^k k\lg(k)$ pour $k \geqslant 2$ on trouve :

$$T'_p = 2^{p+1}M_1 + 2^{p-1}M_2 + K\left(\sum_{i=2}^{p-1} 2^{p-i}2^i i\lg(i)\right) \approx \frac{1}{2}K2^pp^2\lg(p) = \frac{p}{2}M_{2^p}.$$

Noter que ces calculs reposent sur l'hypothèse simplificatrice que le temps d'une multiplication ne dépend que de la taille des opérandes et non de la valeur de ces opérandes.

Le principe de la division rapide consiste à calculer $q_0, \ldots, q_{n-1}$ de façon itérative en retranchant de A au fur et à mesure tous les produits $B_0Q_{0,i}$ et $B_jQ_{j-1,k}$ qu'il est possible de calculer compte tenu des chiffres de Q déjà connus. A la fin de la division on aura retranché de A tous les produits constituant BQ ce qui donne le reste de cette division. On note pour $0 \leqslant i < 2^p$:

$$\begin{aligned} \Delta P_i &= B_0Q_{0,i} + \sum_{j,k \text{ tq } (k+1)2^{j-1}=i+1} B_jQ_{j-1,k} \ ; \\ P_i &= \Delta P_0 + \ldots + \Delta P_i \ ; \\ A_i &= A - P_i. \end{aligned}$$

P_i est la somme de tous les produits dans la décomposition de BQ qui ne font intervenir que des chiffres de Q d'indice inférieur ou égal à i, donc des produits calculables dès que $q_0,\ldots,q_i$ sont connus. ΔP_i est la somme des produits apparaissant dans P_i et qui dépendent de q_i. Enfin A_i est le «reste» déduit de A connaissant seulement $q_0,\ldots,q_i$. Par construction on a $A_i \geqslant A - (q_0\pi^{n-1} + \ldots + q_i\pi^{n-i-1})B$ et il y a égalité si $i \in \{2^{p-1}-1, 2^p-1\}$. Les valeurs de P_i, ΔP_i et A_i dépendent de la valeur de Q laquelle évolue au cours de l'exécution de l'algorithme, on a renoncé à ajouter un deuxième indice traduisant cette dépendance par souci de lisibilité, le contexte doit suffire à décider quelle valeur de Q est prise en compte.

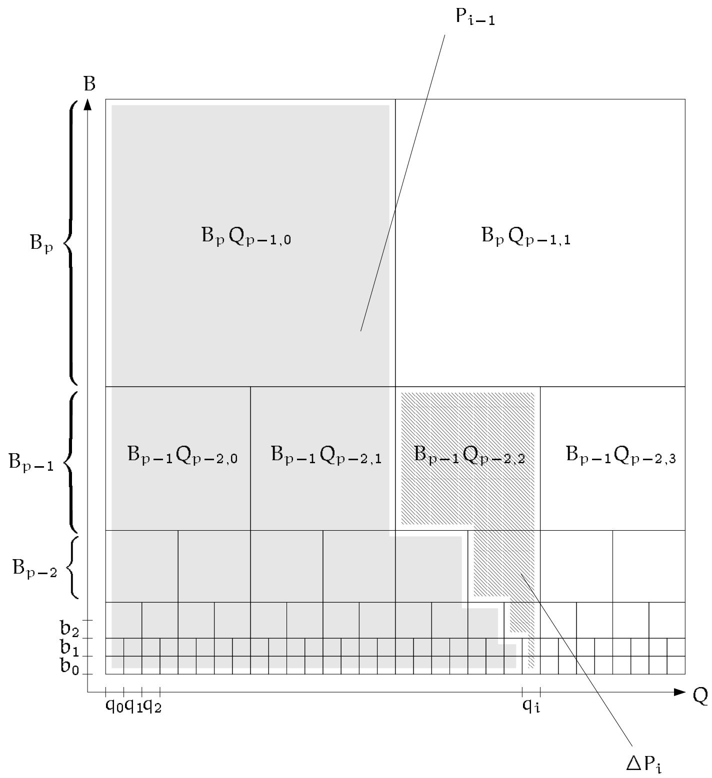

figure 1 : décomposition de BQ

1.2. Algorithme (Q)

`Q` $\longleftarrow 0$; $A_{-1} \longleftarrow A$; `exécuter` $(1\text{-}2\text{-}3)$ `pour` $i = 0, 1, \dots, 2^p - 1$.

(1) `Si` $i < n$: $q_i \longleftarrow \min(\pi - 1, \lfloor A_{i-1}/b_0\pi^{2n-i-2} \rfloor)$ `sinon` $q_i \longleftarrow 0$
`puis` $Q \longleftarrow Q + q_i\pi^{n-i-1}$ `et` $S \longleftarrow A_{i-1} - b_0 q_i \pi^{2n-i-2}$.

