Université Bordeaux 1 — **Algèbre et calcul formel – Master**
Mathématiques — Année 2018–2019

FEUILLE D'EXERCICES n° 3

Travail sur papier et sur machine

Diviser pour régner : Karatsuba, Tri fusion

Nous commençons la séance par un exercice théorique : il s'agit de démontrer le lemme que nous avons utiliser pour évaluer la complexité des algorithmes "Karatsuba" et "tri fusion".

Ensuite, nous passons sur machine. L'exercice 2 prépare l'exercice 3, lequel propose la programmation de l'algorithme de Karatsuba. Bien sûr, sage contient des fonctions pour manipuler des polynômes, mais ici, l'exercice consiste à travailler sur des polynômes définis par des listes. Ainsi, Karatsuba prendra en entrée deux listes de longueur n (où n désigne une puissance de 2), qui représenteront deux polynômes P et Q de degré strictement inférieur à n, et rendra en sortie une liste de longueur $2n$ qui représentera le produit PQ. On rappelle que si l est une liste de longueur n. La liste l représente le polynôme

$$\sum_{i=0}^{n-1} l[i]x^i.$$

Ainsi, le travail de programmation se fera sur des listes. Par contre, à l'extérieur des programmes, il sera commode de savoir facilement transformer une liste en polynôme, ne serait-ce que pour vérifier ses résultats. C'est pourquoi l'exercice 2 présente quelques commandes dédiées aux listes et polynômes.

Exercice 1 – [Le lemme]
On démontre ici le lemme 4.2.2 du cours.

Soit T une fonction de $\mathbb{R}_+$ dans $\mathbb{R}_+$ qui vérifie les propriétés suivantes.

(1) Il existe des réels $a > 0$ et $b > 1$ tels que $T(x) = aT\left(\frac{x}{b}\right) + O(x)$.

(2) Pour tout $x \leqslant 1$, $T(x) \leqslant 1$.

1) la condition (1) signifie qu'il existe $M > 0$ tel que $T(x) \leqslant aT\left(\frac{x}{b}\right) + Mx$ pour tout x. Soit $k = \lfloor \log_b x \rfloor + 1$. Montrer que pour tout x,

$$T(x) \leqslant a^k + Mx\left(1 + \frac{a}{b} + \cdots + \left(\frac{a}{b}\right)^{k-1}\right)$$

2) En déduire que

$$T(x) = \begin{cases} O\left(x^{\log_b(a)}\right) \text{ si } a > b \\ O\left(x \log x\right) \text{ si } a = b \\ O\left(x\right) \text{ si } a < b \end{cases}$$

Exercice 2 – [Listes et polynômes]
Pour manipuler des polynômes sur sage, il faut d'abord définir l'anneau de polynômes correspondant. Pour définir $\mathbb{Q}[x]$ par exemple, exécuter la commande suivante.

```
Qx.<x>=PolynomialRing(QQ)
```

Ici, `Qx` est le nom que l'utilisateur donne à $\mathbb{Q}[x]$. On peut mettre `toto`, `chat` ou n'importe quoi d'autre à la place. Le suffixe `<x>` indique le nom de la variable. À partir de cette commande, `x` sera le x de $\mathbb{Q}[x]$. Le terme `QQ` est défini dans le logiciel : c'est $\mathbb{Q}$.

La commande

```
Zx.<x>=PolynomialRing(ZZ)
```

définit de même $\mathbb{Z}[x]$. On utilise aussi les commandes `RR` pour $\mathbb{R}$, `CC` pour $\mathbb{C}$, `GF(5)` pour $\mathbb{F}_5$.

Exécuter les commandes suivantes.

```
p=9*x^3+2*x-1
parent(p)
```

`p` est donc un élément de $\mathbb{Z}[x]$.

Essayer aussi

```
factor(p)
p.coefficients()
p(1/3)
p[1]
p.degree()
[p[i] for i in range(4)]
p[9]
```

On remarque que l'on obtient les coefficients de `p` par les commandes `p[i]`, et que si $i > 3$, la commande `p[i]` donne 0.

Par contre, la commande

```
p[0:2]
```

ne fonctionne pas.

Il n'est donc pas forcément utile de transformer p en liste si l'on utilise des commandes communes aux listes et aux polynômes. Si le programme `Karatsuba` utilise de telles commandes, alors il acceptera indifféremment listes ou polynômes en entrées.

Si l'on veut tout de même faire cette transformation, on peut utiliser la commande

```
l=list(p)
```

`l` est une liste de longueur 4.

