TP Caml 6: Multiplication de polynômes

lionel.rieg@ens-lyon.fr

13 & 20 janvier 2009

Dans ce TP, nous allons nous intéresser à la multiplication de polynômes, qui peut s'adapter à celle des grands entiers. Pour permettre aux polynômes d'avoir des degrés arbitraires, on va utiliser un type « extensible », les listes, qui contiendront les coefficients du polynôme et qu'on va ordonner par degrés croissants. Le type des éléments dépendra des coefficients des polynômes, int, float voire num. On prend ici le type int pour simplifier (sauf indication contraire) mais tout resterait valable avec float ou num.

► **Question 1** Écrire les polynômes 0, X et $4X^3 - 2X^2 + 1$ avec cette représentation. Programmer ensuite le calcul du degré.

1 Multiplication naïve

On s'intéresse ici à la multiplication selon la formule théorique : $P \cdot Q = \sum_{n\geqslant 0} \left(\sum_{j+k=n} p_j \cdot q_k\right) X^n = \sum_{j\geqslant 0} p_j X^j \cdot Q$.

► **Question 2** Écrire des fonctions d'addition et de soustraction de polynômes ainsi que la multiplication par un monôme

```
monomial_mult : int list → int → int → int list
```

telle que monomial_mult p a n réalise la multiplication du polynôme p par le monôme aX^n.

► **Question 3** Écrire l'algorithme de multiplication naïve. Quelle est sa complexité ?

2 Algorithme de Karatsuba

Pour diminuer la complexité de la multiplication de polynômes, on va utiliser un paradigme très classique de programmation : *diviser pour régner*. Pour cela, on va décomposer les polynômes à multiplier P et Q de degrés strictement inférieurs à $2n$ en

$$P = P_1 + P_2 \cdot X^n \qquad \text{et} \qquad Q = Q_1 + Q_2 \cdot X^n$$

avec les degrés de P_1, P_2, Q_1 et Q_2 strictement inférieurs à n.

► **Question 4** Écrire une formule qui réduit la multiplication des polynômes P et Q de degrés strictement inférieurs à $2n$ en multiplications de polynômes de degrés strictement inférieurs à n.

► **Question 5** Programmer un algorithme récursif de multiplication qui utilise la formule précédente. Quelle est sa complexité ?

On peut raffiner cette méthode avec la remarque suivante de Karatsuba : le terme intermédiaire de $P \cdot Q$ s'écrit

$$P_1 \cdot Q_2 + P_2 \cdot Q_1 = (P_1 + P_2) \cdot (Q_1 + Q_2) - P_1 Q_1 - P_2 Q_2$$

On échange donc deux multiplications et une addition contre une multiplication et quatre additions.

► **Question 6** Écrire une fonction qui réalise la multiplication de polynômes à la Karatsuba.

► **Question 7** Trouver la formule de récurrence qui définit la complexité de la multiplication de Karatsuba. Quelle est sa solution ?

3 Algorithme de Schönhage-Strassen

Dans cette partie, on va étudier une technique qui est asymptotiquement plus rapide que celle de Karatsuba et qui se base sur la transformée de Fourier rapide. Elle n'est utilisée que pour les grandes valeurs de n, qui doit être une puissance de 2. Le principe de cette technique est de remarquer que l'évaluation d'un produit ou d'une somme de polynômes se fait facilement à partir des valeurs des polynômes à multiplier ou sommer. De plus, l'interpolation de Lagrange permet de revenir des valeurs aux coefficients.

Cette idée peut se voir comme un changement de représentation : on passe d'une représentation par coefficients à une représentation par valeurs. C'est un compromis car certaines opérations comme déterminer le degré ou la valuation, triviales avec les coefficients, ne le sont plus avec des valeurs. Ce qui rend cette idée viable, c'est le fait que l'évaluation et l'interpolation sont des opérations pas trop coûteuses (par la transformée de Fourier) comme nous allons le voir.

3.1 (Multi-)Évaluation

L'évaluation va se faire dans $\mathbb{C}$, choix qui se justifiera dans la suite. Pour représenter les nombres complexes, on prend une représentation cartésienne avec le type suivant :

```
type complex = { Re : float ; Im : float };;
```

► **Question 8** Écrire les fonctions d'arithmétique usuelle sur les complexes : addition, soustraction, multiplication, division et conjugaison. Programmer aussi les complexes 0 et 1 ainsi que des fonctions de conversion entre entiers et complexes.