(2) `Exécuter` $(2.1\text{-}2.2)$ `pour` $j = 1, 2, 3, \dots$ `tant que` 2^{j-1} `divise` $i + 1$.
(2.1) $k \longleftarrow (i + 1)/2^{j-1} - 1$; $S \longleftarrow S - B_j Q_{j-1,k}$.
(2.2) `Tant que` $S < 0$ `faire` : $Q \longleftarrow Q - \max(1, \pi^{n-i-1})$, $S \longleftarrow S + (B_0 + \dots + B_j)\pi^{n-i-1}$.

(3) $A_i \longleftarrow S$.

`La division est terminée, le quotient est` Q `et le reste est` A_{2^p-1}.

1.3. Validité

Par construction on a $A_{2^p-1} \geqslant 0$ donc il suffit de démontrer que Q ne peut devenir négatif au cours de l'algorithme, que la valeur de A_i calculée en (3) est bien égale à $A - P_i$, et que $A_{2^p-1} < B$. Supposons que l'on ait à l'entrée dans la boucle (1-2-3) les propriétés suivantes :

- Q est un multiple positif de $\max(1, \pi^{n-i})$;
- $0 \leqslant A_{i-1} = A - P_{i-1}$ où P_{i-1} correspond à la valeur présente de Q.

En (1) la valeur calculée pour q_i minore le quotient $A_{i-1}/b_0\pi^{2n-i-2}$ donc $S = A_{i-1} - b_0 q_i \pi^{2n-i-2} \geqslant 0$. Si les soustractions en (2.1) ne rendent pas de résultat négatif alors en sortie de boucle Q est un multiple positif de $\max(1, \pi^{n-i-1})$ et la valeur affectée à A_i est correcte car les indices j, k tels que $B_j Q_{j-1,k}$ apparaît dans ΔP_i sont exactement ceux tels que $(k + 1)2^{j-1} = i + 1$.

Si une soustraction en (2.1) produit un résultat négatif alors on avait nécessairement $Q_{j-1,k} > 0$ donc la première décrémentation en (2.2) maintient la propriété «Q multiple positif de $\max(1, \pi^{n-i-1})$» et la correction apportée à S préserve l'invariant : «$S = A_{i-1} -$ les produits $B_u Q_{v-1,w}$ déjà calculés», compte tenu de la nouvelle valeur de Q. Il en va de même pour les décrémentations suivantes tant que $Q_{j-1,k} \geqslant 0$ et si l'on obtient $Q_{j-1,k} = 0$ alors $S = A_{i-1} \geqslant 0$ donc on quitte la boucle (2.2).

Supposons à présent $A_{2^p-1} \geqslant B$ et soit i tel que $A_i < B\pi^{n-i-1}$ et $A_\ell \geqslant B\pi^{n-\ell-1}$ pour tout $\ell \in [\![i + 1, 2^p - 1]\!]$. Un tel entier i existe car $A_{-1} = A < B\pi^n$ par hypothèse et $A_{2^p-1} \geqslant B \geqslant B\pi^{n-2^p}$. Alors pour $i < \ell < 2^p$ il n'y a pas eu de correction en (2.2) lors de l'itération d'indice ℓ car une telle correction a pour effet de retourner une valeur de S strictement inférieure à $B\pi^{n-\ell-1}$. Donc :

$$b_0\pi^{2n-\ell-2} \leqslant B\pi^{n-\ell-1} \leqslant A_\ell \leqslant A_{\ell-1} - B_0 Q_{0,\ell} = A_{\ell-1} - b_0 q_\ell \pi^{2n-\ell-2},$$

ce qui implique que q_ℓ est strictement inférieur au quotient de $A_{\ell-1}$ par $b_0\pi^{2n-\ell-2}$ et par conséquent $q_\ell = \pi - 1$ si $\ell < n$. D'où :

$$A_i \geqslant A_{2^p-1} + (q_{i+1}\pi^{n-i-2} + \dots + q_{n-1})B \geqslant B(1 + (\pi - 1)(\pi^{n-i-2} + \dots + 1)) = B\pi^{n-i-1}$$

et ceci est contradictoire avec la définition de i. Ainsi $A_{2^p-1} < B$ et la validité de l'algorithme est établie.
■

1.4. Complexité

Le temps de la division complète de A par B se décompose en deux parties : le temps cumulé de toutes les multiplications, T_p ou T'_p, et le temps des autres opérations noté $T(1)+T(2.1)+T(2.2)$ où $T(x)$ désigne le temps cumulé des opérations effectuées à l'étape (x). On peut supposer que A les A_i et S partagent la même mémoire (la valeur initiale de A étant alors perdue) donc $T(1)$ est approximativement proportionnel au nombre de chiffres de Q et est négligeable devant T_p ou T'_p pour p grand.

