TD 4 :
Transformée de Fourier rapide

Correction

1 Opérations sur polynômes

Question 1 : Correction : Pour $E \subseteq \{0 \dots n\}$, on utilise le polynôme caractéristique :

$$P_E = \sum_{k=0}^{10n} \chi_E(k) X^k$$

Ainsi, $P_A \times P_B = \sum_{k=0}^{20n} \alpha_k X^k$ avec :

$$\alpha_k = \sum_{p+q=k} \chi_A(p)\chi_B(q) = \sum_{p+q=k} \chi_{A\times B}((p,q)) = |\{(x,y) \in A \times B | x+y = k \in C\}|$$

En utilisant cette idée, on obtient l'algorithme :

1. calculer les tableaux des coefficients de P_A et P_B (en $\Theta(n)$)
2. calculer $P_A \times P_B$ à l'aide de FFT (dans l'anneau $A[X]/(X^{20n}-1)$, donc $O(n \log n)$),
3. le résultat demandé est donné par les coefficients de $P_A \times P_B$.

L'algorithme est donc en $O(n \log n)$.

Question 2 : Correction : Supposons qu'on ait une représentation par point-valeur de A, c'est à dire $(u_1, \dots, u_n) = (A(v_1), \dots, A(v_n))$. Pour des valeurs inversibles x ($x \neq 0$), on a :

$$\widetilde{A}(x) = \sum_{j=0}^{n-1} a_{n-1-j} x^j = \sum_{j=0}^{n-1} a_j x^{n-1-j} = x^{n-1} \sum_{j=0}^{n-1} a_j \left(\frac{1}{x}\right)^j = x^{n-1} A\left(\frac{1}{x}\right)$$

D'où $\widetilde{A}\left(\frac{1}{v_i}\right) = \left(\frac{1}{v_i}\right)^{n-1} A(v_i) = \frac{u_i}{v_i^{n-1}}$. Le calcul de $\frac{u_i}{v_i^{n-1}}$ nécessite $O(\log n)$ opérations, par exponentiation rapide de v_i ; la complexité pour les n valeurs est donc $O(n \log n)$.

Si les v_i sont les n racines n-ième de l'unité, alors $v_i^{n-1} = \frac{1}{v_i}$ d'où $\widetilde{A}\left(\frac{1}{v_i}\right) = v_i u_i$. De plus, nous avons obtenu une représentation point-valeur sur les racines n-ème de l'unité pour $\widetilde{A}$ (modulo une permutation par rapport à A). Dans ce cas, la complexité du calcul est en $O(n)$.

Question 3 : Correction : Il faut calculer $Pr_{0,n-1} = \prod_{0 \leq k \leq n-1}(X - z_k)$. On pose $Pr_{i,j} = \prod_{i \leq k \leq j}(X - z_k)$. On calcule en temps constant les $Pr_{i,i} = (X - z_i)$. On utilise la relation de récurrence, pour $i < j$, $Pr_{i,j} = Pr_{i,p} Pr_{p+1,j}$ avec $p = \lfloor \frac{i+j}{2} \rfloor$.

Ainsi, un algorithme de type diviser pour régner peut être mis en place et la complexité C_n de calcul de $P_{0,n-1}$ vérifie, en prenant $p = \lfloor \frac{n}{2} \rfloor$ (f est le coût du produit de polynômes) la relation

suivante :

$$\begin{aligned}
C_n &= C_{p+1} + C_{n-1-p} && + f\left(\max(p+1, n-1-p)\right)\\
&\leq 2C_{\frac{n+1}{2}} && + O(n\log n)\\
&\leq 2(2C_{\frac{n}{2^2}} + O(\frac{n}{2}\log\frac{n}{2})) && + O(n\log n)\\
&\leq 2^2 C_{\frac{n}{2^2}} && + 2O(n\log n)\\
(\log n \ fois)... &\leq \log^n O(n\log n) = O(n\log^2 n)
\end{aligned}$$

où $f(n) = O(n\log n)$ car on utilise FFT pour deux polynômes de degrés bornés par n.

Question 4 : <u>Correction</u> : On a $P = A \times (x-z) + R$ avec $deg(R) < deg(x-z)$, donc R est une constante. Nécessairement, cette constante est égale à $P(z)$ car $x - z$ s'annule en $x = z$.

