Université de Bordeaux
Mathématiques

Algèbre et calcul formel – Master
Année 2016–2017

FEUILLE D'EXERCICES n° 3

Travail sur machine

Diviser pour rêgner : Karatsuba, Tri fusion

Dans ce travail, l'exercice 1 prépare l'exercice 2, 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 et vice-versa, ne serait-ce que pour vérifier ses résultats. C'est pourquoi l'exercice 1 propose quelques commandes dédiées aux listes et polynômes.

Exercice 1 – [Listes et polynômes]
Exécuter les commandes suivantes.

```
p=x^4+x+1;p
parent(p)
list(p)
```

Cela ne fonctionne pas. Pour pouvoir transformer un polynôme en liste de cette façon, il faut indiquer que p est un polynôme, et spécifier l'anneau de polynômes dont il est élément. Pour cela, on peut exécuter la commande suivante.

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

Alors `Zx` est le nom de l'anneau que l'on définit (on aurait pu l'appeler `toto`, ou autre chose). Le suffixe `.<x>` indique que `x` est la variable de l'anneau de polynômes, et `ZZ` indique l'ensemble où l'on prend les coefficients (`ZZ` pour $\mathbb{Z}$, `QQ` pour $\mathbb{Q}$, `RR` pour $\mathbb{R}$, `CC` pour $\mathbb{C}$, `GF(5)` pour $\mathbb{F}_5$) Ainsi, la commande ci-dessus définit `Zx`$= \mathbb{Z}[x]$. Exécuter à nouveau les commandes :

```
p=3*x^4+2*x+1;p
parent(p)
l=list(p);l
```

l est une liste de longueur 5.

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
```

Pour évaluer q en un élément a, il suffit de faire `q(a)`. Par exemple :

```
q(5)
q(i)
q(1.2)
```

Les commandes

```
q[2]
q[5]
q[10]
```

donnent les coefficients de x^2, x^5, x^{10} dans le polynôme q. Exécuter la commande

```
[q[i] for i in range(4)]
```

Ainsi, si le programme `Karatsuba` utilise de telles commandes, alors il acceptera indifféremment listes ou polynômes en entrées.

Attention, il y a quand même des différences. Essayer par exemple `l[2:5]` et `q[2:5]`. Pour plus de souplesse, il sera donc préférable d'utiliser `[l[i] for i in range(2,5)]` puisque cette dernière commande s'applique aussi bien aux listes qu'aux polynômes.

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 2 – [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.

3) La tester numériquement avec de gros polynômes que l'on choisira de façon aléatoire.

Exercice 3 – [Tri fusion]

1) Écrire une procédure permettant de générer une liste l dont les éléments sont les entiers 1, 2, ... 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`.