Le calcul de $T(2.1)$ et $T(2.2)$ fait appel à la quantité :

$$\tau_p \;=\; \sum_{i=0}^{2^p-1} \sum_{\substack{1\leqslant j\leqslant p \\ 2^{j-1}|i+1}} 2^j \;=\; \sum_{j=1}^{p} \sum_{\substack{0\leqslant i<2^p \\ 2^{j-1}|i+1}} 2^j \;=\; \sum_{j=1}^{p} 2^{p+1} \;=\; p2^{p+1}.$$

Pour $T(2.1)$ il faut compter le temps d'une soustraction sur 2^j chiffres pour tous les couples (i,j) tels que $B_jQ_{j-1,k}$ participe à ΔP_i soit $T(2.1)=O(\tau_p)=O(p2^p)$ ce qui est négligeable devant T_p ou T'_p. Pour estimer $T(2.2)$ on utilise les deux propriétés suivantes.

Proposition 1. *S'il faut décrémenter Q dans l'étape (2.2) à la suite de la soustraction de $B_jQ_{j-1,k}$ alors cette décrémentation n'affecte que les chiffres q_ℓ apparaissant dans $Q_{j-1,k}$.*

Ceci a été prouvé en 1.3.

Proposition 2. *Lors d'une phase de correction en (2.2) S devient positif ou nul après au plus deux incrémentations.*

En effet, en (2.1) on obtient $S \geqslant -B_jQ_{j-1,k} > -\pi^{n-2^{j-1}}\pi^{n-k2^{j-1}} = -\pi^{2n-(k+1)2^{j-1}} = -\pi^{2n-i-1}$ et l'on a par hypothèse $(B_0+\ldots+B_j)\pi^{n-i-1} \geqslant b_0\pi^{2n-i-2} \geqslant \frac{1}{2}\pi^{2n-i-1}$ donc $S+2(B_0+\ldots+B_j)\pi^{n-i-1} \geqslant 0$. ∎

D'après la proposition 1, une opération $Q \longleftarrow Q-\max(1,\pi^{n-i-1})$ pour un couple (i,j) donné a une complexité $O(2^j)$ et c'est aussi le cas d'une opération $S \longleftarrow S+(B_0+B_1+\ldots+B_j)\pi^{n-i-1}$ car la longueur de $B_0+B_1+\ldots+B_j$ est égale à $\min(n,2^j)$. D'après la proposition 2 on effectue au plus deux fois ces opérations à (i,j) donné donc $T(2.2)$ est dominé par τ_p et est négligeable devant T_p ou T'_p.

Proposition 3. *Soient M_{2^p} le temps d'une multiplication 2^p chiffres par 2^p chiffres et D_{2^p} le temps d'une division 2^{p+1} chiffres par 2^p chiffres selon l'algorithme (Q). On a pour p tendant vers l'infini :*

$$\text{si } M_{2^p} \sim K3^p \quad \text{alors } D_{2^p} \sim 2M_{2^p}\;;$$
$$\text{si } M_{2^p} \sim K2^p p\lg(p) \text{ alors } D_{2^p} \sim \frac{p}{2}M_{2^p}.$$

1.5. Troncature

Lorsque $n<2^p$ les dernières étapes de l'algorithme (Q) peuvent être simplifiées en ne calculant pas les produits $B_jQ_{j-1,k}$ qui sont à coup sûr nuls du fait de la convention $q_n=\ldots=q_{2^p}-1=0$. Soit $i\in[\![n,2^p-1]\!]$ et j,k tels que $B_jQ_{j-1,k}$ fasse partie de ΔP_i, c'est à dire $(k+1)2^{j-1}=i+1$. On a :

$$Q_{j-1,k} = \sum_{k2^{j-1}\leqslant\ell<(k+1)2^{j-1}} q_\ell\pi^{n-\ell-1}$$

donc $Q_{j-1,k}=0$ si $k2^{j-1}\geqslant n$. Par contraposée un produit $B_jQ_{j-1,k}$ est non à priori nul si et seulement s'il existe $i\geqslant n$ tel que $(k+1)2^{j-1}=i+1$ et $k2^{j-1}<n$ soit $k2^{j-1}<n<(k+1)2^{j-1}$. En particulier pour chaque entier $j\in[\![2,p]\!]$ il existe au plus un tel produit et on peut modifier l'algorithme (Q) pour effectuer les dernières étapes par balayage en j :

Algorithme (Q')

$Q \longleftarrow 0$; $A_{-1} \longleftarrow A$; exécuter (1-2-3) pour $i = 0, 1, \ldots, n-1$ puis exécuter (4) si $n < 2^p$.

(1) $q_i \longleftarrow \min(\pi - 1, \lfloor A_{i-1}/b_0\pi^{2n-i-2} \rfloor)$; $Q \longleftarrow Q + q_i\pi^{n-i-1}$; $S \longleftarrow A_{i-1} - b_0 q_i \pi^{2n-i-2}$.

