(PSI*)

ÉCOLE POLYTECHNIQUE

CONCOURS D'ADMISSION 2010 **FILIÈRES MP ET PC**

ÉPREUVE D'INFORMATIQUE

(Durée : 2 heures)

L'utilisation des calculatrices **n'est pas autorisée** pour cette épreuve.
Le langage de programmation choisi par le candidat doit être spécifié en tête de copie.

⋆ ⋆ ⋆

Échangeurs de Polynômes

Dans ce problème on s'intéresse à des polynômes à coefficients réels qui s'annulent en 0. Un tel polynôme P s'écrit donc $P(x) = a_1x + a_2x2 + \ldots + a_mx^m$. Le but de ce problème est d'étudier la position relative autour de l'origine de plusieurs polynômes de ce type.

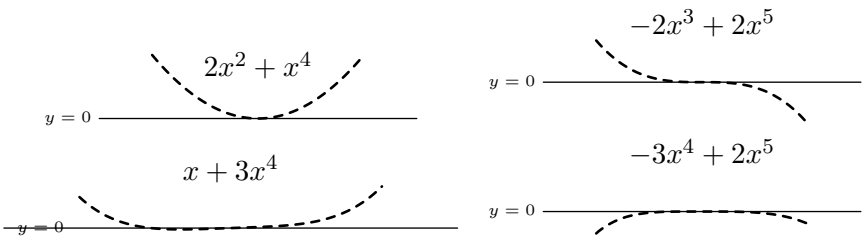
Dans tout le problème, les polynômes sont représentés par des tableaux de nombres flottants de la forme $[a_1; a_2; \ldots; a_m]$. Le nombre a_m peut être nul, par conséquent un polynôme donné admet plusieurs représentations sous forme de tableau. Ces tableaux sont indexés à partir de 1 et les éléments d'un tableau de taille m sont donc indexés de 1 à m. On suppose qu'il existe également une primitive `allouer(m)` pour créer un tableau de m cases. La taille m d'un tableau t est renvoyée par la primitive `taille(t)`. L'accès à la $i^{ème}$ case d'un tableau t est noté $t[i]$. Par ailleurs, on suppose que les tableaux peuvent être passés en argument ou renvoyés comme résultat de fonction, quel que soit le langage utilisé par le candidat pour composer. Enfin, les booléens `vrai` et `faux` sont utilisés dans certaines questions de ce problème. Le candidat est libre d'utiliser les notations propres à ces booléens dans le langage dans lequel il compose.

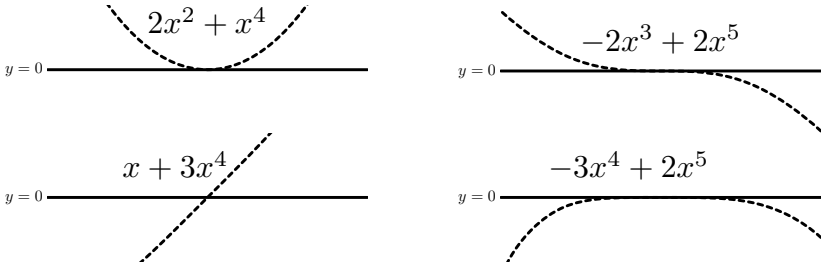
Le problème est découpé en deux parties qui peuvent être traitées de manière indépendante. Cependant, la **partie II** utilise les notions et notations introduites dans la **partie I**.

1 I. Permutation de n polynômes

Question 1 Afin de se familiariser avec cette représentation, écrire une fonction `evaluation` qui prend en arguments un polynôme P, représenté par un tableau, et un nombre flottant v, et qui renvoie la valeur de $P(v)$.

Nous commençons notre étude par quelques observations. Voici des exemples de graphes de polynômes autour de l'axe des abscisses.



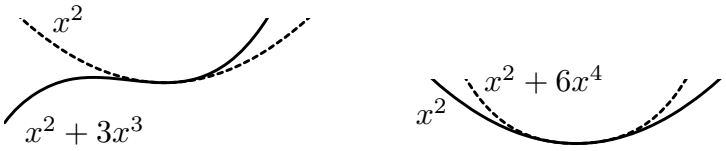


On remarque que le comportement au voisinage de l'origine est décrit par le premier monôme $a_k x^k$ dont le coefficient a_k est non nul (les coefficients $a_1, ..., a_{k-1}$ étant donc tous nuls). En effet, quand x est petit, le terme $a_{k+1}x_{k+1} + \ldots + a_m x^m$ est négligeable devant le terme $a_k x^k$. Cet entier k est la valuation du polynôme à l'origine. Par exemple, la valuation du polynôme $-2x^3 - 3x^5 + 4x^7$ est 3. On remarque alors les deux règles suivantes au voisinage de l'origine :

– Si la valuation k est paire, le graphe du polynôme reste du *même* côté de l'axe des abscisses.
– Si la valuation k est impaire, le graphe du polynôme *traverse* l'axe des abscisses.

