DS 2

CPES « Sciences des données, arts et cultures » : Introduction à l'algorithmique

Antoine GAUQUIER antoine.gauquier@ens.fr
Pierre SENELLART pierre@senellart.com

16 janvier 2024

L'examen dure deux heures, comporte un unique problème et 10 questions. Les documents sont autorisés, mais limités à 10 feuilles recto-verso par personne. Aucun appareil numérique n'est autorisé, donc, de façon non-exhaustive : pas d'ordinateur, pas de téléphone, pas de montre connectée, etc. Pas de calculatrice non plus. Pour les questions contenant du code Python, les erreurs mineures de syntaxe Python ne seront pas pénalisées, tant que le code écrit ressemble à un programme Python. Veiller à indiquer votre nom sur l'ensemble des copies rendues, et à les numéroter le cas échéant.

Dans tout le sujet, quand on parlera de complexité, il s'agira de complexité algorithmique en temps, donnée avec un $\Theta()$ si possible ou un $O()$ si on a seulement une borne supérieure.

Multiplication de polynômes

On note $\mathbb{Z}[X]$ l'ensemble des polynômes dont les coefficients sont des entiers relatifs. Tout polynôme P de $\mathbb{Z}[X]$ non nul peut s'écrire de manière unique sous la forme $P(X) = \sum_{i=0}^{d} c_i X^i$ avec c_i entier relatif et $c_d \neq 0$. Le *degré* de P est d.

Par exemple, le polynôme $X^3 + X^2 + 1$ est de degré 3 et la liste de ses coefficients est $c_0 = 1$, $c_1 = 0$, $c_2 = 1$, $c_3 = 1$. Par convention, le polynôme nul P_0 se note $P_0(X) = 0$ et son degré est de -1.

Dans tout le problème, on considérera que les nombres entiers manipulés sont de taille constante et donc que les opérations arithmétiques sur les nombres entiers se font en $O(1)$.

L'objet de ce problème est d'étudier différentes méthodes pour calculer le résultat de la multiplication de deux polynômes de $\mathbb{Z}[X]$.

1. (2 points) On souhaite représenter un polynôme de $\mathbb{Z}[X]$ par une classe Python `Polynome` comportant deux propriétés :

   `_degre`, le degré du polynôme ;

   `_coeff`, le tableau des coefficients représenté sous forme de liste Python.

   Écrire le constructeur d'une telle classe, qui prend en argument un tableau de coefficients. Ainsi, `Polynome([1,0,1,1])` créera un objet représentant le polynôme $X^3 + X^2 + 1$.

2. (1 point) Ajouter à cette classe une méthode magique `__getitem__` permettant d'accéder au i-ème coefficient du polynôme P en écrivant `P[i]`.

3. (2 points) De même que la méthode magique `__getitem__` permet de définir l'indexation d'un objet `P` d'une classe avec des crochets (`P[i]`), la méthode magique `__call__` permet de définir la manière dont utiliser un objet `P` comme une fonction avec des parenthèses, de la forme `P(x)`.

   On souhaite ainsi ajouter à la classe `Polynome` une méthode magique `__call__` permettant d'utiliser un objet de classe polynôme comme une fonction afin d'évaluer un polynôme en un

point : `P(42)` doit retourner la valeur du polynôme P en le point 42. Pour effectuer ce calcul sans avoir besoin de calculer des puissances, on s'appuie sur la *méthode de Horner* qui consiste à réécrire le polynôme de la manière suivante :

$$c_0 + c_1 x + c_2 x^2 + \cdots + c_n x^{n-1} + c_n x^n = c_0 + x(c_1 + x(c_2 + x(c_3 + \cdots + x(c_{n-1} + x c_n) \ldots)))$$

Implémenter une telle fonction `__call__` basée sur la méthode de Horner. Quelle est sa complexité ?