(2) Exécuter (2.1-2.2) pour $j = 1, 2, 3, \ldots$ tant que 2^{j-1} divise $i+1$.
(2.1) $k \longleftarrow (i+1)/2^{j-1} - 1$; $S \longleftarrow S - B_j Q_{j-1,k}$.
(2.2) Tant que $S < 0$ faire : $Q \longleftarrow Q - \pi^{n-i-1}$, $S \longleftarrow S + (B_0 + \ldots + B_j)\pi^{n-i-1}$.

(3) $A_i \longleftarrow S$.

(4) $S \longleftarrow A_{n-1}$; pour $j = 2, 3, \ldots, p$ exécuter (4.1-4.2) si n n'est pas multiple de 2^{j-1}.
(4.1) $k \longleftarrow \lfloor n/2^{j-1} \rfloor$; $i \longleftarrow (k+1)2^{j-1} - 1$; $S \longleftarrow S - B_j Q_{j-1,k}$.
(4.2) Tant que $S < 0$ faire : $Q \longleftarrow Q - 1$, $S \longleftarrow S + (B_0 + \ldots + B_j)\pi^{n-i-1}$.

La division est terminée, le quotient est Q et le reste est S.

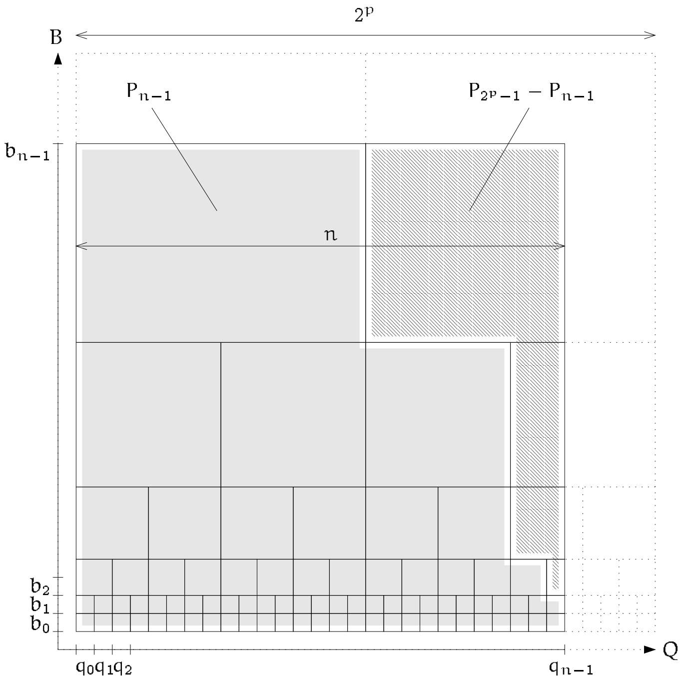

figure 2 : troncature des produits si $n < 2^p$

La complexité de cet algorithme est plus difficile à évaluer que celle de (Q) car elle fait intervenir des multiplications avec des opérandes longueurs différentes ces longueurs n'étant pas forcément des puissances de 2. En admettant que le temps d'une multiplication x chiffres par y chiffres avec $x \leqslant 2^k$ et $y \leqslant 2^k$ est dominé par le temps d'une multiplication 2^k chiffres par 2^k chiffres on a encore pour $n \leqslant 2^p$:

$D_n = O(M_{2^p})$ avec une multiplication de type Karatsuba ;

$D_n = O(pM_{2^p})$ avec une multiplication de type FFT.

1.6. Division sans reste

Le principe de la division sans reste consiste à interrompre l'algorithme (Q′) dès que le quotient est connu avec certitude, sans calculer les produits $B_j Q_{j-1,k}$ restant. Le dessin figure 3 montre que lorsque n est inférieur à 2^p les produits $B_j Q_{j-1,k}$ calculés en (4.1) sont de longueur inférieure à n donc n'influent sur les chiffres de poids fort du reste S en cours que par l'intermédiaire de retenues éventuelles. En effet, on a vu dans la proposition 2 que $B_j Q_{j-1,k} < \pi^{2n-i-1}$ pour i quelconque et on a $2n-i-1 < n$ si $i \geqslant n$. Si A_{n-1} est suffisament grand pour «absorber» ces retenues sans changer de signe alors on peut sauter l'étape (4) qui ne modifiera pas Q. Le nombre de produits retranchés de S dans cette étape (4) est au plus égal à $p-1$ d'où :

Proposition 4. *Si $n < 2^p$ et si l'on a $A_{n-1} \geqslant (p-1)\pi^{n-1}$ alors le quotient Q est correct dès la fin de la boucle* (1-2-3) *et les multiplications suivantes peuvent être omises si seul le quotient est recherché.*