Réciproquement, on peut définir un polynôme à partir d'une liste. Exécuter les commandes suivantes.

```
l=[0,1,2,3,4,5,0,0]
q=Zx(l);q
```

Bien sûr, la commande `p*q` donne le produit de p par q.
Pour choisir un polynôme au hasard dans $\mathbb{Z}[x]$ ˜ :

```
Zx.random_element(100,-100,100)
```

donne un polynôme de degré 100 dont les coefficients sont choisis au hasard dans $[[-100, 100]]$. Pour obtenir un polynôme de degré inférieur ou égal à 100, on peut utiliser la commande :

```
Zx.random_element((0,100),-100,100)
```

Si l'on veut choisir un polynôme au hasard dans $\mathbb{Q}[x]$, en imposant des bornes aux numérateurs et dénominateurs :

```
Qx.<x>=PolynomialRing(QQ)
Qx.random_element((0,10),num_bound=10,den_bound=10)
```

Exercice 3 – [KARATSUBA]
Rappelons le principe. On désire calculer le produit de deux polynômes $P, Q \in R[X]$ de degrés $< n$, où R est un anneau commutatif. L'approche naïve a une complexité algébrique en $O(n^2)$. Une façon d'améliorer ce résultat est la suivante. Considérons nos polynômes comme des polynômes de degrés strictement inférieurs à n où n est une puissance de 2. Supposons $n > 1$. On pose $m = n/2$ et on écrit

$$P = X^m P_1 + P_0 \quad \text{et} \quad Q = X^m Q_1 + Q_0,$$

où P_1, P_0, Q_1 et Q_0 sont des polynômes de degré(s) $< m$. On a alors

$$\begin{aligned} PQ &= X^n P_1 Q_1 + X^m (P_1 Q_0 + P_0 Q_1) + P_0 Q_0 \\ &= X^n P_1 Q_1 + X^m \left((P_1 + P_0)(Q_1 + Q_0) - P_1 Q_1 - P_0 Q_0 \right) + P_0 Q_0, \end{aligned}$$

de telle sorte que nous avons juste à calculer trois produits

$$A = P_1 Q_1,\ B = P_0 Q_0 \text{ et } C = (P_1 + P_0)(Q_1 + Q_0)$$

de polynômes de degrés strictement inférieurs à m. On utilise alors cette idée de façon récursive. Cela donne un algorithme dont la complexité algébrique est en $O(n^{\log 3/\log 2})$.

Soit donc n une puissance de 2. On considère deux polynômes P et Q de degrés $< n$.

1) Écrire une procédure récursive `Karatsuba(`P, Q, n`)` utilisant le principe rappelé ci-dessus. Cette procédure prendra en entrée des listes P et Q, et un entier n qui sera une puissance de 2. Les listes P et Q seront de longueur n. Ils représenteront donc des polynômes de degré strictement inférieur à n (pour représenter un tel polynôme par une liste de longueur n, on complète éventuellement la liste par des 0). La procédure renverra en sortie une liste de longueur $2n$ correspondant à PQ.

Dans cette procédure, on s'appliquera à n'utiliser que des opérations sur les listes : on n'y introduira aucun polynôme.

2) Tester cette procédure sur des polynômes de petits degrés, puis sur de gros polynômes que l'on choisira de façon aléatoire.

Exercice 4 – [Tri fusion]

1) Écrire une procédure permettant de générer une liste l dont les éléments sont les entiers 1, 2, $\dots n$, placés dans un ordre aléatoire.

2) Programmer le tri fusion et le tester sur l. On pourra

- écrire une procédure `division` coupant une liste de longueur n en deux tableaux de longueurs respectives $\lceil n/2 \rceil$ et $\lfloor n/2 \rfloor$ dont la concaténation est la liste initiale ;
- écrire une procédure `fusion` prenant en argument deux listes d'entiers supposés triés et qui renvoie une liste triée contenant tous les éléments des deux tableaux initiaux (voir ci-dessus) ;
- écrire une procédure `tri` à l'aide des deux procédures précédentes.

3) Comparer avec le tri naïf, dit tri sélection, qui est en $O(n^2)$ et qui consiste à rechercher successivement le plus petit élément de l, puis de l privé de cet élément, etc. Pour comparer les temps de calcul, on peut utiliser la commande `time`.

Exercice 5 – [Recherche dans une liste triée]

Écrire une fonction qui cherche un élément dans une liste triée par dichotomie. Montrer que le nombre de comparaisons utilisées par cet algorithme est en $O(\log n)$.