L'algorithme de Hörner

1 L'algorithme de Hörner binaire

1.1 Présentation

L'algorithme de Hörner est très connu et très simple. Nous en redonnons ici une version orientée "informatique". Nous suggérons d'autres algorithmes qui l'utilisent d'une manière plus ou moins cachée.

1.2 L'algorithme de Hörner binaire

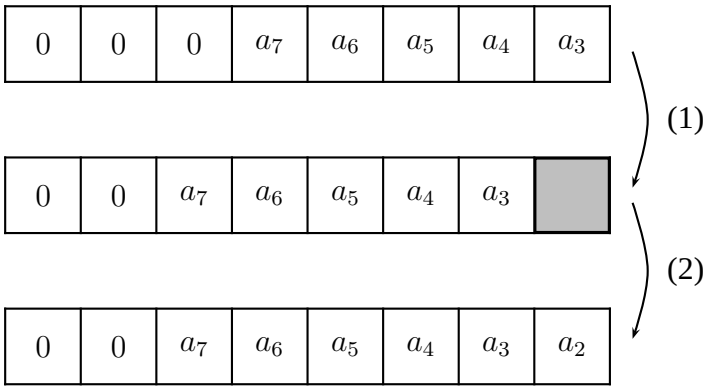

FIG. 1 – La transformation de Hörner binaire

Soit $a_7a_6a_5a_4a_3a_2a_1a_0$ où les a_i valent 0 ou 1, l'écriture binaire d'un nombre S. Ainsi :

$$S = a_7 2^7 + a_6 2^6 + a_5 2^5 + a_4 2^4 + a_3 2^3 + a_2 2^2 + a_1 2^1 + a_0.$$

L'algorithme de Hörner peut être vu comme l'entrée en mémoire d'un tel nombre conformément à la figure 1. On utilise les deux opérations :
(1) on décale les bits déjà entrés d'une position vers la gauche,
(2) on insère le bit suivant à la position la plus à droite, laissée libre par le décalage.
Notons S_i le nombre déjà entré, qui s'écrit en binaire $a_7a_6 \cdots a_{7-i}$. Alors le nombre S_{i+1} obtenu à partir de S_i par transformation de Hörner s'écrit :

$$S_{i+1} = 2S_i + a_{7-i-1}.$$

Dans cette formule la multiplication par 2 correspond au décalage à gauche des bits déjà entrés, et l'addition de a_{7-i-1} correspond à l'insertion du bit suivant à la place laissée libre. Si on décrit la suite des opérations on calcule successivement :

$$S_0 = a_7, S_1 = 2S_0 + a_6, \cdots, S_7 = 2S_6 + a_0,$$

ou encore :

$$S = S_7 = 2(2(2(2(2(2(2a_7 + a_6) + a_5) + a_4) + a_3) + a_2) + a_1) + a_0.$$

L'algorithme est décrit à la figure 2.

entrées : un tableau A de N bits
sortie : le nombre $S = \sum_{i=0}^{N-1} A[i]2^i$

début
  $S \leftarrow 0$;
  $i \leftarrow N - 1$;
  tant que $i \geq 0$ **faire**
    $S \leftarrow 2 * S + A[i]$;
    $i \leftarrow i - 1$;
  fintq ;
  retourner S ;
fin

FIG. 2 – L'algorithme de Hörner binaire

Cet algorithme effectue N tours de boucle, chaque boucle coûtant une multiplication par 2 et une addition. Ici, N représente le nombre de bits de S, c'est-à-dire la taille de S. L'algorithme coûte $O(N)$ opérations élémentaires.

1.3 Généralisation

Cet algorithme s'adapte immédiatement à toute écriture polynomiale de la forme :

$$S(X) = a_{N-1}X^{N-1} + \cdots + a_1X + a_0.$$

Il suffit en effet d'effectuer la suite d'opérations suivantes :

$$\begin{array}{lcl} S_0(X) & = & a_{N-1}, \\ S_1(X) & = & XS_0(X) + a_{N-2}, \\ \cdots & = & \cdots \\ S_{N-1}(X) & = & XS_{N-2}(X) + a_0. \end{array}$$

L'algorithme de Hörner intervient dans de nombreuses situations :

1. évaluation d'un polynôme en un point,
2. traduction binaire - décimal,
3. division euclidienne,
4. calcul d'une puissance.