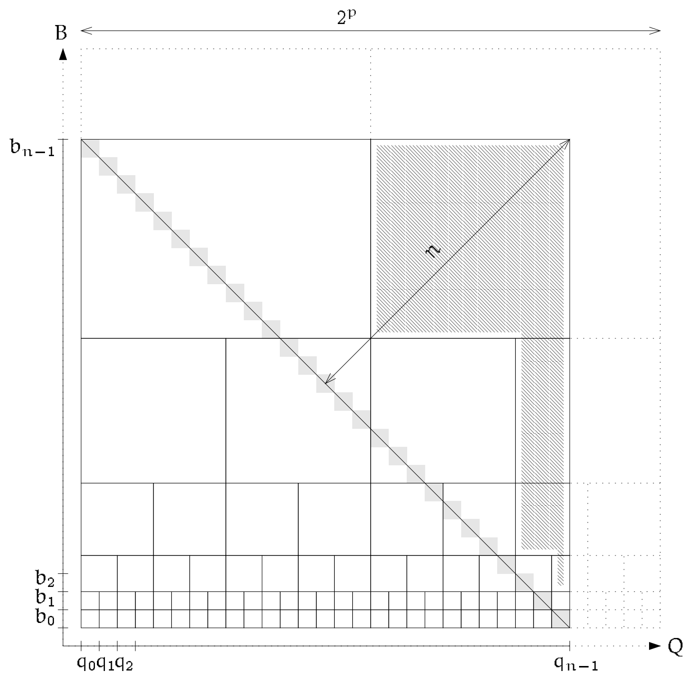

figure 3 : division sans reste

Si cette condition n'est pas remplie alors il suffit d'effectuer les corrections en (4.1-4.2) tant que $j \leqslant p$ et $S < (p-j+1)\pi^{n-1}$. Lorsque A et B sont «aléatoires», en considérant que le chiffre de poids π^{n-1} de A_{n-1} est un entier aléatoire entre 0 et $b_0 - 1$ selon une distribution uniforme alors la probabilité qu'on ait à exécuter au moins partiellement l'étape (4) est égale à $p/b_0 \leqslant 2p/\pi$ ce qui est en général très faible (sur une machine 32 bits $2p/\pi \leqslant 2^{-26}$).

Algorithme (Q″)

$Q \longleftarrow 0$; $A_{-1} \longleftarrow A$; `exécuter` (1-2-3) `pour` $i = 0, 1, \ldots, n-1$ `puis exécuter` (4) `si` $n < 2^p$.

(1) $q_i \longleftarrow \min(\pi - 1, \lfloor A_{i-1}/b_0\pi^{2n-i-2} \rfloor)$; $Q \longleftarrow Q + q_i\pi^{n-i-1}$; $S \longleftarrow A_{i-1} - b_0 q_i \pi^{2n-i-2}$.

(2) `Exécuter` (2.1-2.2) `pour` $j = 1, 2, 3, \ldots$ `tant que` 2^{j-1} `divise` $i+1$.
(2.1) $k \longleftarrow (i+1)/2^{j-1} - 1$; $S \longleftarrow S - B_j Q_{j-1,k}$.
(2.2) `Tant que` $S < 0$ `faire :` $Q \longleftarrow Q - \pi^{n-i-1}$, $S \longleftarrow S + (B_0 + \ldots + B_j)\pi^{n-i-1}$.

(3) $A_i \longleftarrow S$.

(4) $S \longleftarrow A_{n-1}$; pour $j = 2, 3, \ldots, p$ exécuter (4.0-4.1-4.2) si n n'est pas multiple de 2^{j-1}.
(4.0) si $S \geqslant (p-j+1)\pi^{n-1}$ interrompre la boucle.
(4.1) $k \longleftarrow \lfloor n/2^{j-1} \rfloor$; $i \longleftarrow (k+1)2^{j-1} - 1$; $S \longleftarrow S - B_j Q_{j-1,k}$.
(4.2) Tant que $S < 0$ faire : $Q \longleftarrow Q - 1$, $S \longleftarrow S + (B_0 + \ldots + B_j)\pi^{n-i-1}$.

La division est terminée, le quotient est Q.

