Multiplication de polynômes

Dans ce TP, nous allons étudier 3 façons différentes de multiplier des polynômes à coefficients entiers. Ce sont 3 des 4 façons les plus utilisées dans les différents logiciels. De plus, les mêmes techniques s'appliquent très facilement aux multiplications de grands nombres entiers. Deux des méthodes que nous allons voir utilisent la technique dite diviser pour régner*, qui consiste à découper un gros problème en deux (ou plus) sous-problèmes qu'on subdivisera de nouveau. Cette technique est par exemple utilisée dans le tri fusion.*

1 Différentes fonctions auxilliaires

En Caml, aucun type "polynôme" n'est prédéfini ; nous allons donc en définir un, ainsi que des fonctions de base (notamment pour accéder aux coefficients).

▶ **Question 1** *Définissez un nouveau type Caml pour représenter un polynôme. Cela pourra être par exemple un vecteur d'entiers.*

Maintenant que l'on a ce type, il faut définir des fonctions de base pour le rendre utilisable :

▶ **Question 2** *Ecrivez les fonctions* degre *qui renvoie le degré d'un polynôme (ou plutôt la taille du vecteur le représentant),* get_coef *et* set_coef *qui renvoie et définit le coefficient de degré i d'un polynôme, une fonction* nouveau_polynome *qui crée un polynôme nul de taille donnée, ainsi que la fonction* rand_polynome *qui génère un polynôme de degré donné aléatoirement (il n'y a pas besoin de faire attention à avoir une distribution uniforme, c'est uniquement pour tester).*
Pour la fonction rand_polynome*, on pourra utiliser la fonction* random__int*.*

```
degre : polynome −> int = <fun>
nouveau_polynome : int −> polynome = <fun>
get_coef : polynome −> int −> int = <fun>
set_coef : polynome −> int −> int −> unit = <fun>
rand_polynome : int −> polynome = <fun>
```

Maintenant que nous avons ces fonctions de base, nous pouvons définir plusieurs opérations de base :

▶ **Question 3** *Ecrivez les fonctions* addition *(respectivement* soustraction *) qui additionne (!) (respectivement soustrait) deux polynômes. Ecrivez également une fonction* decale_polynome *telle que* decale_polynome : $(\mathrm{P}(X), n) \mapsto \mathrm{P}(X).X^n$

```
addition : polynome −> polynome −> polynome = <fun>
soustraction : polynome −> polynome −> polynome = <fun>
decale_polynome : polynome −> int −> polynome = <fun>
```

2 Multiplication naïve

Dans cette partie, nous allons nous attarder sur l'algorithme naïf de multiplication. Considérons les polynômes $P(X) = a_3X^3 + a_2X^2 + a_1X + a_0$ et $Q(X) = b_2X^2 + b_1X + b_0$.
Notons $R(X) = P(X)Q(X) = c_5X^5 + c_4X^4 + c_3X^3 + c_2X^2 + c_1X + c_0$.
On a

$$c_k = \sum_{i+j=k} (a_i b_j)$$

. ▶ **Question 4** *Ecrivez une fonction* mult_coef *qui prend en entrée deux polynômes et calcule le k^{eme} coefficient du produit.*

```
mult_coef : polynome −> polynome −> int −> int = <fun>
```

▶ **Question 5** *En utilisant la fonction précédente, écrivez une fonction* multiplication *qui calcule le produit de deux polynômes.*

```
multiplication : polynome −> polynome −> polynome = <fun>
```

3 Une méthode moins naïve : l'algorithme de Karatsuba

L'objectif principal est de faire le moins de multiplications possibles, même si pour cela on doit rajouter un certain nombre d'additions. Considérons pour l'instant deux polynômes de degré 1, $P(X) = a_1X + a_0$ et $Q(X) = b_1X + b_0$.
Leur produit est égal à $PQ(X) = a_1b_1X^2 + (a_0b_1 + a_1b_0)X + a_0b_0$, qui nécessite 4 multiplications (a_1b_1, a_0b_1, a_1b_0, a_0b_0).

