Option Informatique

TP n°4 : Algorithme de Karatsuba
pour le produit de polynômes ou de grands entiers

Dans cette feuille, on sera amené à déterminer diverses complexités ; il sera donc bon d'avoir sous les yeux la fiche de synthèse sur les sommes et récurrences usuelles (à moins que vous ne les connaissiez déjà !).

Partie théorique

On souhaite additionner et multiplier deux polynômes P et Q, de degrés semblables, à coefficients dans un anneau. Quitte à compléter avec des coefficients nuls, on peut supposer qu'ils s'écrivent sous la forme $P = a_n X^n + \cdots + a_1 X + a_0$ et $Q = b_n X^n + \cdots + b_1 X + b_0$. On considèrera pour simplifier qu'une addition ou une multiplication de scalaires se fait en temps constant [1].

Exercice 1. *Addition des polynômes*
Donner l'algorithme permettant de calculer $P + Q$ qu'on obtient en écrivant la définition de la somme. Quelle est sa complexité ? Pourrait-on imaginer faire mieux (asymptotiquement) ? Pourquoi ?

Exercice 2. *Algorithme direct pour la multiplication*
Donner l'algorithme permettant de calculer $P \times Q$ qu'on obtient en écrivant la définition du produit. Quelle est sa complexité ? *On va voir dans la suite qu'on peut mieux...*

Exercice 3. *Une stratégie "diviser pour régner" trop naïve*
Les stratégies "diviser pour régner" que nous étudierons procèdent en trois étapes. Pour un problème de taille n :

- on découpe le problème en a sous-problèmes de taille $\approx \frac{n}{2}$;
- on résout récursivement les sous-problèmes ;
- on recombine les solutions des sous-problèmes pour en déduire une solution du problème de départ.

On a déjà vu, pour $a = 1$, les exemples de la recherche dichotomique ou de l'exponentiation rapide.

Voici ce qu'une approche "diviser pour régner" naïve conduit naturellement à écrire ici : on note $P = AX^{\lceil \frac{n}{2} \rceil} + B$ et $Q = CX^{\lceil \frac{n}{2} \rceil} + D$, où A, B, C, D sont des polynômes de degré (au plus) $\lceil \frac{n}{2} \rceil$. La distributivité du produit sur la somme indique qu'on a $PQ = (AC)X^N + (AD + BC)X^{\frac{N}{2}} + BD$ où $N = 2\lceil \frac{n}{2} \rceil = n$ ou $n+1$. On peut donc écrire un algorithme récursif qui, pour calculer $P \times Q$, calcule les 4 produits (qui portent sur les polynômes de taille $\approx \frac{n}{2}$) AC, AD, BC, BD récursivement, puis calcule les trois sommes de polynômes restantes (et le produit par $X^{\frac{N}{2}}$ mais qui correspond juste à un décalage des coefficients).

1. Quels seraient les cas d'arrêt d'un tel algorithme ?
2. En notant c_n la complexité d'un tel algorithme, donner une relation de récurrence vérifiée par c_n.
3. En déduire l'ordre de grandeur de c_n.
4. Pourquoi l'approche explicitée ici est-elle trop naïve ?

1. Ce qui est vrai par exemple si les coefficients sont des entiers de Caml ou des flottants, qui sont de taille fixe.

Exercice 4. *L'algorithme de Karatsuba pour la multiplication des polynômes*

L'**algorithme de Karatsuba** reprend l'idée précédente en l'optimisant afin d'en améliorer la complexité. En effet on a $(AD+BC) = (A+B)(C+D) - (AC+BD)$; et donc $PQ = (AC)X^N + ((A+B)(C+D) - (AC+BD))X^{\frac{N}{2}} + BD$.

Voilà qui augmente le nombre de sommes à calculer, mais celles-ci sont peu couteuses ! D'un autre côté, nous n'avons plus que trois produits à calculer : AC, BD et $(A+B)(C+D)$, et ceux-ci portent toujours sur des polynômes de taille $\approx \frac{n}{2}$.

1. En notant c_n la complexité d'un tel algorithme, donner une relation de récurrence vérifiée par c_n.
2. En déduire l'ordre de grandeur de c_n.

Exercice 5. *L'algorithme de Karatsuba pour la multiplication des grands entiers*

Proposer une adaptation de l'algorithme de Karatsuba permettant de multiplier de grands entiers de taille proche de façon plus efficace que la méthode naturelle (on rappelle que celle-ci nécessite $\Theta\big(\lg(a)\lg(b)\big)$ additions bit-à-bit pour multiplier a et b).

Implémentation

Dans la suite on se limite au cas du produit de deux polynômes à coefficients entiers. Un polynôme de degré n est implémenté par un tableau de longueur $n+1$.

```
type polynome = int array;;
```

On considèrera que céer un tableau de longueur n est de complexité $O(n)$ et accéder/modifier une case d'un tableau en $O(1)$, ce qui ne change donc pas les complexités précédentes.

Exercice 6. En n'hésitant pas à recopier les tableaux autant de fois que nécessaire :

1. Écrire des fonctions `add, sub : polynome → polynome → polynome` pour l'addition et la soustraction.
2. Écrire une fonction `prod : polynome → polynome → polynome` implémentant un calcul de la multiplication en utilisant l'algorithme de Karatsuba.