► **Question 9** Écrire un algorithme d'évaluation de polynômes. Il sera de type

```
eval : int list → complex → complex
```

Quelle est sa complexité en nombres d'additions et de multiplications ?

L'évaluation d'un polynôme peut se faire de façon incrémentale par le schéma de Horner qui consiste à remarquer que le polynôme $P = a_nX^n + a_{n-1}X^{n-1} + \cdots + a_1X + a_0$ peut se récrire $P = (\ldots((a_nX) + a_{n-1})X + \cdots + a_1)X + a_0$. Il ne faut donc que $2n$ opérations.

► **Question 10** Améliorer l'algorithme précédent à l'aide du schéma de Horner. Pourrait-on faire mieux que $\Theta(n)$ opérations ?

Maintenant que l'on sait évaluer un polynôme en un point, on peut utiliser cet algorithme n fois sur P et Q pour obtenir les évaluations qui permettront de reconstruire le produit $P \cdot Q$ par interpolation. Malheureusement, on obtient ainsi un algorithme en $\Theta(n^2)$, qui n'est pas meilleur que la méthode naïve. Il nous faut donc économiser des calculs lors des évaluations futures en réutilisant ceux effectués auparavant, et ce par un choix judicieux des points d'évaluation. Par exemple, si $b = -a$, alors les termes de degrés pairs de $P(a)$ et $P(b)$ sont les mêmes et ceux de degrés impairs sont opposés. En fin de compte, les candidats naturels comme points d'évaluation sont les racines (complexes) n^{e} de l'unité. Dans la suite, on note ω_n une racine primitive n^{e} de l'unité, par exemple $e^{\frac{2i\pi}{n}}$.

► **Question 11** Écrire une fonction de découpage d'un polynôme en ses parties paire et impaire : son résultat sur un polynôme P sera le couple (P_{pair}, P_{impair}) tel que $P(X) = P_{pair}(X^2) + X \cdot P_{impair}(X^2)$.

Évaluer P en ω_n^k et $-\omega_n^k = \omega_n^{k+\frac{n}{2}}$ se ramène à évaluer P_{pair} et P_{impair} en ω_n^{2k} plus quelques multiplications et additions/soustractions (si l'on suppose déjà connus les ω_n^k pour $k \in [\![0, n-1]\!]$). On est donc naturellement porté à utiliser une solution récursive et cela justifie d'utiliser des racines de l'unité car elles forment un groupe multiplicatif : on connaît déjà les $\omega_n^{2k} = \omega_{n/2}^k$ car ce sont des ω_n^k.

► **Question 12** Écrire une fonction qui prend en argument les ω_n^k pour $k \in [\![0, n-1]\!]$ et les valeurs de P_{pair} et P_{impair} en les $\omega_n^{2k} = \omega_{n/2}^k$ pour $k \in \left[\!\left[0, \frac{n}{2} - 1\right]\!\right]$ et construit les valeurs de P en les ω_n^k pour $k \in [\![0, n-1]\!]$.

► **Question 13** Écrire un algorithme récursif qui réalise l'évaluation d'un polynôme (à coefficients complexes) en les n puissances successives d'une racine primitive n^{e} de l'unité ω_n qui seront données en argument. (c'est la transformée de Fourier du polynôme)

► **Question 14** Écrire un algorithme qui calcule la liste des puissances successives ordonnées par ordre croissant d'un nombre complexe donné en argument. Il renverra la liste $[1; \omega; \omega^2; \ldots; \omega^{n-1}]$ si on lui passe ω et n. Il sera donc de type

```
compute_powers : complex → int → complex list
```

A-t-on intérêt ici à utiliser l'exponentiation rapide ?

3.2 Interpolation

► **Question 15** Expliquer pourquoi l'interpolation est une opération aussi simple que l'évaluation (penser que l'évaluation est une opération linéaire et possède donc une matrice associée).

► **Question 16** Calculer le résultat de la composée d'une évaluation en les ω_n^k suivie d'une autre en les ω_n^{-k} ($k \in [\![0, n-1]\!]$) pour un polynôme de degré inférieur à n. On considérera la première évaluation comme les coefficients du polynôme pour la seconde.

► **Question 17** À l'aide de la question précédente, écrire un algorithme d'interpolation de polynômes en les ω_n^k pour $k \in [\![0, n-1]\!]$.

► **Question 18** Combiner toutes les fonctions écrites jusqu'à présent et écrire la multiplication par FFT.

► **Question 19** Quelle est la complexité de l'algorithme de multiplication de Schönhage-Strassen ?