Option Informatique
Complexité

Corrigé

10 octobre 2005

1 Retour sur le calcul de puissances

1.1 Exponentiation naïve

L'unité de mesure de complexité est le coût d'une multiplication. L'exponentiation naïve, comme vu en TD, a une complexité linéaire.

L'algorithme décrit part de la valeur 1, et parcourt la représentation binaire minimale de n : pour chaque chiffre rencontré, il élève sa valeur au carré, puis si le chiffre est 1, multiplie la valeur par x. Il s'agit de la méthode de Legendre, dont la démonstration a été vue en TD. Sa complexité est de $\lambda(n) + \nu(n) - 2$ multiplications. Un majorant en est $O(\log_2(n))$.

1.2 Exponentiation rapide

1.

```
let rec puissance x n=
      if n mod 2 =0
      then  if n=0
              then 1
              else (a*a where a=puissance x (n quo 2))
      else (x*(a*a) where a=puissance x (n quo 2));;
```

2. On a facilement par récurrence $x_p = x^{2^n}$. En y substituant la définition de n, on obtient

$$x^n = x_0^{m^0} x_1^{m_1} \dots x_k^{m^k}$$

.

3. D'après le calcul de la $r.b.m$ de n, on sait que $n_i = 2n_{i+1} + m_i$, d'où $m_i = n_i \texttt{mod2}$. On peut ainsi montrer par récurrence :
$$\Pi_{j=0}^{i-1}(x_j^{m_j})x_i^{n_i} = x^n$$

4.

```
let puissiter x n =
  let y = ref x and p = ref n and res = ref 1 in
  while !p <>0 do
    if ( !p mod 2) = 1 then res :=!res * !y ;
    y :=!y * !y ; p := ( !p/2)
  done ;
 !res ; ;
```

5. Il suffit de considérer l'invariant que nous avons démontré, vrai à chaque itération de la boucle `while`. Nous avons, à la dernière itération, $\Pi_{j=0}^{k}(x_j^{m_j}) = x^n$
6. Comme `p` est divisé par 2 à chaque passage dans la boucle, qu'il est initialisé à `n` et que celle-ci s'arrête lorsque `p`=0, la boucle ne peut s'effectuer plus de $O(\log_2(\mathtt{n}))$ fois.
7. L'exponentiation rapide est adaptable puisque n'utilisant que les lois d'un anneau : il a des variantes dans l'exponentiation de matrices, aussi bien que pour la multiplication rapide, ou l'addition rapide.

2 L'algorithme de Hörner

1.
$$P(X) = \Sigma_{j=0}^{k}(a_j X^j)$$
$$n = \Sigma_{j=0}^{k}(m_j 2^j)$$

2.

```
let Horner T a =
  let l = vect_length T and res = ref 0. in
  for i = 0 to (l-1) do
    res := (a *. !res) ;
    res :=!res +. T.(i) ;
  done ;
 !res ; ;
```

3 L'algorithme de Karatsuba

1. L'algorithme naïf de multiplication de deux polynômes est quadratique.
2.
$$PQ = P_1Q_1 + X^n(P_1Q_2 + P_2Q_1) + X^{2n}P_2Q_2$$
Cette équation conduit à un algorithme diviser-pour régner dont la complexité vérifie $T(2n) = 4T(n) + O(n/2)$, soit en $O(n^2)$.
3.
$$P_1Q_2 + P_2Q_1 = (P_1 + P_2)(Q_1 + Q_2) - P_1Q_1 - P_2Q_2$$
4.
$$PQ = R_1 + (R_2 - R_1 - R_3)X^n + R_3X^{2n}$$
5. On effectue le calcul de R_1, R_2 et R_3 récursivement, et le calcul du résultat de l'équation précédente est manifestement un $O(2n)$. On en tire l'équation suivante $T(2n) = 3T(n) + O(n/2)$. D'où $T(n) = O(n^{\log_2(3)})$.

6.

```
let rec mult_poly p q =
  let N = vect_length p in
  if N = 1 then
    begin
      let res = make_vect 1 (p.(0)*q.(0)) in res
    end
  else
    begin
      match scinde_poly p with (p1,p2) - >
          match scinde_poly q with (q1,q2) - >
              let r1 = mult_poly p1 q1 and r3 = mult_poly p2 q2 in
              let r2aux = mult_poly  (add_poly p1 p2) (add_poly q1 q2) in
              let r2 = sub_poly (sub_poly r2aux r1) r3 in
              fusion r1 r2 r3 N
    end;;
```

4 L'algorithme de Strassen

1. On trouve 8 multiplications et 4 additions.
2.

$$\begin{array}{rcl} s & = & p_1 + p_2 \\ t & = & p_3 + p_4 \\ r & = & p_5 + p_4 - p_2 + p - 6 \\ u & = & p_5 + p_1 - p_3 - p_7 \end{array}$$

La méthode de Strassen demande donc 7 produits et 18 additions de matrices de taille n/2.

3.

$$T(n) = 7T(n/2) + 18(n/2)^2,\ T(1) = 1$$

4. La première inégalité s'obtient en observant que

$$T(n) \leq\ 7T(n/2)$$

ce qui est directement donné par l'équation de récurrence précédente. Pour la seconde, comme $\log_2(7)\ >\ 2$, on a $\exists p_0 \forall n > p_0$:

$$18(n/2)^2 \leq \varepsilon(n/2)^{\log_2(7)} \leq \varepsilon T(n/2)$$

On en tire $T(n) \leq (7+e)T(n/2)$, puis

$$T(2^p) \leq (7+e)^{p-p_0} T(2^{p_0})$$

En posant $C = (7+\varepsilon)^{-p_0} T(2^{p_0})$, on trouve

$$T(n) \leq C(7+\varepsilon)^p = C(7+\varepsilon)^{\log_2(n)} = Cn^{\log_2(7+\varepsilon)}$$