Manipulation de polynômes

L'objet du TP est d'étudier une implémentation des polynômes à une variable et des opérations élémentaires sur les polynômes. Un polynôme :

$$P = a_0 + a_1X + a_2X^2 + \ldots + a_nX^n$$

sera représenté en machine par le vecteur Caml :

$$\texttt{p} = [|a_0; a_1; a_2; \ldots; a_n|].$$

On se contentera de manipuler des polynômes à coefficients entiers donc $a_0, \ldots, a_n$ seront de type `int` et `p` de type `int vect`.

1) Addition, soustraction, multiplication

Si $P = \sum_{i=0}^{d_p} a_iX^i$ et $Q = \sum_{i=0}^{d_q} b_iX^i$ alors $P \pm Q = \sum_{i=0}^{\max(d_p,d_q)} (a_i \pm b_i)X^i$ en convenant que les a_i ou b_i manquants valent zéro. Pour implémenter ceci en Caml la méthode la plus simple (mais peut-être pas la plus efficace) consiste à définir un vecteur résultat, `r`, suffisament long initialisé à zéro, à y recopier `p` puis à lui ajouter ou retrancher `q`. Voici le code d'une fonction `++` additionnant deux polynômes :

```
(* addition *)
let prefix ++ p q =
  let lp = vect_length(p)
  and lq = vect_length(q)         in
  let r = make_vect (max lp lq) 0 in
  for i=0 to lp-1 do r.(i) <- p.(i) done;
  for i=0 to lq-1 do r.(i) <- r.(i) + q.(i) done;
  r
;;

(* essai *)
let p = [| 1; 2; 3   |]     (* 1 + 2x + 3x^2 *)
and q = [| 4; 5; 6; 7|]     (* 4 + 5x + 6x^2 + 7x^3 *)
;;

p ++ q;;
```

On écrit `let prefix ++ p q =`... et non `let ++ p q =`... ni `let p ++ q =`... car `++` est un identificateur légal en Caml mais qui doit figurer en position infixe (c'est à dire entre ses arguments) du fait que son nom commence par un symbole opératoire (`+`). Le mot-clé `prefix` permet d'utiliser le symbole qui suit en position préfixe ce qui est obligatoire dans une définition Caml.

Recopier ce code, définir de même la soustraction et la multiplication des polynômes qui seront notées `--` et `**`. Tester vos fonctions avec les polynômes `p` et `q` définis ci-dessus ou avec d'autres.

2) Les polynômes de Chebychev

Il s'agit d'une suite de polynômes définis par les relations :

$$T_0 = 1, \quad T_1 = X, \quad T_n = 2XT_{n-1} - T_{n-2} \text{ si } n \geqslant 2.$$

Programmer le calcul du n-ème polynôme de Chebychev à l'aide des opérations `++`, `--` et `**` définies précédemment. Vous pouvez au choix utiliser un vecteur `t` de polynômes avec `t.(i)` $= T_i$ les `t.(i)` étant calculés de proche en proche ou, mais c'est plus compliqué à implémenter, n'utiliser que deux polynômes variables (donc en fait des références sur des polynômes) contenant les deux derniers polynômes calculés et faire « évoluer » ces variables pour calculer les T_i de proche en proche. Quelque soit votre choix, votre fonction `chebychev : int -> int vect` ne devra retourner que le polynôme dont l'indice est fourni en argument.

3) Composition

Si $P = \sum_{i=0}^{d_p} a_i X^i$ et Q est un polynôme quelconque alors $P \circ Q = \sum_{i=0}^{d_p} a_i Q^i$. En particulier si a est un entier, $P(a) = P \circ a$ où le deuxième a désigne le polynôme constant aX^0. La composition des polynômes est une opération de même nature que l'évaluation en un point et les algorithmes d'évaluation peuvent être utilisés pour calculer une composée, en particulier l'algorithme de Hörner. Programmer cette opération de composition en Caml, elle sera notée `compose`. L'utiliser pour calculer :

`compose [|0;0;0;0;0;0;1|] [|1;1|]`

et expliquer quel est le résultat obtenu.