Le gain obtenu avec l'algorithme (Q'') est d'autant plus intéressant que n est proche de 2^p car les multiplications évitées sont alors des «grosses» multiplications. Supposons par exemple $n = 2^p - 1$. On a alors :

$$P_{2^p-1} - P_{n-1} = \sum_{j=2}^{p} B_j Q_{j-1,k+1}$$

donc le temps de calcul économisé est :

$$S_p = M_{2^{p-1}-1} + M_{2^{p-2}-1} + \ldots + M_1 \approx \begin{cases} \frac{1}{2} M_{2^p} & \text{avec la multiplication de Karatsuba ;} \\ M_{2^p} & \text{avec une multiplication par FFT.} \end{cases}$$

Pour n quelconque on peut artificiellement le rendre proche d'une puissance de 2 en regroupant les chiffres par paquets de taille convenable. Soit par exemple n_0 le nombre entier constitué des quatre bits de poids fort dans la représentation binaire de n et $q = n_0 + 1$. Il existe k et n_1 entiers tels que $n = n_0 2^k + n_1$ avec $0 \leqslant n_1 < 2^k$ d'où :

$$\frac{n}{q} = 2^k - \frac{2^k - n_1}{q} \qquad \text{et} \qquad 8 \leqslant q \leqslant 16$$

donc la longueur de B et Q, exprimée en paquets de q chiffres, est comprise entre $\frac{7}{8}2^k$ et $2^k - 1$. On peut alors appliquer l'algorithme (Q') en opérant sur des paquets de q chiffres qui sont suffisamment petits pour que les opérations entre paquets puissent être implémentées naïvement. En pratique cela revient à remplacer π par π^q et donc la probabilité pour que le test d'évitement de (4) échoue devient inférieure à $2p/\pi^q$ soit moins de 2^{-250} sur une machine 32 bits.

1.7. Division de longueur quelconque

La division avec ou sans reste d'un nombre A de $m+n$ chiffres par un nombre B de $m \leqslant 2^p$ chiffres produisant un quotient Q de n chiffres peut être réalisée par une adaptation immédiate des algorithmes (Q) ou (Q'). Ainsi que le montre la figure 4 il est avantageux pour la division sans reste de cadrer le quotient Q en ajoutant des zéros fictifs en tête de sorte qu'il manque manque juste un chiffre à droite de Q pour atteindre une longueur multiple de 2^p cet avantage étant surtout appréciable lorsque $n < m$.

D'un point de vue macroscopique, chaque bloc de Q taille 2^{p-1} est entièrement traité (c'est à dire tous les produits faisant intervenir des chiffres de ce bloc sont calculés et retranchés de A) avant de commencer le bloc suivant, on reconnaît le schéma de division d'un nombre long par un nombre à deux chiffres. *Remarque: expliciter quo_2 ici ?* En particulier lors des décrémentations de Q aucune retenue ne peut traverser une frontière de bloc et les premiers chiffres de Q ne seront pas remis en cause par les calculs dans les blocs suivants. On dispose ainsi d'une division en ligne avec possibilité de recycler la mémoire utilisée par Q en cours de calcul.

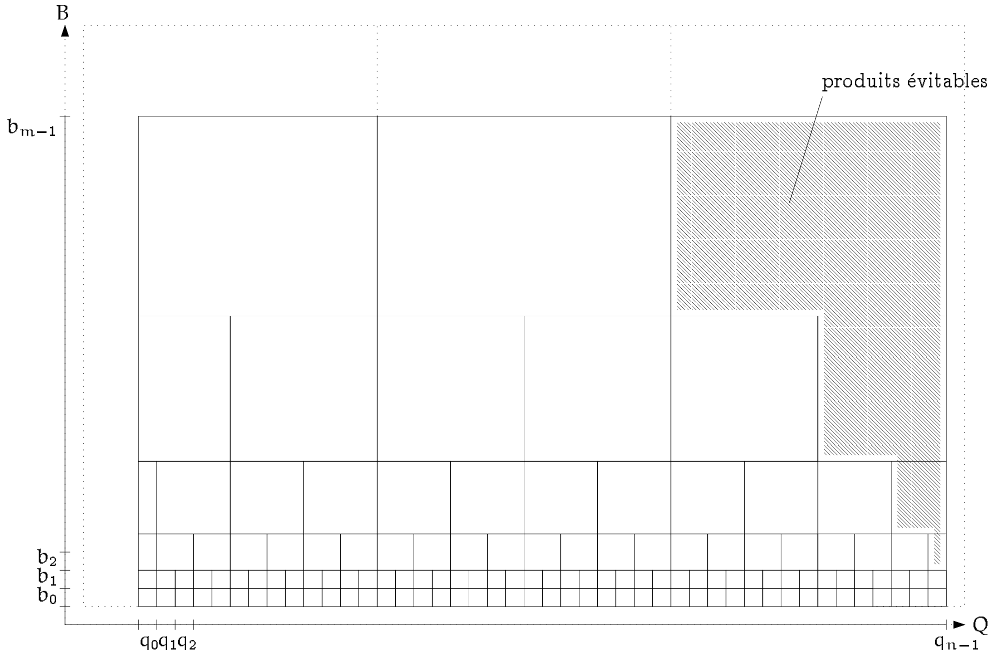

figure 4 : division longue

2. Racine carrée rapide

Soient A un nombre de $2n$ chiffres et $B = \lfloor\sqrt{A}\rfloor$ sa racine carrée entière. On a par définition :

