TP n°2

Rappels Java : tableaux et objets

Dans le premier TP, vous avez vu comment resoudre des equations bi-carrees. Le polynôme etait represente par un objet comportant trois variables d'instances pour conserver les trois coe cients. Malheureusement, tous les polynômes ne sont pas de degres 4, avec uniquement les puissances paires de x ! Nous allons explorer dans ce TPs plusieurs facons de representer des polynômes arbitraires.

On ne tentera pas ici de trouver les racines de ces polynômes de degres quelconques puisque Galois a montre qu'au dela du degre 5, on ne peut les trouver de facon generale. On se contentera donc d'evaluations en un point, d'addition, multiplication, etc.

Exercice 1 *Implémentation naïve*

Un polynôme sera represente par le tableau de ses coe cients. C'est a dire qu'un polynôme de degre n est un tableau `coeffs` de taille $n + 1$, ou pour tout indice i, `coeffs[i]` est le coe cient du monome x^i

1. Ecrire une classe publique `Polynome`, avec propriete privee `coeffs` de type `double[]`. Ajouter un constructeur (attention, reportez vous au TD1 pour voir quoi faire du tableau passe en argument au constructeur !), ainsi que trois methodes
   ```
   String toString() ;
   int getDegre() ;
   double getCoeff(int i)
   ```

   Votre methode `getDegre` est elle toujours correctes ?

   Ne passez pas trop de temps a peau ner la methode `toString`, elle est surtout la pour vous aider a debuguer.
2. Ecrire une methode `void multParConst(double c)` qui multiplie le polynôme par une constante
3. Ecrire une methode `void multParMonome(int i)` qui multiplie le polynôme par x^i
4. Ecrire une methode na•ve `double eval(double x)`

6. Faire pareil pour la soustraction et la multiplication. N'hesitez pas a ecrire la multiplication de petits polynômes sur une feuille de papier pour bien comprendre ce qu'il se passe !

Exercice 2 *Retour sur l'évaluation en un point*

Lorsque l'on manipule des ottants en informatique, la multiplication est une operation assez coûteuse (elle prend nettement plus de temps qu'une addition par exemple). Il est donc bon de limiter au maximum le nombre de multiplication dans un programme. Dans le cas de la fonction eval de tout a l'heure, avec une implementation na•ve de la puissance i (avec $i-1$ multiplication), on arrive avec un polynôme de degre n et un coe cient non nul pour chaque indice a $n*(n+1)$=2 multiplications (exercice facultatif, retrouver ce resultat !). On va voir que l'on peut facilement descendre a n multiplication grâce a l'algorithme de Horner.

L'idee est de constater que le polynôme $a*x^3+b*x^2+c*x+d$ peut s'ecrire $((a*x+b)*x+c)*x+d$

Developpez le second polynôme pour vous en convaincre, et generalisez a un polynôme de degre quelconque. En deduire une methode `double evalHorner(double x)` ;

Exercice 3 *Polynômes creux*

Il arrive tres souvent que les polynômes que l'on manipule soient creux, c'est a dire que la grade majorite des coe cients sont nuls. Par exemple, $3*x^{512}+51*x^{42}+8*x$ est un tel polynôme creux. Dans la representation de la partie precedente, un tel polynôme demande un tableau de taille 513 pour seulement trois coe cients !

Une solution est de ne stocker que les coe cients non nuls. Pour cela, vous pouvez utiliser la classe suivante

```
public class IndiceCoeff{
 public int indice;
 public double coeff;

 public IndiceCoeff(int i; double c){
   indice = i;
   coeff = c;
 }
}
```

1. Ecrivez une classe `PolynomeCreux` avec une propriete privee `indicesCoeffs` de type `IndiceCoeff`. On supposera que les paires passees en argument au constructeurs (dans un tableau) sont triees par ordre croissant d'indice, et on veillera a systematiquement maintenir cet invariant. (on peut eventuellement le veri er, et lancer une exception s'il n'est pas valide).
2. Sur les polynômes creux, ecrivez les methodes `String toString()` ;
   `int getDegre()` ;
   `double getCoeff(int i)`
   `void multParConst(double c)`
   `void multParMonome(int i)`

   Pour `getCoeff`, une recherche par dichotomie serait la bienvenue !

3. Ecrire une methode `void normalise()` qui supprime toutes les paires telles que le coe - cient est nul. Apres un appel a normalise, le tableau indicesCoe s sera de taille minimale
4. Pour la methode `addition`, on pourra commencer par allouer un tableau de taille `t1.length + t2.length`, le remplir, puis normaliser le polynôme.
5. Ecrire de même la multiplication de deux polynômes creux.
6. Adaptez l'algorithme de Horner pour les polynômes creux. On pourra utiliser un algorithme d'exponentiation rapide (Vous ne connaissez pas ? `http://tinyurl.com/ybbwblz`) pour les coe cients d'indices "eloignes".