La division euclidienne

1 La définition de la division Euclidienne

Sur l'anneau $\mathbb{Z}$ des nombres entiers, la division euclidienne est définie grâce au théorème d'existence et d'unicité suivant :

Théorème 1.1 *Soient a et b deux nombres entiers. On suppose que b est non nul. Alors, il existe un couple unique (q, r) de nombres entiers tels que :*

$$\begin{cases} a = bq + r \\ 0 \leq r < |b| \end{cases}$$

2 Algorithmes de division euclidienne

2.1 Algorithme d'Euclide

On suppose dans la suite que $a \geq 0$ et $b > 0$. Il est facile de voir qu'on peut toujours se ramener à ce cas.
L'algorithme qui est conceptuellement le plus simple pour calculer le quotient et le reste de la division euclidienne de a par b consiste à soustraire autant de fois b de a qu'il est possible, jusqu'à obtenir un reste $< b$. Voici l'algorithme correspondant :

```
division:=proc(a,b)
   local r, q, u ;

   r := a ;
   q := 0 ;
   while (r >= b)
      do
         r := r − b ;
         q := q + 1 ;
      od ;
   u := [q, r] ;
   return(u) ;
end ;
```

Hélas, on se rend vite compte que cet algorithme n'aboutit pas lorsque a est trop grand devant b, ce qui arrive très souvent dans les applications concrètes qui utilisent de grands nombres. Prendre par exemple un nombre a de 1024 bits (donc de l'ordre de 2^{1024}) et un b ayant 128 bits. Quel est, à la louche, le nombre de soustractions à faire ?

2.2 Division binaire

Nous présentons un algorithme beaucoup plus performant, basé sur l'écriture binaire des nombres, qui consiste à rechercher les digits successifs du quotient par dichotomie, et à reconstruire ce quotient par l'algorithme de Hörner. Dans un premier temps on détermine un n tel que $2^n b > a$ et on met $2^n b$ dans une mémoire auxiliaire aux.

```
divisionbin :=proc(a,b) ;
   local r,q,aux,n,u ;

   r := a ; q := 0 ; n := 0 ; aux := b ;
   while (aux <= a)
      do
         aux := 2 * aux ;
         n := n + 1 ;
      od ;
   while (n > 0)
      do
         aux := aux/2 ;
         n := n - 1 ;
         if (r < aux)
            then
               q := 2 * q ;
            else
               q := 2 * q + 1 ;
               r := r - aux ;
         fi ;
      od ;
   u := [q, r] ;
   return(u) ;
end ;
```

2.3 Preuve de l'algorithme

Nous utilisons la méthode des invariants de boucle (voir fichecrypto_100). Montrons que les conditions :

$$\begin{cases} aux = 2^n b \\ a = aux * q + r \\ 0 \leq r < aux \\ n \geq 0 \end{cases}$$

sont un invariant de boucle. Ces conditions sont bien réalisées à l'état initial.
Nous noterons aux', q', r', n' les valeurs en sortie d'un tour de boucle de aux, q, r, n. Si en entrée de boucle les conditions précédentes sont remplies alors : dans la boucle $aux' = aux/2$ et $n' = n - 1$ donc $aux' = 2^{n'} b$. De plus :

– 1^{e}cas : $r < aux'$. Dans ce cas $r' = r$, aux est divisé par 2 tandis que q est multiplié par 2. On a donc bien les conditions indiquées en sortie.
– 2^{e}cas : Si $r \geq aux'$ alors on sait que :

$$aux' \leq r < aux = 2 * aux'.$$

On a aussi $aux' = aux/2$, $q' = 2 * q + 1$, $r' = r - aux'$ et $r' < aux'$. On a donc bien les conditions attendues.

De plus n décroît strictement, donc le programme se termine avec $n = 0$. Quand $n = 0$ la variable aux contient b, q contient le quotient cherché et r contient le reste cherché.

2.4 Comparaison

Le premier algorithme (Euclide pour la division) est inutilisable car le nombre de tours de boucles contenant des opérations simples est $O(a)$ tandis que le deuxième aboutit même pour de grands nombres, son nombre de tours de boucles contenant des opérations simples est $O(\ln(a))$.