Comment réduire ce nombre ?
Karatsuba a remarqué que $a_0b_1 + a_1b_0$ pouvait s'écrire sous la forme $((a_0 + a_1)(b_1 + b_0) - a_1b_1 - a_0b_0$ qui ne nécessite qu'une seule multiplication (car les deux autres ont déjà été calculées).
Nous allons donc découper les deux polynôme de taille $n = 2^p$ en deux parties de degré $n/2 - 1$: $P(X) = p_1(X)X^{p/2} + p_0$ et $Q(X) = q_1(X)X^{p/2} + q_0$. Evidemment, les 3 produits intermédiaires seront faits récursivement, en les redécoupant en 2.
Cette méthode permet une complexité de l'ordre de $O(n^{1.59})$ multiplications pour des polynômes de degré n, au lieu de $O(n^2)$ multiplications pour la méthode naïve (on gagne un facteur 100 pour $n = 100000$, par exemple).

► **Question 6** *Malheureusement, tous les polynômes en liberté n'ont pas un degré de la forme $n = 2^p - 1$ (et donc une taille de la forme 2^p). Pour corriger ce regrettable défaut, programmez une fonction qui prend en argument deux polynômes et les "aggrandit" pour leur donner une taille correcte (donc de la forme 2^p).*

```
arrondi_polynome : polynome -> polynome ->
polynome * polynome = <fun>
```

Maintenant, il faut pouvoir faire l'opération inverse, afin d'avoir des résultats plus lisibles.

► **Question 7** *Ecrivez une fonction* simplifie_polynome *qui supprime les coefficients nuls en tête d'un polynôme.*

```
simplifie_polynome : polynome -> polynome = <fun>
```

► **Question 8** *Pour utiliser la méthode de Karatsuba, il faut pouvoir découper les deux polynômes à multiplier en 2 polynômes de tailles égales. Ecrivez donc une fonction* decoupe_polynome *qui prend en argument 2 polynômes quelconques, leur donne une taille correcte (de la forme 2^p) et les coupe en deux sous-polynômes de même taille.*

```
decoupe_polynome : polynome -> polynome ->
polynome * polynome * polynome * polynome = <fun>
```

► **Question 9** *Ecrivez une première version* karatsuba_simple *qui découpe les deux arguments en 2 polynômes de même taille, calcule les produits intermédiaires avec la fonction* multiplication *naïve et reconstruit la réponse avec les fonctions* addition, decale_polynome *et* soustraction .

```
karatsuba_simple : polynome -> polynome -> polynome = <fun>
```

► **Question 10** *Ecrivez une fonction récursive* karatsuba *qui ne plus appel à* multiplication *pour les produits intermédiaires (sauf quand les produits à calculer ont un degré inférieur à 1).*

```
karatsuba : polynome -> polynome -> polynome = <fun>
```

En "vrai", les multiplications des petits polynômes de degré 1 ne sont pas faites avec la méthode naïve. La méthode utilisée consiste à les évaluer en 3 points (par exemple 0, 1, et 2), calculer les 3 produits $P(0)Q(0)$, $P(1)Q(1)$ et $P(2)Q(2)$ et interpoler le résultat.

4 Autre méthode : Toom-3

Nous allons étudier un cas particulier (*Toom-3*) d'une généralisation de la méthode de Karatsuba (*Toom-k*). Au lieu de découper P et Q en 2 sous-polynômes, l'idée est de les couper en 3 morceaux de même taille, toujours dans l'espoir de réduire le nombre de multiplications.
Supposons que P et Q soient tous les deux de taille $3n$ (donc de degré $3n - 1$). On peut alors écrire $P(X) = p_2X^{2n} + p_1(X)X^n + p_0(X)$ et $Q(X) = q_2X^{2n} + q_1(X)X^n + q_0(X)$. Le produit $PQ(X)$ vaut ainsi $PQ(X) = p_2q_2X^{4n} + (p_2q_1 + p_1q_2)X^{3n} + (p_2q_0 + p_1q_1 + p_0q_2)X^{2n} + (p_1q_0 + p_0q_1)X^n + p_0q_0$.
On écrit alors $(p_2q_1 + p_1q_2) = (p_1 + p_2)(q_2 + q_1) - p_1q_1 - q_2p_2$, $(p_0q_1 + p_1q_0) = (p_1 + p_0)(q_0 + q_1) - p_1q_1 - q_0p_0$, et $(p_2q_0 + p_0q_2) = (p_0 + p_2)(q_2 + q_0) - p_0q_0 - q_2p_2$. Cette méthode se fait en 6 multiplications (au lieu de 9) mais une méthode avec 5 multiplications est censée exister.
Cette méthode permet une complexité de l'ordre de $O(n^{1.47})$ multiplications, on gagne alors un facteur 450 pour $n = 100000$ par rapport à la multiplication naïve.