4. (3 points) Ajouter à cette classe une méthode magique `__str__` permettant d'afficher le polynôme sous une forme lisible en listant les monômes par ordre décroissant (p. ex., « `X^3 + X^2 + 1` » pour le polynôme $X^3 + X^2 + 1$ ou bien « `X^5 - X^4 - X^3 + 2X^2 - 2X + 1` » pour le polynôme $X^5 - X^4 - X^3 + 2X^2 - 2X + 1$). Veiller à traiter soigneusement les cas particuliers (par exemple, le polynôme nul).
5. (2 points) Ajouter à cette classe une méthode magique `__add__` permettant de calculer le polynôme résultant de l'addition des deux polynômes en entrée. Quelle est la complexité de cette méthode ? Grâce à cette méthode, le code suivant :

   ```
   P = Polynome([1,-2,1])
   Q = Polynome([1,0,1,1])
   R = P+Q
   print(R)
   ```

   doit produire l'affichage « `X^3 + 2X^2 - 2X + 2` ».
6. (1 point) Ajouter à cette classe une méthode magique `__sub__` permettant de calculer le polynôme résultant de la soustraction des deux polynômes en entrée. Quelle est la complexité de cette méthode ? Grâce à cette méthode, le code suivant :

   ```
   P = Polynome([1,-2,1])
   Q = Polynome([1,0,1,1])
   R = P-Q
   print(R)
   ```

   doit produire l'affichage « `- X^3 - 2X` ».
7. (2 points) Ajouter à cette classe une méthode magique `__mul__` permettant de calculer le polynôme résultant de l'addition des deux polynômes en entrée, *le plus simplement possible, sans chercher à optimiser ce calcul* ; on appellera cette méthode l'algorithme **N**. Quelle est la complexité de cette méthode ? Grâce à cette méthode, le code suivant :

   ```
   P = Polynome([1,-2,1])
   Q = Polynome([1,0,1,1])
   S = P*Q
   print(S)
   ```

   doit produire l'affichage « `X^5 - X^4 - X^3 + 2X^2 - 2X + 1` ».
8. (2 points) On considère l'algorithme suivant (appelé algorithme **B**) pour calculer la multiplication de deux polynômes P et Q de $\mathbb{Z}[X]$ dont le degré maximal est d :
   - Si $d = 0$, on multiplie P et Q comme deux nombres entiers.
   - Sinon, soit $m = \lceil \frac{d}{2} \rceil$ (où $\lceil \cdot \rceil$ dénote la fonction *partie entière supérieure*) ; on écrit P et Q comme, chacun, la somme de deux polynômes :

$$\begin{cases} P = P_2 X^m + P_1 \\ Q = Q_2 X^m + Q_1 \end{cases}$$

avec P_1, P_2, Q_1, Q_2 des polynômes de degré $< m$. On pose alors :

$$\begin{cases} R_1 = P_1 \times Q_1 \\ R_{12} = P_2 \times Q_1 + P_1 \times Q_2 \\ R_2 = P_2 \times Q_2 \end{cases}$$

et on calcule $P \times Q$ comme $R_2 X^{2m} + R_{12} X^m + R_1$ en appliquant l'algorithme récursivement.

Calculer la complexité asymptotique de l'algorithme **B** et la comparer à celle de l'algorithme **N**. Comme d'habitude on pourra faire l'hypothèse pour simplifier l'analyse que les parties entières tombent toujours juste. On ne demande pas d'implémenter `B`.

9. (2 points) En développant $(P_1 + P_2) \times (Q_1 + Q_2)$, proposer une autre manière de calculer R_{12} (avec les notations de la question précédente). En déduire un autre algorithme (que l'on appellera algorithme **K**) pour le calcul du produit de deux polynômes. Calculer la complexité asymptotique de l'algorithme **K** et la comparer à celle des algorithmes **N** et **B**. Comme d'habitude on pourra faire l'hypothèse pour simplifier l'analyse que les parties entières tombent toujours juste.
10. (3 points) Ajouter à la classe `Polynome` une méthode statique `algok` permettant d'utiliser l'algorithme **K** pour calculer le produit de deux polynômes passés en argument. (On utilisera donc cette méthode avec `Polynome.algok(P,Q)`.)

    Remarque finale : il est en fait possible d'avoir une complexité bien meilleure que celle des techniques vues dans ce problème : plus précisément, la multiplication de deux polynômes de degré n peut se faire en $\Theta(n \log n)$, mais l'approche est assez complexe. Pour plus de détails, voir le chapitre 30 du livre de référence.