Option Informatique
Complexité

Sujet

10 octobre 2005

1 Retour sur le calcul de puissances

1.1 Exponentiation naïve

Avant même le TD *Calcul efficace de* x^n, nous connaissons déjà deux méthodes de calcul de puissance, basés sur les deux équations suivantes :

$$\forall n \geq 1 \; x^n = x^{n-1}x$$
$$x^n = \Pi_{i=1}^{n} x$$

Rappeller leur complexité.

Supposons maintenant que la représentation binaire minimale du nombre n est rangée dans le tableau `T`, dont le bit de poids fort est l'élément `T[0]`.

Que fait le programme CAML suivant ?

```
let calcul T x =
let l = vect_length T and res = ref 1 in
for i = 0 to (l-1) do
  res := (!res*!res);
  if T.(i) = 1 then
    res :=!res * x;
done;
!res;;
```

Quel est le nom de cet algorithme ? Rappeller sa complexité[1], en donner un majorant.

[1] On notera $\lambda(n)$ le nombre de chiffres dans la représentation binaire de n, et $\nu(n)$ le nombre de 1 dans cette représentation.

1.2 Exponentiation rapide

L'exponentiation rapide est une méthode diviser pour régner basée sur l'observation de la méthode précédente, qui permet de souligner les équations suivantes :

$$\forall k > 0 \left\{ \begin{array}{lcl} x^{2k} & = & (x^2)^k \\ x^{2k+1} & = & xx^{2k} \end{array} \right.$$

1. Donner une implémentation récursive de cet algorithme. On demande :
   `puissance : int → int → int = <fun>`
2. Notons $m_0, m_1, \ldots m_k$ les coefficients de la représentation binaire de n, avec $n - E(n/2) = m_0$. On a :

   $$n = m_0 2^0 + m_1 2^1 + \ldots + m_k 2^k$$

   Soit x_p la suite définie par $x_0 = x$ et $x_{p+1} = (x_p)^2$.
   Exprimez x_p en fonction de x, puis x^n en fonction des termes de $x_0 \ldots x_k$ et de $m_0 \ldots m_k$.
3. Soit n_i la suite définie par $n_0 = n$ et $n_{i+1} = n_i$`div2`. Montrer qu'on sait calculer les m_i définis ci-dessus à l'aide des n_i. En déduire un invariant (une équation valable quelle que soit i) sur $\Pi_{j=0}^{i-1} x_j^{m_j}, x_i, n_i$ et x^n.
4. En déduire une implémentation itérative de l'algorithme d'exponentiation rapide, indicée par i, et conservant l'invariant trouvé ci-dessus. On demande :
   `puissiter : int → int → int = <fun>`
5. Montrer la correction de cet algorithme.
6. Evaluer sa complexité.
7. Quelles propriétés utilise cet algorithme ? Serait-il adaptable ?

2 L'algorithme de Hörner

1. Soit P un polynôme de degré k en X de coefficients $a_0 \ldots a_k$. Ecrivez l'expression de P sous la forme d'une somme de puissances de X.
   Soit n un entier quelconque et $m_0 \ldots m_k$ les coefficients de sa *r.b.m.* Ecrivez l'expression de sa *r.b.m* sous la forme d'une somme de puissances de 2.
2. Supposez alors donné un tableau `T` des coefficients de P, avec `T.(0)` le coefficient dominant de P. En vous inspirant de 1.1, donner une implémentation itérative d'un algorithme de calcul de $P(a)$ pour a fixé, de complexité maximale $O(\log_2(n))$. On demande :
   `Horner : float vect → float → float = <fun>`

3 Multiplication de polynômes : L'algorithme de Karatsuba

Soient P et Q deux polynomes de degré $\leq 2n$, que l'on représentera, comme usuellement, à l'aide des tableaux de leurs coefficients.
On considère P_1, P_2, Q_1, Q_2 tels que :

$$\begin{array}{lcl} P & = & P_1 + X^n P_2 \\ Q & = & Q_1 + X^n Q_2 \end{array}$$

1. Rappeller la complexité de l'algorithme naïf de multiplication de deux polynomes.
2. Exprimer PQ à l'aide de P_1, P_2, Q_1, Q_2. Quelle serait la complexité d'un algorithme diviser pour régner sur la base de cette équation ?
3. Exprimer $P_1Q_2 + P_2Q_1$ à l'aide de $(P_1 + P_2)$, $(Q_1 + Q_2)$, P_1Q_1 et P_2Q_2.