► **Question 11** *Ecrivez une fonction* arrondi_toom3 *qui prend en argument 2 polynômes et renvoie deux polynômes égaux, mais dont la taille est un multiple de 3.*

```
arrondi_toom3 : polynome -> polynome -> polynome * polynome = <fun>
```

► **Question 12** *Ecrivez une fonction* decoupe_toom3 *qui prend en argument 2 polynômes et qui renvoie 6 sous-polynômes de même taille $n/3$.*

```
decoupe_toom3 : polynome -> polynome ->
 polynome * polynome * polynome * polynome * polynome * polynome = <fun>
```

► **Question 13** *Ecrivez une fonction* toom3 *qui calcule le produit de 2 polynômes en utilisant les fonctions précédentes. On pourra utiliser la fonction* multiplication *pour les polynômes de degré inférieur à 2.*

```
toom3 : polynome -> polynome -> polynome = <fun>
```

*La dernière méthode classique de multiplication de polynômes est basée sur la Transformée de Fourier Rapide (*FFT*), qui a une complexité en $O(n\log n)$ (gain supérieur à 8000 pour $n = 100000$).*

Multiplication de polynômes

Un corrigé

▶ Question 1

```
type polynome = coef of int vect;;
```

▶ Question 2

```
let degre = fun (coef a) -> (vect_length a) - 1;;

let nouveau_polynome = fun n -> (coef (make_vect (n+1) 0));;

let get_coef = fun (coef a) i ->
let n = (vect_length a - 1) in
if (i > n) then 0
else a.(n-i)
;;

let set_coef = fun (coef a) i v ->
let n = (vect_length a - 1) in
if(i <= n) then a.(n-i) <- v
;;

let rand_polynome = fun degre ->
let resultat = nouveau_polynome degre in
for i = 0 to degre do (
set_coef resultat i ((random__int 1000) - 500 );
)
done;
resultat
;;

let a = rand_polynome 10;;
let b = rand_polynome 15;;
```

▶ Question 3

```
let addition = fun (coef p) (coef q) ->
let n = degre (coef p) and m = degre (coef q) in
let resultat = nouveau_polynome (max n m) in
for i = 0 to (max n m) do
set_coef resultat i ((get_coef (coef p) i) + (get_coef (coef q) i))
done;
resultat
;;

addition a b;;
let soustraction = fun (coef p) (coef q) ->
let n = degre (coef p) and m = degre (coef q) in
let resultat = nouveau_polynome (max n m) in
for i = 0 to (max n m) do
set_coef resultat i ((get_coef (coef p) i) - (get_coef (coef q) i))
done;
resultat
;;

soustraction a b;;

let decale_polynome = fun (coef p) k ->
let n = degre (coef p) in
let resultat = nouveau_polynome (n + k) in
for i = 0 to n do (set_coef resultat (i+k) (get_coef (coef p) i);) done;
resultat
;;
```

▶ Question 4

```
let mult_coef = fun (coef p) (coef q) i ->
let n = degre (coef p) and m = degre (coef q) and temp = ref 0 in
temp := 0;
for j = (max (i-n) 0) to (min m i) do (
temp := !temp + (get_coef (coef p) (i-j)) * (get_coef (coef q) j);
)
done;
!temp
;;
```

▶ Question 5

```
let multiplication = fun (coef p) (coef q) ->
let n = degre (coef p) and m = degre (coef q) in
let resultat = nouveau_polynome (n + m) in
for i = 0 to (n + m) do (
set_coef resultat i (mult_coef (coef p) (coef q) i);
)
done;
resultat
;;
multiplication a b;;
```