$$B^2 \leqslant A \leqslant B^2 + 2B \qquad \text{soit} \qquad B \leqslant \frac{A}{B} \leqslant B + 2$$

donc il «suffit» pour obtenir B de calculer le quotient Q de la division de A par B, on a alors $B = Q$ ou $Q - 1$ ou $Q - 2$. Dans cette division B est inconnu mais on verra que ce n'est pas un problème, on peut amorcer le processus en calculant le premier chiffre b_0 à part et les autres chiffres seront déterminés en cours de division à partir des chiffres déjà connus. En reprenant les notations de la partie précédente on a :

$$\begin{aligned}
B^2 &= (B_0 + B_1 + \ldots + B_p)^2 \\
&= B_0^2 \\
&\quad + B_1^2 + 2B_0(B_1 + \ldots + B_p) \\
&\quad + B_2^2 + 2B_1(B_2 + \ldots + B_p) \\
&\quad + \ldots \\
&\quad + B_{p-1}^2 + 2B_{p-1}B_p \\
&\quad + B_p^2 \\
\\
&= B_0^2 + 2B_0B_{0,1} + \ldots + 2B_0B_{0,2^p-1} \\
&\quad + B_1^2 + 2B_1B_{0,2} + \ldots + 2B_1B_{0,2^p-1} \\
&\quad + B_2^2 + 2B_2B_{1,2} + \ldots + 2B_2B_{1,2^{p-2}-1} \\
&\quad + \ldots \\
&\quad + B_{p-1}^2 + 2B_{p-1}B_{p-2,2} + 2B_{p-1}B_{p-2,3} \\
&\quad + B_p^2
\end{aligned}$$

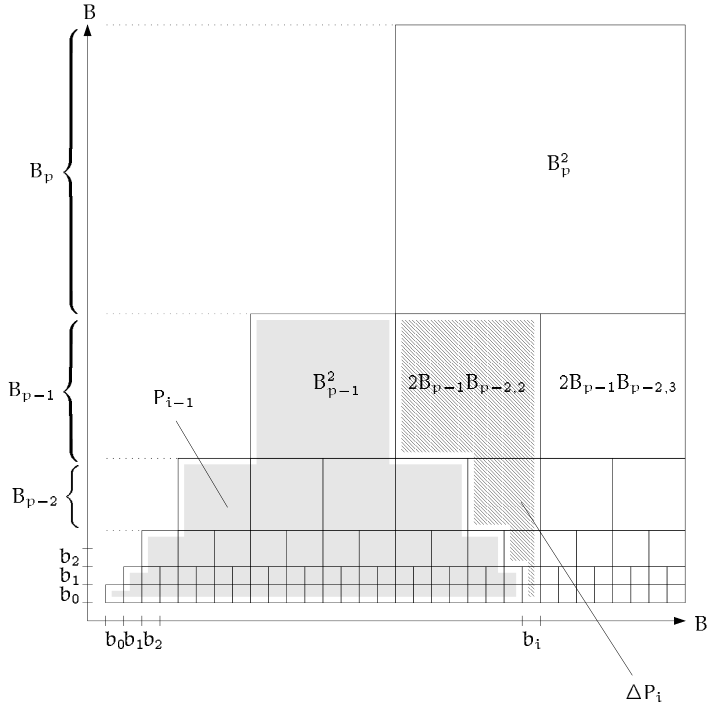

figure 5 : racine carrée

Cette décomposition est déduite de celle de BQ utilisée pour la division en remettant à plus tard le calcul des produits $B_j Q_{j-1,k}$ situés au dessus de la première bissectrice au motif qu'un tel produit n'est pas calculable quand on ne connaît que les premiers chiffres de B.

De même que pour la division on note pour $0 \leqslant i < 2^p$:

$$\Delta P_0 = B_0^2 \;;$$

$$\Delta P_i = 2B_0 B_{0,i} + \sum_{\substack{j \geqslant 1, k \geqslant 2 \\ (k+1)2^{j-1} = i+1}} 2B_j B_{j-1,k} + \begin{cases} B_j^2 & \text{si } i+1 = 2^j \\ 0 & \text{sinon ;} \end{cases} \qquad (\text{pour } i \geqslant 1)$$

$$P_i = \Delta P_0 + \ldots + \Delta P_i \;;$$

$$A_i = A - P_i.$$

et on suppose $\pi/16 \leqslant a_0 < \pi/4$ de sorte que $\pi/2 \leqslant 2b_0 < \pi$.

2.1. Algorithme (R)

$b_0 \longleftarrow \lfloor \sqrt{a_0 \pi + a_1} \rfloor$; $B \longleftarrow b_0 \pi^{n-1}$; $A_0 \longleftarrow A - B_0^2$; exécuter (1-2-3-4) pour $i = 1, \ldots, 2^p - 1$.