En fait cette liste n'est pas limitative. En effet, dans de nombreuses situations une partie de l'algorithme consiste à reconstituer un nombre à partir de ses bits. Dans cette situation l'algorithme de Hörner s'applique.

2 Exemple

Nous nous proposons d'évaluer un polynôme t en un point x. Cette fonction existe dans les systèmes de calcul, néammoins à titre d'exemple nous la reprogrammons dans le style Maple (logiciel utilisé : xcas).

```
evaluation:=proc(t, x)
  local n, i, s;

  n :=length(t) ;
  i := n ;
  s := 0 ;
    while (i >= 1)
      do
        s := x * s + t[i] ;
        i := i - 1 ;
      od ;
  return(s) ;
end ;
```

3 Itération d'une loi associative $\oplus$ avec élément neutre e

Soit $\oplus$ une loi associative sur un ensemble G, ayant un élément neutre e. Il s'agit de calculer lorsqu'on se donne un entier $n \geq 0$ et un élément $a \in G$ la puissance :

$$S_n = a^{\oplus n} = \begin{cases} e & \text{si} \quad n = 0 \\ a \oplus a \oplus \cdots \oplus a & \text{si} \quad n > 0. \end{cases}$$

L'algorithme de Hörner s'applique :

entrées : Un tableau A de k bits tel que $n = \sum_{i=0}^{k-1} A[i]2^i$,
sortie : $S = a^{\oplus n}$.

début
  $S \leftarrow e$;
  $i \leftarrow k - 1$;
  tant que $i \geq 0$ **faire**
    $S \leftarrow S \oplus S \oplus a^{\oplus A[i]}$;
    $i \leftarrow i - 1$;
  fintq ;
  retourner S ;
fin

FIG. 3 – Itération d'une opération $\oplus$

Par exemple, si l'opération $\oplus$ est l'addition des entiers avec son élément neutre 0, et si $a = 1$, l'algorithme est exactement l'algorithme de Hörner binaire, de reconstitution de l'entier n à partir de ses bits, que nous avons décrit précédemment.

Si l'opération $\oplus$ est l'opération de multiplication dans $\mathbb{Z}/c\mathbb{Z}$:

$$a \oplus b = a.b \mod c,$$

qui a $e = 1$ pour élément neutre, l'algorithme calcule $a^n \mod c$. Cet algorithme prend alors le nom de "square and multiply" pour la raison suivante :
à chaque tour de boucle on est amené à calculer $S \oplus S \oplus a^{\oplus A[i]}$, c'est-à-dire $S^2 \mod c$ si le bit $A[i] = 0$ et dans ce cas on a une élévation au carré, et $S^2.a \mod c$ si le bit $A[i] = 1$ et dans ce cas on a une élévation au carré suivie d'une multiplication.

4 Considération des bits de bas en haut

Dans l'algorithme de Hörner donné précédemment, on a traité les bits de la "puissance" en commençant par ceux de poids forts. Peut on décrire un algorithme fondé sur un principe analogue qui traiterait d'abord les bits de poids faible ?

```
entrées : Un entier n
sortie : S = a^{⊕n}.

début
   A ← a ;
   S ← e ;
   N ← n ;
   tant que N ≥ 1 faire
      si N pair alors
         A ← A ⊕ A ;
         N ← N/2 ;
      sinon
         S ← S ⊕ A ;
         N ← N − 1 ;
      finsi ;
   fintq ;
   retourner S ;
fin ;
```

FIG. 4 – Algorithme de bas en haut

On pourra trouver une preuve de cet algorithme dans la fiche fichecrypto_100.

Remarquons qu'ici on a remplacé le tableau de bits par la recherche de la parité de N, ce qui revient théoriquement au même, mais qui souligne l'intérêt de cette méthode lorsqu'on travaille dans un langage de programmation où on n'a pas accès directement aux bits d'un entier n, mais à sa parité.