Le but est donc de calculer $R_{i,i}$ où $R_{i,j} = P \mod Pr_{i,j}$ avec $Pr_{i,j} = \prod_{i\leq k\leq j}(X - z_k)$ les polynômes calculés en $O(n\log^2 n)$ au point précédent.

D'après l'hypothèse, le calcul de $R_{0,n-1} = P \mod Pr_{0,n-1}$ est fait en $O(n\log n)$.

On remarque que pour $p = \lfloor n/2 \rfloor$ on a $R_{0,p} = P \mod Pr_{0,p} = (A \times Pr_{0,n-1} + R_{0,n-1}) \mod Pr_{0,p} = (A \times Pr_{0,p}Pr_{p+1,n-1} + R_{0,n-1}) \mod Pr_{0,p} = R_{0,n-1} \mod Pr_{0,p}$ et $deg(R_{0,p}) < p + 1$.

On peut utiliser un algorithme diviser pour régner (prenons $n = 2^k$ pour faciliter l'écriture) :

— en utilisant l'exercice précédent, on pré-calcule tous les $Pr_{i,j}$ (en $O(n\log^2 n)$) ;

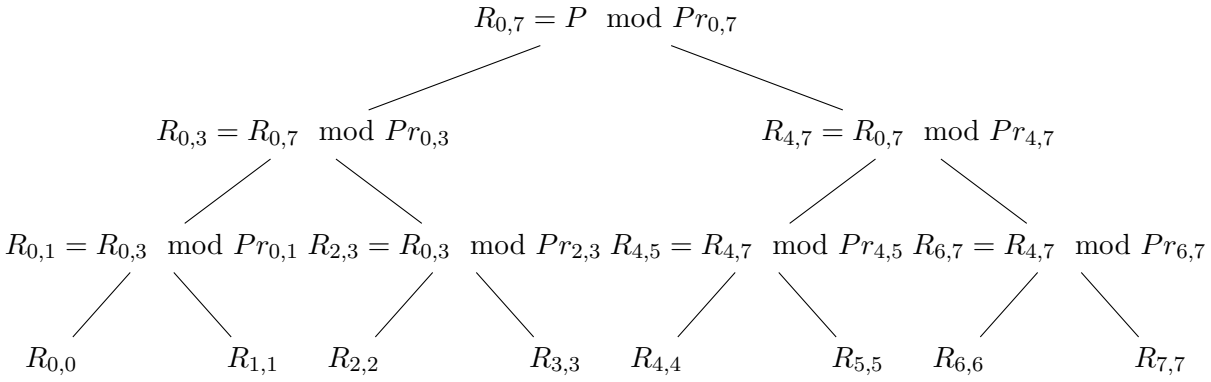
— puis on appelle $Rem(0, k, Pr, R)$ où R sera en entrée et en sortie de la procédure $Rem(i, h, Pr, R)$ avec $0 \leq i \leq n-1$ et $0 \leq h \leq k = \log n$ définie par :

— si $h = k$ alors $R_{0,n-1} = P \mod Pr_{0,n-1}$,

— sinon $R_{i,i+2^h-1} = R_{i-(i \mod 2^{h+1}),i-(i \mod 2^{h+1})+2^{h+1}-1} \mod Pr_{i,i+2^h-1}$ et, si $h \geq 1$ alors on appelle $Rem(i, h-1, Pr, R)$ et $Rem(i + 2^{h-1}, h-1, Pr, R)$.

On a un arbre d'appels récursif de profondeur $k = \log n$ où, à la hauteur h, le degré des polynômes est d'au plus 2^h.

La complexité totale est $O(n\log^2 n) + O(n\log n + 2n\log n + 4\frac{n}{2}\log\frac{n}{2} + \ldots)$ où on a moins de $\log n$ termes $n\log n$, donc $O(n\log^2 n)$.

