Devoir en temps libre n° 6

Ce devoir est à préparer pour le lundi 18 juin 2018 et sera à rendre au début du TP. Vous devez apporter[1] votre code CAML avec vous, sous la forme d'un unique fichier `.ml` dont le nom sera obligatoirement et exactement `<login>.ml` où `<login>` est votre nom d'utilisateur du laboratoire d'informatique (tout en minuscules avec les tirets).

✔

Les questions commençant par le symbole ✍ sont à rédiger au propre et à rendre en début de TP.

Votre programme devra respecter scrupuleusement les noms et les types indiqués (ou un type plus général). Le code devra compiler sans erreurs ni aucun message d'avertissement. L'encodage du fichier devra être UTF-8 et vous n'utiliserez que des caractères ASCII pour les noms de variables. Il est impératif de veiller au strict respect de ces consignes.

Lorsque l'on pose des hypothèses sur les arguments (par exemple qu'un argument est un entier naturel), il n'est pas nécessaire dans vos fonctions de le vérifier ni de prévoir le comportement si l'hypothèse est violée (par exemple si l'argument est négatif). On pourra cependant lever des exceptions (`failwith "Argument non valable"` par exemple) si on le souhaite.

I Multiplication de polynômes

Dans ce problème, on se propose d'étudier un algorithme de multiplication de deux polynômes plus efficace que la version naïve que nous avions déjà rencontrée en DM.

1. Je vous conseille de vous l'envoyer par mail depuis chez vous *et* de l'apporter sur clef USB.

I.1 Préliminaire

1. Écrire une fonction `sub : 'a array -> int -> int -> 'a array` telle que `sub tableau debut longueur` renvoie un sous-tableau de `tableau` à partir de la position `debut` et de longueur `longueur`. On supposera que les arguments désignent bien un sous-tableau valable (éventuellement vide).

☞ **Remarque 1**

C'est la fonction `Array.sub` *que nous avons rencontrée au dernier DS.*

I.2 Opérations de base sur les polynômes

On s'intéresse dans cette section à des polynômes à coefficients entiers que l'on représentera à l'aide de tableaux[2]. Un polynôme $P = \sum_{k=0}^{n-1} a_k X^k \in \mathbb{Z}[X]$ est ainsi représenté par un tableau `poly` de type `int array` de taille n, l'entier `poly.(k)` représentant le coefficient a_k. Ainsi, le polynôme X^2+1 peut être représenté — par exemple — par le tableau `[|1; 0; 1; 0|]`. Il y a unicité de la représentation si a_{n-1} est non nul, c'est-à-dire si $n-1$ est le degré du polynôme. Dans ce cas, on dit que la représentation est *adaptée*. La représentation adaptée de X^2+1 est `[|1; 0; 1|]`.

2. ✍ Quelle représentation adaptée choisir pour le polynôme nul ?
3. Écrire une fonction `degre : int array -> int` qui calcule le degré d'un polynôme dont la représentation n'est pas nécessairement adaptée.
4. Écrire une fonction `reduire_repr : int array -> int array` qui transforme la représentation éventuellement non adaptée d'un polynôme en sa représentation adaptée.
5. Écrire une fonction `agrandir_repr : int array -> int -> int array` telle que `agrandir poly m` renvoie une représentation de `poly` de taille `m`. Cette fonction supposera que le degré du polynôme est inférieur ou égal à $m-1$.

2. Contrairement au DM n° 1 dans lequel on considérait des polynômes à coefficients flottants représentés par des listes.

Les complexités seront évaluées en fonction de la taille des représentations des polynômes, qui n'est pas nécessairement leur degré plus un si la représentation n'est pas adaptée.