(1) si $i < n$: $b_i \longleftarrow \min(\pi - 1, \lfloor A_{i-1}/2b_0 \pi^{2n-i-2} \rfloor)$ sinon $b_i \longleftarrow 0$
puis $B \longleftarrow B + b_i \pi^{n-i-1}$ et $S \longleftarrow A_{i-1} - 2b_0 b_i \pi^{2n-i-2}$.

(2) Exécuter (2.1-2.2) pour $j = 1, 2, 3, \ldots$ tant que 2^{j-1} divise $i+1$ et $(i+1)/2^{j-1} > 2$.
(2.1) $k \longleftarrow (i+1)/2^{j-1} - 1$; $S \longleftarrow S - 2B_j B_{j-1,k}$.
(2.2) Tant que $S < 0$ faire : $B \longleftarrow B - \max(1, \pi^{n-i-1})$,
$S \longleftarrow S + 2(B_0 + \ldots + B_j)\pi^{n-i-1}$.

(3) Exécuter (3.1-3.2) si $i + 1 = 2^j$.
(3.1) $S \longleftarrow S - B_j^2$.
(3.2) Tant que $S < 0$ faire : $B \longleftarrow B - \max(1, \pi^{n-i-1})$,
$S \longleftarrow S + (2B_0 + \ldots + 2B_j + \pi^{n-i-1})\pi^{n-i-1}$.

(4) $\quad A_i \longleftarrow S$.

`Le calcul est terminé,` $B = \lfloor\sqrt{A}\rfloor$ `et` $A_{2^p-1} = A - B^2$.

2.2. Validité

Comme pour la division, on vérifie que A_i tel qu'il est défini en (4) est bien égal à $A - P_i$, que B reste toujours positif au cours de l'algorithme et que l'on a $A - B^2 = A_{2^p-1} \geqslant 0$. Il reste à prouver que $A_{2^p-1} \leqslant 2B$ ce qui se fait encore en suivant le même type de raisonnement que pour la division, avec une adaptation pour tenir compte du fait que le diviseur varie au cours de la division.

On note β_i la valeur de B au début de l'itération d'indice i dans la boucle (1-2-3-4). Supposons $A_{2^p-1} > 2B$ et considérons un entier i tel que $A_i \leqslant 2\beta_i\pi^{n-i-1}$ et $A_\ell > 2\beta_\ell\pi^{n-\ell-1}$ pour tout $\ell \in [\![i+1, 2^p-1]\!]$. D'après le choix de b_0 on a $a_0\pi + a_1 - b_0^2 \leqslant 2b_0$ donc $A_0 \leqslant 2\beta_0\pi^{n-1}$ ce qui assure l'existence de i. Comme pour $\ell \in [\![i+1, 2^p-1]\!]$ les corrections en (2.2) et (3.2) lors de l'itération d'indice ℓ de (1-2-3-4) rendent des valeurs de S strictement inférieures β_ℓ il n'y a pas eu de telles corrections et :

$$A_{\ell-1} - 2b_0b_\ell\pi^{2n-\ell-2} \geqslant A_\ell > 2\beta_\ell\pi^{n-\ell-1} \geqslant 2b_0\pi^{2n-\ell-2}$$

donc la division de $A_{\ell-1}$ par $2b_0\pi^{2n-\ell-2}$ a été tronquée ce qui implique $b_\ell = \pi - 1$ si $\ell < n$. On en déduit :

$$A_i \geqslant A_{2^p-1} + 2b_{i+1}\beta_{i+1}\pi^{n-i-2} + \ldots + 2b_{n-1}\beta_{n-1} > 2\beta_i\pi^{n-i-1}$$

ce qui contredit la définition de i et achève la démonstration.

2.3. Complexité

Les raisonnements sont similaires à ceux faits pour la division et on obtient :

Proposition 5. *On note M_n le temps d'une multiplication n chiffres par n chiffres, C_n le temps d'élévation au carré d'un nombre de n chiffres, R_n le temps de calcul de la racine carrée entière d'un nombre de $2n$ chiffres selon l'algorithme (R) et R'_n le temps de calcul de la même racine carrée en évitant les dernières multiplications lorsque le reste A_n est suffisament grand.*

Avec une multiplication et une élévation au carré de type Karatsuba on a :

$$R_{2^p} \sim M_{2^p} + \frac{1}{2}C_{2^p}, \qquad R'_{2^p-1} \approx \frac{5}{6}M_{2^p} + \frac{1}{6}C_{2^p}.$$

Avec une multiplication et une élévation au carré de type FFT on a :

$$R_{2^p} \approx R'_{2^p-1} \approx \frac{p}{4}M_{2^p}.$$

Remarque: Expérimentalement $C_n \approx 0.7M_n$, *(*0.6 *avec* `dlong`*). Donner l'algorithme avec* 2B.