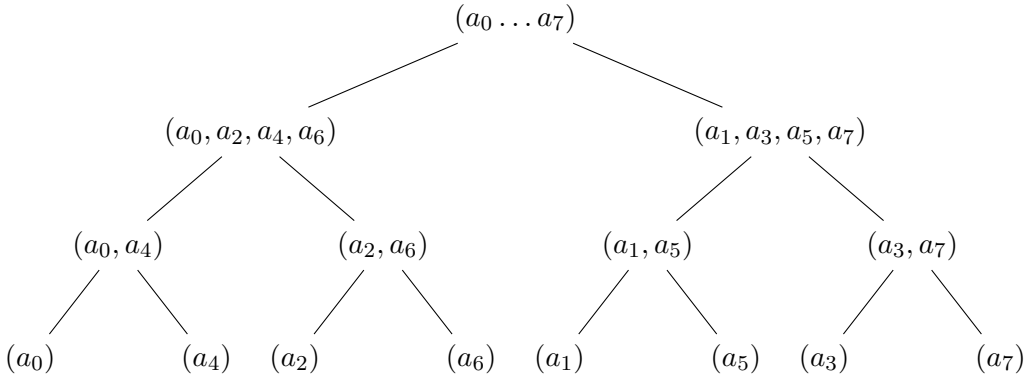


2 FFT itérative, FFT parallèle

Question 5 : Correction : L'algorithme de FFT est basé sur la décomposition :

$$P(X) = P^{(0)}(X^2) + XP^{(1)}(X^2)$$
$$P^{(0)} = \sum_{i=0}^{n/2-1} p_{2i}X^i$$
$$P^{(1)} = \sum_{i=0}^{n/2-1} p_{2i+1}X^i$$

Pour l'exemple indiqué, l'arbre d'appels est donné ci-dessous. A la racine (profondeur $p = 0$), le fils gauche récupère la partie (des coefficients de parité) paire du polynôme tandis que la partie droite récupère la partie impaire. Pour une profondeur $p > 0$, le fils gauche (resp. droit) récupère les indices i tels que $i/2^p$ est paire (resp. impair).



Question 6 : Correction : Notons F_k la liste réordonnée des indices de $a_0 \dots a_{2^k-1}$. Ainsi, $F_0 = (0)$. On remarque de plus que $F_{k+1} = (map\ (\lambda x.\ 2 * x)\ F_k)@(map\ (\lambda x.\ 2 * x + 1)\ F_k)$ où @ désigne l'opérateur de concaténation et *map* applique la fonction à chaque élément de la liste.

Algorithm 1: RÉORDONNE

Input : $n = 2^k$ et $a = [a_0, \dots, a_{n-1}]$
Output: T la liste ordonnée des a_i selon l'ordre des feuilles de l'arbre

```
T[0] ← 0
n' ← 1
while n' < n do
    for i = 0 to n' − 1 do
        T[i] ← 2 ∗ T[i]
        T[i + n'] ← T[i] + 1
    n' ← 2 ∗ n'
for i = 0 to n − 1 do
    T[i] ← a[T[i]]
```

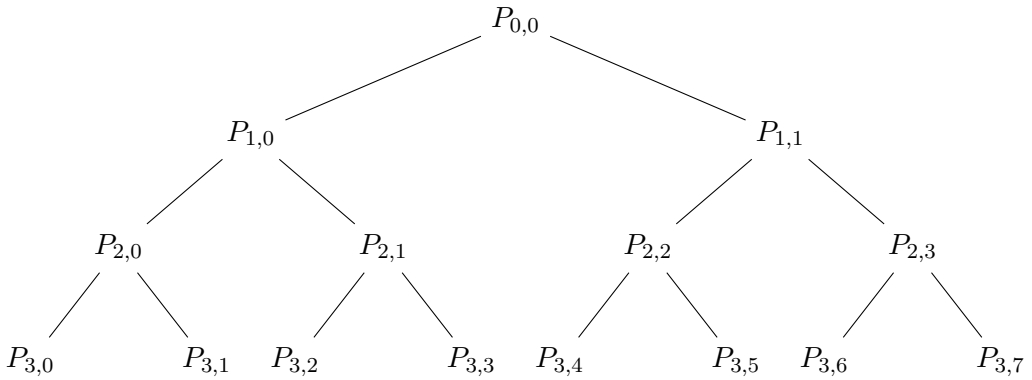
Complexité : Notons $n' = 2^p$ ($n = 2^k$). Il y a 5 opérations dans la boucle **for**, donc la complexité totale est $\sum_{p=0}^{k-1}(5 \cdot 2^p + 2) = 5 \cdot \left(2^k - 1\right) + 2k \in O(n)$

Une autre solution est basée sur la représentation binaire des indices. On remarque que RÉORDONNE$(P)[i] = P[\widetilde{i}]$ avec $\widetilde{i}$ le miroir de la représentation binaire de i. En utilisant un

compteur partant de 0, RÉORDONNE(P) doit incrémenter son image miroir, ce qui peut être fait en complexité amortie de $O(1)$.