I.3 Multiplication naïve

6. Écrire une fonction `somme` de type `int array -> int array -> int array` qui calcule la somme de deux polynômes quelconques. On ne suppose pas les représentations adaptées.
7. ✍ Quel est le nombre d'additions d'entiers nécessaires, si les deux polynômes ont une représentation de même taille n, en fonction de cette taille ?
8. Écrire une fonction `multiplication_naive : int array -> int array -> int array` qui calcule le produit de deux polynômes quelconques. On ne suppose pas les représentation adaptées.
9. ✍ Quel est le nombre nécessaire d'additions d'entiers d'une part et de multiplications d'autre part, si les deux polynômes ont une représentation de même taille n, en fonction de cette taille ?

I.4 Méthode de Karatsuba

Par la suite on va chercher à réduire le nombre de multiplications entières mises en jeu pour multiplier deux polynômes.

Soient P et Q deux polynômes dont la représentation a une taille $n = 2^k$ avec $k \in \mathbb{N}$. Si $n \geq 2$, on pose $m = \frac{n}{2}$ et on écrit $P = X^m P_1 + P_2$ et $Q = X^m Q_1 + Q_2$, la représentation de chacun des quatre polynômes P_1, P_2, Q_1 et Q_2 étant de taille m.

10. ✍ Que vaut PQ en fonction de P_1, P_2, Q_1 et Q_2 ?
11. ✍ Comment s'appelle le paradigme (la technique) que nous sommes en train de mettre en place ? Justifier rapidement.
12. ✍ Montrer que le nombre de multiplications $M(n)$ vérifie la relation de récurrence $M(n) = 4M(n/2) + \Theta(n)$.
13. ✍ Conclure.
14. ✍ On pose $R_1 = P_1Q_1$, $R_2 = (P_1 + P_2)(Q_1 + Q_2)$ et $R_3 = P_2Q_2$. Exprimer $P_1Q_2 + P_2Q_1$ en fonction de R_1, R_2 et R_3.
15. ✍ En déduire une expression de PQ qui ne nécessite que 3 appels récursifs au lieu de 4.
16. ✍ Quelle est alors la relation de récurrence vérifiée par $M(n)$?
17. ✍ Montrer que $M(n) = \Theta(n^c)$ ou c est une constante que vous expliciterez. Vérifier que $c \approx 1.58$.
18. Écrire une fonction `karatsuba : int array -> int array -> int array` qui implémente le produit de deux polynômes dont les deux représentations sont de taille $n = 2^k$ avec $k \in \mathbb{N}$ en utilisant la méthode décrite ci-dessus.
19. ✍ Quelle est sa complexité spatiale ?
20. Écrire une fonction `multiplication_karatsuba : int array -> int array -> int array` qui généralise la fonction précédente à des polynômes quelconques dont la représentation n'est pas nécessairement adaptée. La représentation du polynôme renvoyé comme résultat devra être adaptée. *Indication : il suffit de rajouter ou de supprimer des zéros.*

La similitude entre la représentation binaire d'un entier et les polynômes permet d'adapter l'algorithme précédent aux grands entiers (et inversement), la seule différence se trouve dans la gestion de la retenue. On peut utiliser la même idée pour la multiplication de deux matrices (algorithme de Strassen).

Dans les années 1950, Andreï Kolmogorov travaille sur la complexité des opérations arithmétiques et conjecture qu'une multiplication de deux nombres de n chiffres ne peut être réalisée en moins de $O(n^2)$ opérations. À l'époque, aucun algorithme plus rapide que la multiplication standard n'est connu. À l'automne 1960, Kolmogorov organise un séminaire dans lequel il parle de sa conjecture, auquel assiste Anatolii Alexevich Karatsuba. Une semaine plus tard, Karatsuba avait trouvé cet algorithme, qui prouvait que la conjecture était fausse.

L'algorithme Toom-Cook est un raffinement qui consiste à découper les polynômes en r blocs avec $r > 2$. La complexité peut alors passer en $O(n^{1+\epsilon})$ où ϵ est un réel strictement positif arbitraire. L'algorithme de Schönhage-Strassen, qui utilise la transformation de Fourier rapide, permet d'obtenir une complexité de $O(n \log n)$.