4. Soit $R_1 = P_1Q_1$, $R_2 = (P_1 + P_2)(Q_1 + Q_2)$ et $R_3 = P_2Q_2$. Exprimer PQ à l'aide des seuls R_1, R_2, R_3 et X.
5. En tirer la complexité du calcul de deux polynômes.
6. Donner une implémentation de cet algorithme. On pourra se servir des fonctions auxiliaires qui suivent.

Fonctions auxiliaires

- `scinde`$_{poly}$ recoit comme paramètre un polynome `p` de degré `N` et renvoie comme résultat les deux polynomes `p1` et `p2` de degrés strictement inférieurs à `N/2` tels que

$$\texttt{p}(X) = \texttt{p1}(X) + X^{\texttt{N}/2}\texttt{p2}(X)$$

.

- `add`$_{poly}$ et `sub`$_{poly}$ effectuent la somme et la différence entre deux polynomes.
- `fusion` recoit les trois polynomes `r1`, `r2` et `r3` de degré au plus `N-1` et l'entier `N` et renvoie le polynôme

$$\texttt{r1}(X) + \texttt{r2}(X)X^{\texttt{N}/2} + \texttt{r3}(X)X^N$$

.

```
let scinde_poly p =
  let N = vect_length p in
  (sub_vect p 0 (N/2), sub_vect p (N/2) (N/2));;
```

```
let fusion r1 r2 r3 N =
  let p = make_vect (2*N) 0 in
  for i = 0 to N-2 do
    p.(i)< - r1.(i)
  done;
  for i = 0 to N-2 do
    p.(N+i) < - r3.(i)
  done;
  for i = 0 to (N-2) do
    p.(N/2+i)< - p.(N/2+i)+r2.(i)
  done;
  p;;
```

```
let add_poly p q =
   let N = vect_length p in
   let somme  = make_vect N 0 in
   for i = 0 to N-1 do
     somme.(i)< - p.(i)+q(i)
   done;
   somme;;
```

```
let sub_poly p q =
  let N = vect_length p in
  let diff  = make_vect N 0 in
  for i = 0 to N-1 do
    diff.(i)< - p.(i)-q(i)
  done;
  diff;;
```

4 Multiplication de matrices : l'algorithme de Strassen

On va tenter d'appliquer une stratégie diviser pour régner à une multiplication de matrices. Soient A et B deux matrices $n \times n$. Nous cherchons à calculer le produit $C = AB$. On suppose (quitte à s'y ramener) que n est une puissance exacte de 2, et on divise A, B et C en quatre matrices $n/2 \times n/2$.

$$\begin{bmatrix} r & s \\ t & u \end{bmatrix} = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} e & g \\ f & h \end{bmatrix}$$

On en tire ainsi les quatre équations suivantes :

$$\begin{aligned} r &= ae + bf \\ s &= ag + bh \\ t &= ce + df \\ u &= eg + dh \end{aligned}$$

1. Comptez le nombre de multiplications et de sommations à effectuer pour ce calcul par la méthode naïve.

   On note ensuite :

$$\begin{aligned} p_1 &= a(g-h) \\ p_2 &= (a+b)h \\ p_3 &= (c+d)e \\ p_4 &= d(f-e) \end{aligned} \qquad \begin{aligned} p_5 &= (a+d)(e+h) \\ p_6 &= (b-d)(f+h) \\ p_7 &= (a-c)(e+g) \end{aligned}$$

2. Exprimez s à l'aide de p_1, p_2, t à l'aide de p_3, p_4, r à l'aide de $p_5 + p_4$ et $p_2 - p_6$ et enfin u à l'aide de $p_5 + p_1$ et $p_3 + p7$. En déduire le nombre de sommes et de multiplications nécessaires à la multiplication de deux matrices à l'aide de la méthode de Strassen.
3. Ecrire l'équation de récurrence que vérifie la complexité de l'algorithme de Strassen.
4. Montrer qu'il existe une constante $C > 0$ telle que pour tout $\varepsilon > 0$ fixé :

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

5. On s'entraînera à donner une implémentation de cet algorithme. Votre programme acceptera les matrices ayant une taille qui ne soit pas une puissance de 2, en remplissant les lignes et colonnes ajoutées par des zéros (padding).