Question 2 Écrire une fonction `valuation` qui prend en argument un polynôme P et renvoie sa valuation. Par définition, cette fonction renverra 0 si P est le polynôme nul.

On s'intéresse maintenant aux positions relatives autour de l'origine des graphes de deux polynômes P_1 et P_2. La figure suivante montre les graphes de polynômes autour de l'origine.



On remarque que le comportement de ces graphes dépend de la parité de la valuation de la différence $P_1 - P_2$:

– Si la valuation de $P_1 - P_2$ est paire, les deux graphes se touchent mais ne se traversent pas à l'origine.
– Si la valuation de $P_1 - P_2$ est impaire, les deux graphes se traversent à l'origine.

Question 3 Écrire une fonction difference qui prend en arguments deux polynômes P_1 et P_2 (dont les tailles peuvent être différentes) et qui renvoie la différence des polynômes $P_1 - P_2$.

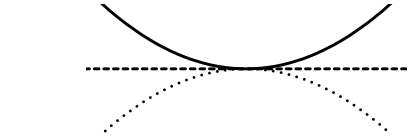
Question 4 Écrire une fonction `compare_neg` qui prend en arguments deux polynômes P_1 et P_2 et qui renvoie :

– un entier strictement négatif si $P_1(x)$ est plus petit que $P_2(x)$, pour x négatif assez petit
– 0 si les deux polynômes P_1 et P_2 sont égaux
– un entier strictement positif si $P_1(x)$ est plus grand que $P_2(x)$, pour x négatif assez petit.

On admettra sans démonstration que la fonction `compare_neg` définit une relation d'ordre.

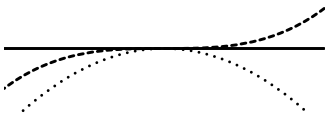
Enfin, passons à l'étude des graphes de trois polynômes. Les figures ci-après montrent les positions relatives de trois polynômes P_1, P_2 et P_3 autour de l'origine, avec la légende suivante :

$P_1(x)$ —— $P_2(x)$ ------- $P_3(x)$ ·····

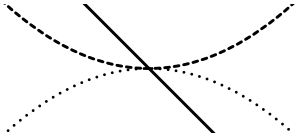


$P_1(x) = x^2 \quad P_2(x) = 0 \quad P_3(x) = -x^2$

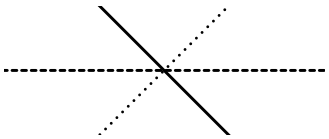
$P_1(x) = x^2 \quad P_2(x) = -x^3 \quad P_3(x) = 0$



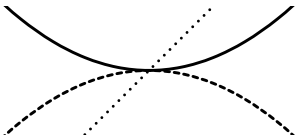
$P_1(x) = 0 \quad P_2(x) = x^3 \quad P_3(x) = -x^2$



$P_1(x) = -x \quad P_2(x) = x^2 \quad P_3(x) = -x^2$



$P_1(x) = -x \quad P_2(x) = 0 \quad P_3(x) = x$



$P_1(x) = x^2 \quad P_2(x) = -x^2 \quad P_3(x) = x$

Le choix de ces polynômes est fait pour qu'à chaque fois les inégalités $P_1(x) > P_2(x) > P_3(x)$ soient vérifiées pour x légèrement négatif. Maintenant, observons les positions relatives de ces graphes pour x légèrement positif. On remarque que l'ordre des courbes est permuté : on passe de l'ordre $P_1(x) > P_2(x) > P_3(x)$ à un autre ordre. La donnée des trois polynômes P_1, P_2 et P_3 définit donc une unique permutation π de $\{1, 2, 3\}$ telle que $P_{\pi(1)}(x) > P_{\pi(2)}(x) > P_{\pi(3)}(x)$, pour x positif et assez petit. On note que les six permutations de $\{1, 2, 3\}$ sont possibles, comme le montrent les six exemples ci-dessus.

De manière générale, on dit qu'une permutation π de $\{1, 2, ..., n\}$ permute les polynômes $P_1, P_2, \ldots, P_n$ si et seulement si :

$$P_1(x) > P_2(x) > \ldots > P_n(x) \qquad \text{pour } x \textit{ négatif} \text{ assez petit}$$

$$\text{et} \quad P_{\pi(1)}(x) > P_{\pi(2)}(x) > \ldots > P_{\pi(n)}(x) \quad \text{pour } x \textit{ positif} \text{ assez petit.}$$

Ce qui était vrai pour trois polynômes ne l'est plus à partir de quatre polynômes : il existe des permutations qui ne permutent aucun ensemble de polynômes $P_1, P_2, \ldots, P_n$.