▶ Question 6

```
let arrondi_polynome = fun (coef p) (coef q) ->
let n = degre (coef p) and m = degre (coef q) in
let temp0 = ((max n m) + 1) and temp1 = ref 0 and temp2 = ref 0. in
temp1 := temp0;
temp2 := log (float_of_int !temp1);
temp2 := (!temp2) /. (log 2.);
(* oui, je sais, ce n'est pas joli, mais je prefere bien separer les etapes*)
temp1 := int_of_float !temp2;
temp1 := int_of_float (2. **. (float_of_int !temp1));
if ( !temp1 <> temp0) then temp1 := !temp1 * 2;
(* il n'y a pas de partie entière supérieure en Caml :( *)
temp1 := (!temp1) - 1;
let p_prime = nouveau_polynome (!temp1) in
let q_prime = nouveau_polynome (!temp1) in
for i = 0 to n do (set_coef p_prime i (get_coef (coef p) i);) done;
for i = 0 to m do (set_coef q_prime i (get_coef (coef q) i);) done;
(p_prime, q_prime)
;;
arrondi_polynome (rand_polynome 3) (rand_polynome 4);;
```

▶ Question 7

```
let simplifie_polynome = fun (coef p) ->
let n = degre (coef p) in
let i = ref n in
while((!i >= 0)&&(get_coef (coef p) !i = 0)) do ( i := !i - 1 ; ) done;
let resultat = nouveau_polynome !i in
for j = 0 to (!i) do (set_coef resultat j (get_coef (coef p) j);) done;
resultat
;;
```

▶ Question 8

```
let decoupe_polynome = fun (coef p) (coef q) ->
let (p_prime, q_prime) = arrondi_polynome (coef p) (coef q) in
let n = (((degre p_prime) + 1)/2 - 1) in
let p1 = nouveau_polynome n and p0 = nouveau_polynome n in
let q1 = nouveau_polynome n and q0 = nouveau_polynome n in
for i = 0 to n do (
set_coef p0 i (get_coef p_prime i);
set_coef p1 i (get_coef p_prime (i+n+1));
set_coef q0 i (get_coef q_prime i);
set_coef q1 i (get_coef q_prime (i+n+1));
)
done;
(p0, p1, q0, q1)
;;
```

▶ Question 9

```
let karatsuba_simple = fun p q ->
let (p0, p1, q0, q1) = decoupe_polynome p q in
(*p = p0 + X^(2n)*p1; q = q0 + X^(2n)*q1 *)
let p1q1 = multiplication p1 q1 and p0q0 = multiplication p0 q0 in
let n = (degre p0) + 1 in
let temp1 = addition p1q1 p0q0 in
let p1_p0 = addition p1 p0 in
let q1_q0 = addition q1 q0 in
let temp2 = multiplication p1_p0 q1_q0 in
let centre0 = soustraction temp2 temp1 in
let centre1 = decale_polynome centre0 (n) in
simplifie_polynome (addition (decale_polynome p1q1 (2*n)) (addition centre1 p0q0))
;;
```

▶ Question 10

```
let rec karatsuba = fun p q ->
let (p0, p1, q0, q1) = decoupe_polynome p q in
(*p = p0 + X^(2n)*p1; q = q0 + X^(2n)*q1 *)
let n = (degre p0) + 1 in
if n <= 1 then (
multiplication p q;
)
else (
let p1q1 = karatsuba p1 q1 and p0q0 = karatsuba p0 q0 in
let temp1 = addition p1q1 p0q0 in
let p1_p0 = addition p1 p0 in
let q1_q0 = addition q1 q0 in
let temp2 = multiplication p1_p0 q1_q0 in
let centre0 = soustraction temp2 temp1 in
let centre1 = decale_polynome centre0 (n) in
simplifie_polynome (addition
(decale_polynome p1q1 (2*n))
(addition centre1 p0q0));
)
;;
multiplication a b;;
karatsuba_simple a b;;
karatsuba a b;;
```