4) Élimination des zéros de tête

La représentation d'un polynôme par un vecteur à coefficients entiers est ambiguë car plusieurs vecteurs peuvent représenter le même polynôme, il suffit d'ajouter des zéros « non significatifs » au delà du coefficient dominant. Cela peut être gênant en particulier pour l'affichage des polynômes inutilement brouillé par ces zéros non significatifs mais aussi pour le temps d'exécution des opérations de base, en particulier de la multiplication qui peuvent être fortement ralenties par ces zéros parasites. Vous allez donc écrire une fonction Caml `réduit` qui retourne la forme réduite d'un polynôme, c'est à dire le vecteur de longueur minimale représentant ce polynôme ; le dernier coefficient doit être non nul sauf pour le polynôme nul qui sera par convention représenté par le vecteur `[|0|]`.

Affichage amélioré

Malgré la fonction de réduction définie à la question précédente, Caml persiste à afficher les polynômes de façon déplaisante comme si c'étaient de vulgaires vecteurs d'entiers. Un affichage plus « mathématique » serait souhaitable, voici une fonction que vous pouvez recopier et qui améliore la présentation des polynômes (en fait de toute valeur de type `int vect`).

```
(* affichage *)
#open "format";;
let affiche(p) =
  let lp = vect_length(p)
  and premier = ref(true) in
  for i=0 to lp-1 do
    if p.(i) <> 0 then begin
      if p.(i) < 0 then print_string " - "
      else if not(!premier) then print_string " + ";
      if i=0 or abs(p.(i)) <> 1 then print_int(abs(p.(i)));
      premier := false;
      match i with
        | 0 -> ()
        | 1 -> print_string "x"
        | _ -> print_string "x^"; print_int i
    end
  done;
  if !premier then print_string "0"
;;

install_printer "affiche";;
chebychev(10);;
```

C'est un excellent exercice que de chercher à comprendre comment « marche » cette fonction.

5) Division euclidienne

La division euclidienne du polynôme A par le polynôme B, A et B étant à cofficients entiers, est possible dans $\mathbb{Q}$ si $B \neq 0$. Si de plus le coefficient dominant de B vaut ± 1 alors le quotient Q et le reste R sont eux aussi à coefficients entiers (cette condition est seulement suffisante). Écrire une fonction `division` qui calcule le couple (Q, R) à partir de A et B en vérifiant que le coefficient dominant de B vaut effectivement ± 1 et en déclenchant une erreur dans le cas contraire. On utilisera l'algorithme de calcul vu en cours de maths.

6) Polynômes cyclotomiques (pour les forts)

Il s'agit d'une suite de polynômes définis par la formule de récurrence :

$$C_1 = X - 1, \quad C_n = \frac{X^n - 1}{\prod_{i|n} C_i} \text{ si } n \geqslant 2$$

où le produit porte sur tous les indices $i \in [\![1, n-1]\!]$ tels que i divise n. Par exemple $C_2 = (X^2 - 1)/C_1 = X + 1$, $C_3 = (X^3 - 1)/C_1 = X^2 + X + 1$, $C_4 = (X^4 - 1)/(C_1 C_2) = X^2 + 1$ etc. On démontre en considérant le groupe des racines n-èmes de l'unité dans $\mathbb{C}$ que toutes les divisions sont exactes et que C_n est un polynôme à coefficients entiers de coefficient dominant 1 pour tout n. Programmer le calcul du n-ème polynôme cyclotomique en utilisant la même technique que pour le calcul des polynômes de Chebychev : un vecteur de polynômes que l'on remplit de proche en proche, seuls les polynômes C_i tels que i divise n doivent être calculés, les autres peuvent être « laissés en blanc ». Il existe des méthodes plus rapides mais pour les bas degrés celle-là suffira. Au vu des premiers polynômes on peut penser que les coefficients de C_n valent toujours 0, 1 ou -1 ; c'est faux et vous êtes invités à chercher quel est le premier contre-exemple.