EA4 – Éléments d'algorithmique
TP n^{o} 3 : Multiplication de Karatsuba sur les polynômes

Modalités de rendu : À chaque TP, vous devrez rendre les exercices marqués par un symbole ✍. Le rendu de l'exercice K du TP N doit être inclus dans le fichier `tpN_exK.py`, à télécharger depuis la section « Énoncés de TP ». Vous devez remplir les zones marquées par le commentaire `A REMPLIR` dans ce fichier. Ne modifiez pas les autres fonctions du fichier, sauf demande explicite de l'énoncé. Chaque fichier contient une fonction `main` qui teste les fonctions que vous devez programmer et qui vous affiche un score donné par le nombre de tests passés avec succès. Pour passer ces tests, vous devez exécuter le programme écrit. La date limite du rendu d'un TP est le dimanche à 16 heures (heure de Paris).

Dans ce TP, on considère qu'un polynôme de degré n, $P(x) = p_0 + p_1 \cdot x + \ldots + p_n \cdot x^n$, est encodé par un tableau de longueur $n+1$ qui contient, en case d'indice i, le cœfficient p_i du monôme de degré i.

Exercice 1 : Produit de polynômes avec Karatsuba

Rappel du cours : L'algorithme de Karatsuba s'applique au cas des polynômes P dont le degré est de la forme $2^k - 1$. Pour tout tel polynôme P, on note $P^{(0)}$ et $P^{(1)}$ les polynômes de degré $2^{k-1} - 1$ tels que :

$$P = P^{(0)} + P^{(1)} \cdot X^{2^{k-1}}.$$

Le produit de tels polynômes peut se calculer par :

$$P \cdot Q = P^{(0)} \cdot Q^{(0)} + (P^{(1)} \cdot Q^{(0)} + P^{(0)} \cdot Q^{(1)}) \cdot X^{2^{k-1}} + P^{(1)} \cdot Q^{(1)} \cdot X^{2^k}$$

ou encore (méthode de Karatsuba)

$$\begin{aligned} P \cdot Q = \; & P^{(0)} \cdot Q^{(0)} + P^{(1)} \cdot Q^{(1)} \cdot X^{2^k} + \\ & \left[(P^{(0)} + P^{(1)})(Q^{(0)} + Q^{(1)}) - P^{(0)} \cdot Q^{(0)} - P^{(1)} \cdot Q^{(1)}\right] \cdot X^{2^{k-1}} \end{aligned}$$

Dans cet exercice, on considérera comme élémentaires les opérations d'additions et multiplications entre entiers, ainsi que chaque opération traitant un élément d'une liste : ajout d'un élement, affectation d'une valeur, etc... En particulier, on considérera donc que la création d'un tableau de longueur n coûtera n opérations ...

1. ✍ Écrire une fonction `minPuissanceDe2(k)` qui renvoie la plus petite puissance de 2 supérieure ou égale à `k`. (et `0` si `k==0`)
2. ✍ Écrire une fonction `polyDegreeAdapte(P,k)` qui prend un polynôme `P` et lui ajoute des termes de coefficient `0` pour renvoyer un polynôme de degré `k-1`.[1]

1. On considèrera que le polynôme representé par le tableau `[0,1,0,0]` correspond au polynôme $0 + X + 0 * X^2 + 0 * X^3$ et qu'il est de degré 3.

3. ✍ Écrire une fonction `polyAjoute(T,S,dec,neg)` qui, si `neg==False`, ajoute à un polynôme `T` un autre polynôme `S` en le décalant de `dec`. Quand `neg` vaut `True`, on soustrait `S` au lieu de l'ajouter. Attention, il faut absolument éviter de créer un nouveau tableau ! Le résultat doit être stocké dans le tableau `T` d'origine.

4. ✍ Enfin, écrire la fonction `polyProdKara(P,Q)` qui calcule le produit des polynômes en argument en utilisant l'algorithme de Karatsuba. Votre programme devra commencer par harmoniser le degré des polynômes à un degré de la forme $2^k - 1$.

5. ✍ Modifiez vos fonctions pour obtenir une fonction `polyProdKaraOps(P,Q)` qui compte le nombre d'opérations élémentaires utilisées en fonction de n, le degré des polynômes.

Pour les questions suivantes, vous pourrez comparer vos algorithmes à ceux déjà codés en utilisant la fonction `courbes_smooth_ops(k,algos)` où `algos` est un tableau de fonctions. Vous pourrez également tester leur correction en appelant `testPolyProd(algo)`. Vous pouvez également décommenter quelques lignes à la fin du fichier.

6. On pourrait encore économiser des opérations en évitant de copier inutilement des données lors des appels récursifs. Écrire une fonction `polyProdKara2Ops(P,Q)` faisant appel à une sous-fonction `polyProdKara2SubOps(P,debP,finP,Q,debQ,finQ)` qui effectue la multiplications des polynômes représentés par les tableaux `P[debP:finP]` et `Q[debQ:finQ]` *sans créer ces tableaux*, afin d'éviter le coût de création des sous-tableaux. On pourra également écrire une fonction

`polyAjoute2Ops(T,S,dec,debS,finS,neg)`

qui ajoute `S[debS:finS]` à `T` avec un décalage de `dec` sans créer `S[debS:finS]`. (On s'autorisera tout de même six créations de tableaux au lieu des huit précédents : un pour stocker le résultat final, deux pour stocker les sommes $P^{(0)} + P^{(1)}$ et $Q^{(0)} + Q^{(1)}$, ainsi que les trois tableaux contenant les sous-produits $P^{(0)}Q^{(0)}$, $P^{(1)}Q^{(1)}$ et $(P^{(0)} + P^{(1)})(Q^{(0)} + Q^{(1)})$)

(Bonus) Dernière économie sur l'allocation de tableaux. Écrire une fonction `polyProdKara3Ops(P,Q)` faisant appel à une sous-fonction `polyProdKara3SubOps(P,debP,finP,Q,debQ,finQ,PQ,dec)` qui effectue la multiplications des polynômes représentés par les tableaux `P[debP:finP]` et `Q[debQ:finQ]` *sans créer ces tableaux*, et écrit le résultat dans le tableau `PQ` avec un décalage de `dec` On remarquera et utilisera que les polynômes $P^{(0)}Q^{(0)}$ et $P^{(1)}Q^{(1)}X^{2^k}$ n'ont aucun monôme de même degré.

(Bonus) Notre version de l'algorithme de Karatsuba effectue encore beaucoup trop d'opérations à cause de la conversion des polynômes de degrés n en polynômes de degré $2^{\lceil \log_2(n+1) \rceil} - 1$. Modifier l'algorithme de façon à traiter directement les polynômes de degrés arbitraires sans les dilater de cette façon. (Il est normalement plus facile de partir de l'algorithme le plus optimisé)

(Bonus) En sachant pour quels degrés la méthode de Karatsuba est plus rapide que le produit naïf, écrire un algorithme de multiplication légèrement plus efficace que les algorithmes déjà écrits en les combinant. Quel est l'ordre du gain ?