Dans la suite, les permutations de $\{1, 2, \ldots, n\}$ seront représentées par des tableaux d'entiers de taille n, indexés à partir de 1 et contenant tous les entiers entre 1 et n.

Question 5 Écrire une fonction `tri` qui prend en argument un tableau t contenant n polynômes et qui le trie en utilisant la fonction `compare_neg`, de telle sorte que l'on ait $t[1](x) > t[2](x) > \ldots > t[n](x)$ pour x négatif et assez petit. Le candidat ne pourra pas utiliser pour cette question de fonction de tri prédéfinie dans la bibliothèque du langage qu'il utilise pour composer.

Question 6 Écrire une fonction `verifier_permute` qui prend en argument une permutation π de $\{1, 2, ..., n\}$ et un tableau t de même taille supposé trié par la fonction tri, et renvoie `vrai` si π permute les n polynômes $t[1], t[2], \ldots, t[n]$ contenus dans t, et `faux` sinon. On pourra s'aider d'une fonction `compare_pos`, similaire à la fonction `compare_neg`, pour comparer deux polynômes pour x *positif* assez petit.

2 II. Échangeurs de n polynômes

Dans la suite, nous dirons qu'une permutation π de $\{1, 2, ..., n\}$ est un *échangeur* s'il existe n polynômes $P_1, P_2, ..., P_n$ tels que π permute ces polynômes. Nous allons maintenant écrire des fonctions qui répondent aux questions suivantes : Une permutation π est-elle un échangeur ? Peut-on dénombrer les échangeurs ? Peut-on énumérer les échangeurs ?

Une condition nécessaire et suffisante pour qu'une permutation soit un échangeur est la suivante : une permutation π de $\{1, 2, \ldots, n\}$ est un échangeur si et seulement si il n'existe aucun entiers $a,\ b,\ c,\ d$ tels que $n \geqslant a > b > c > d \geqslant 1$ et

$$\pi(b) > \pi(d) > \pi(a) > \pi(c) \text{ ou } \pi(c) > \pi(a) > \pi(d) > \pi(b) \tag{1}$$

Question 7 Écrire une fonction `est_echangeur_aux` qui prend en argument une permutation π de $\{1, 2, \ldots, n\}$ et un entier d tel que $1 \leqslant d \leqslant n$ et qui renvoie `vrai` s'il n'existe aucun entier $a,\ b$ et c tels que $n \geqslant a > b > c > d$ et vérifiant (1), et `faux` sinon.

Question 8 En utilisant la fonction `est_echangeur_aux`, écrire une fonction `est_echangeur` qui prend en argument une permutation π de $\{1, 2, \ldots, n\}$ et renvoie `vrai` si π est un échangeur, et `faux` sinon.
On admet sans démonstration que la relation de récurrence suivante permet de compter le nombre $a(n)$ de permutations de $\{1, 2, \ldots, n\}$ qui sont des échangeurs :

$$a(1) = 1, \qquad a(n) = a(n-1) + \sum_{i=1}^{n-1} a(i) \times a(n-i)$$

Question 9 Écrire une fonction `nombre_echangeurs` qui prend un entier n en argument et renvoie le nombre d'échangeurs $a(n)$. Enfin, les deux questions suivantes ont pour but d'énumérer tous les échangeurs de $\{1, 2, \ldots, n\}$.

Question 10 Écrire une fonction `decaler` qui prend en arguments un tableau t de taille n et un entier v, et renvoie un nouveau tableau u de taille $n + 1$ tel que

$$\begin{cases} u[1] = v \\ u[i] = t[i-1] & \text{si } t[i-1] < v \text{ et } 2 \leqslant i \leqslant n+1 \\ u[i] = 1 + t[i-1] & \text{si } t[i-1] \geqslant v \text{ et } 2 \leqslant i \leqslant n+1 \end{cases}$$

L'algorithme que nous allons utiliser pour énumérer les échangeurs de $\{1, 2, \ldots, n\}$ consiste à énumérer successivement les échangeurs de $\{1, 2, \ldots, k\}$, pour tout k de 1 à n, dans un tableau t de taille $a(n)$. Si on suppose qu'un tableau t contient les m échangeurs de $\{1, \ldots, k\}$ entre les cases $t[1]$ et $t[m]$, on peut en déduire les échangeurs de $\{1, \ldots, k+1\}$ de la manière suivante : pour tout entier v entre 1 et $k+1$ et tout entier i entre 1 et m, on décale (à l'aide de la fonction `decaler`) l'échangeur $t[i]$ avec v puis on teste si le résultat est un échangeur (avec la fonction `est_echangeur_aux`).

Question 11 Écrire une fonction `enumerer_echangeurs` qui prend un entier n en argument et renvoie un tableau contenant les $a(n)$ échangeurs de $\{1, 2, \ldots, n\}$. On pourra utiliser un second tableau pour stocker temporairement les nouveaux échangeurs.