▶ Question 11

```
let arrondi_toom3 = fun (coef p) (coef q) ->
let n = degre (coef p) and m = degre (coef q) in
let temp0 = ((max n m) + 1) in
let temp1 = ref (temp0/3) in
if (3 * (!temp1) < temp0) then incr temp1;
temp1 := 3*(!temp1) - 1;

let p_prime = nouveau_polynome (!temp1) in
let q_prime = nouveau_polynome (!temp1) in
for i = 0 to n do (set_coef p_prime i (get_coef (coef p) i);) done;
for i = 0 to m do (set_coef q_prime i (get_coef (coef q) i);) done;
(p_prime, q_prime)
;;
arrondi_toom3 (rand_polynome 3) (rand_polynome 4);;
```

▶ Question 12

```
let decoupe_toom3 = fun (coef p) (coef q) ->
let (p_prime, q_prime) = arrondi_toom3 (coef p) (coef q) in
let n = (((degre p_prime) + 1)/3 - 1) in
let p1 = nouveau_polynome n and p0 = nouveau_polynome n in
let q1 = nouveau_polynome n and q0 = nouveau_polynome n in
let p2 = nouveau_polynome n and q2 = nouveau_polynome n in
for i = 0 to n do (
set_coef p0 i (get_coef p_prime i);
set_coef p1 i (get_coef p_prime (i+n+1));
set_coef p2 i (get_coef p_prime (i+2*(n+1)));
set_coef q0 i (get_coef q_prime i);
set_coef q1 i (get_coef q_prime (i+n+1));
set_coef q2 i (get_coef q_prime (i+2*(n+1)));
)
done;
(p0, p1, p2, q0, q1, q2)
;;
```

▶ Question 13

```
let toom3_simple = fun p q ->
let (p0, p1, p2, q0, q1, q2) = decoupe_toom3 p q in
(* p*q = w4*x^4n + w3*x^3n + w2*x^2n + w1*x^n + w0 *)
let n = (degre p0) + 1 in
let w0 = multiplication p0 q0 in
let p1q1 = multiplication p1 q1 in
let w4 = multiplication p2 q2 in
let w3 = soustraction (multiplication (addition p1 p2) (addition q1 q2))
(addition p1q1 w4) in
let w1 = soustraction (multiplication (addition p1 p0) (addition q1 q0))
(addition p1q1 w0) in
let p2q0p0q2 = soustraction (multiplication (addition p2 p0) (addition q2 q0))
(addition w4 w0) in
let w2 = addition p2q0p0q2 p1q1 in
let w1p = decale_polynome w1 n in
let w2p = decale_polynome w2 (2*n) in
let w3p = decale_polynome w3 (3*n) in
let w4p = decale_polynome w4 (4*n) in
simplifie_polynome (addition w4p (addition w3p (addition w2p (addition w1p w0))))
;;
multiplication a b;;
toom3_simple a b;;

let rec toom3 = fun p q ->
let (p0, p1, p2, q0, q1, q2) = decoupe_toom3 p q in
(* p*q = w4*x^4n + w3*x^3n + w2*x^2n + w1*x^n + w0 *)
let n = (degre p0) + 1 in
if( n <= 1) then (
multiplication p q;
)
else (
let w0 = toom3 p0 q0 in
let p1q1 = toom3 p1 q1 in
let w4 = toom3 p2 q2 in
let w3 = soustraction (toom3 (addition p1 p2) (addition q1 q2))
(addition p1q1 w4) in
let w1 = soustraction (toom3 (addition p1 p0) (addition q1 q0))
(addition p1q1 w0) in
let p2q0p0q2 = soustraction (toom3 (addition p2 p0) (addition q2 q0))
(addition w4 w0) in
let w2 = addition p2q0p0q2 p1q1 in
let w1p = decale_polynome w1 n in
let w2p = decale_polynome w2 (2*n) in
let w3p = decale_polynome w3 (3*n) in
let w4p = decale_polynome w4 (4*n) in
simplifie_polynome (addition w4p (addition w3p (addition w2p (addition w1p w0))))
)
;;
multiplication a b;;
toom3 a b;;
```