Question 7 : Correction : Si $n = 2^k$, l'arbre a la profondeur k. Pour une profondeur $h \in [1, k]$, le nombre de noeuds à profondeur h est 2^h. Pour $0 \leq j < 2^h$, notons $P_{h,j}$ le j-ème polynôme de l'arbre à profondeur h. En utilisant la décomposition de FFT, on a la relation de récurrence :

$$P_{h-1,j}(X) = P_{h,2j}(X^2) + XP_{h,2j+1}(X^2)$$



Soient $(\omega^0, ..., \omega^{n-1})$ les puissances de ω, la racine primitive n-ème de l'unité. On remarque que pour $h = k - i$ on doit évaluer $P_{h,j}$ en 2^i points :

$h = k - 1$ en 2^1 points w^0 et $w^{\frac{n}{2}}$,

$h = k - 2$ en 2^2 points w^0, $w^{\frac{n}{4}}$, $w^{\frac{2n}{4}}$, $w^{\frac{3n}{4}}$.

Donc pour $h = k - i$ on évalue en $\{w^{\frac{\ell n}{2^i}} \mid 0 \leq \ell < 2^i\}$ et la distance entre chaque point est $\frac{n}{2^i}$.

De plus, pour $h = k - i$ et $\ell' \in \{0, \ldots, 2^i - 1\}$ (rappel $\omega^n = 1$ et $\omega^{n/2} = -1$) :

$$P_{h-1,j}(\omega^{\frac{\ell' n}{2^{i+1}}}) = P_{h,2j}(\omega^{\frac{\ell' n}{2^i}}) + \omega^{\frac{\ell' n}{2^{i+1}}} P_{h,2j+1}(\omega^{\frac{\ell' n}{2^i}})$$
$$P_{h-1,j}(\omega^{\frac{(2^i+\ell')n}{2^{i+1}}}) = P_{h,2j}(\omega^{\frac{\ell' n}{2^i}}) - \omega^{\frac{\ell' n}{2^{i+1}}} P_{h,2j+1}(\omega^{\frac{\ell' n}{2^i}})$$

D'où au début de l'algorithme FFT itératif on a le tableau F' à partir de lequel on construit F en utilisant les relations ci-dessous :

F' :	$P_{3,0}(w^0)$	$P_{3,1}(w^0)$	$P_{3,2}(w^0)$	$P_{3,3}(w^0)$	$P_{3,4}(w^0)$	$P_{3,5}(w^0)$	$P_{3,6}(w^0)$	$P_{3,7}(w^0)$
F :	$P_{2,0}(w^0)$	$P_{2,0}(w^4)$	$P_{2,1}(w^0)$	$P_{2,1}(w^4)$	$P_{2,2}(w^0)$	$P_{2,2}(w^4)$	$P_{2,3}(w^0)$	$P_{2,3}(w^4)$

On en déduit l'algorithme FFT-ITÉRATIVE ci-après.

On maintient l'invariant si $h = \log(n/s)$, $\forall 0 \leq k < s$. $\forall 0 \leq p < \frac{n}{s}$. $F[p \cdot s + k] = P_{h,p}(\omega^{k \cdot n/s})$. Il est initialement vrai lorsque $s = 1$ et que l'on a appliqué RÉORDONNE car on obtient les polynômes constants.

Cet algorithme a la même complexité que FFT car à chaque itération on calcule une ligne de l'arbre en $O(n)$ et il y a $\log n$ lignes.

Question 8 : Correction : On peut calculer en $O(n)$ le tableau F' à partir de F, de façon parallèle : on attribue à chaque processeur un des calculs, donné par le couple (p, k). On utilise alors $\frac{n}{s} \cdot \frac{s}{2} = \frac{n}{2}$ processeurs qui n'ont besoin que des données calculées à l'itération précédente. L'algorithme parallèle s'exécute ainsi en $\log n$ itérations.

Algorithm 2: FFT-Itérative

Input : P un polynôme de degré $n = 2^k$
Input : $T = [w^0, \dots, w^{n-1}]$ avec w racine primitive de l'unité
Output: le tableau des valeurs $P(w^0), \dots, P(w^{n-1})$

```
F' ← [0 times n]
F ← Reordonne(n, P)
s ← 1 ; d ← n
while s < n do
    s ← 2s ; d ← d/2
    for p = 0 to n/s − 1 do
        for k = 0 to s/2 − 1 do
            F'[p·s + k] ← F[p·s + k] + T[k·d] · F[p·s + k + s/2]
            F'[p·s + k + s/2] ← F[p·s + k] − T[k·d] · F[p·s + k + s/2]
    swap(F,F')
return F
```

3 Calcul des n premières dérivées d'un polynôme en un point

Question 9 : **Correction** : On a la formule de Taylor en x_0 :

$$\begin{aligned} A(X) = & \sum_{k=0}^{n-1} \frac{A^{(k)}(x_0)}{k!}(X - x_0)^k \\ = & \sum_{k=0}^{n-1} b_k (X - x_0)^k \end{aligned}$$

D'où par identification des coefficients :

$$\forall k \in \{0 \dots n-1\}.\ A^{(k)}(x_0) = k! b_k$$

On calcule $k!$ de proche en proche ($O(1)$) et on obtient ainsi les valeurs $A^{(k)}(x_0)$ en $O(n)$.

Question 10 : **Correction** : On dénote par $A(x_0 + X) = B(X) = \sum_{k=0}^{n-1} b_k X^k$. On connaît donc les valeurs $[B(\omega^0), \dots, B(\omega^{n-1})] = [A(x_0 + \omega^0) \dots A(x_0 + \omega^{n-1})]$. En utilisant FFT on peut trouver les coefficients b_j en $O(n \log n)$.

Question 11 : **Correction** : Pour $k \in \{0 \dots n-1\}$,

$$\begin{aligned} A(x_0 + \omega^k) &= \sum_{j=0}^{n-1} a_j (x_0 + \omega^k)^j \\ &= \sum_{j=0}^{n-1} \sum_{r=0}^{j} \binom{r}{j} a_j x_0^{j-r} \omega^{kr} \\ &= \sum_{r=0}^{n-1} \sum_{j=r}^{n-1} \frac{j!}{r!(j-r)!} a_j x_0^{j-r} \omega^{kr} \\ &= \sum_{r=0}^{n-1} \frac{\omega^{kr}}{r!} \sum_{j=r}^{n-1} a_j j! \frac{x_0^{j-r}}{(j-r)!} \\ &= \sum_{r=0}^{n-1} \frac{\omega^{kr}}{r!} \sum_{j=r}^{n-1} f(j) g(j-r) \end{aligned}$$

Avec

$$\begin{cases} f(j) = & a_j j! \\ g(l) = & \dfrac{x_0^l}{l!} \end{cases}$$

Question 12 : <u>Correction</u> :

$$\begin{aligned} A(x_0+\omega^k) = & \sum_{r=0}^{n-1} \frac{\omega^{kr}}{r!} \sum_{j=r}^{n-1} f(j)g(j-r) \\ = & \sum_{r=0}^{n-1} \frac{\omega^{kr}}{r!} \sum_{j=0}^{n-1-r} f(n-1-j)g(n-1-r-j) \\ = & \sum_{r=0}^{n-1} c_{n-1-r}\omega^{kr} \end{aligned}$$

Avec

$$c_r = \frac{1}{(n-1-r)!} \sum_{j=0}^{r} f(n-1-j)g(r-j)$$

Soient les polynômes $P(X) = \sum_{j=0}^{n-1} f(n-1-j)X^j$ et $Q(X) = \sum_{j=0}^{n-1} g(j)X^j$. Les coefficients du produit PQ sont :

$$\begin{aligned} PQ_0 &= f(n-1)g(0) = (n-1)!c_0 \\ PQ_1 &= f(n-1)g(1) + f(n-1-1)g((0) = (n-1-1)!c_1 \end{aligned}$$

Donc $PQ_r = (n-1-r)!c_r$.

Les coefficients de PQ sont calculés en $O(n \log n)$, et ensuite en $O(n)$ on obtient $\vec{c} = [c_0, \ldots, c_{n-1}]$. On utilise le miroir de $\vec{c}$ pour faire une évaluation de $A_0(X) = \sum_{r=0}^{n-1} c_{n-1-r}X^r$ en les racines n-ème de l'unité. D'après le développement de $A(x_0+\omega^k) = A_0(\omega^k)$. Donc ces valeurs peuvent être calculées en $O(n \log n)$ par FFT.