Épreuve d'informatique de l'X - 2010 - MP/PC

Échangeurs de Polynômes

Question 1. On évalue le polynôme P en utilisant la méthode de Horner pour limiter le nombre de multiplications.

```
1  evaluation := proc (P::array, v::float)
2     local res, i, n;
3     n := taille(P);
4     res := 0;
5     for i from n to 1 by -1 do
6        res := (res+P[i])*v
7     od;
8     return(res)
9     end;
```

Question 2. On parcourt le tableau représentant le polynôme avec la boucle des lignes 3-5 et on retourne la première valeur pour laquelle $a_i \neq 0$. Si P est le polynôme nul, on retourne la valeur 0.

```
1  valuation := proc (P::array)
2     local i;
3     for i from 1 to taille(P) do
4       if P[i] <> 0 then return(i)fi;
5     od;
6     return(0);
7     end;
```

Question 3 : On commence par créer un tableau Q de taille $\max(n_1, n_2)$ où n_1 et n_2 représentent les tailles respectives de P_1 et P_2. On l'initialise avec des zéros (ligne 7). On duplique le polynôme P_1 dans Q (ligne 8) et enfin on lui soustrait P_2 (ligne 8).

```
1  difference := proc (P1::array, P2::array)
2  local Q, i, n1, n2, n;
3    n1 := taille(P1);
4    n2 := taille(P2);
5    if n2 < n1 then n := n1 else n := n2 end if;
6    Q := allouer(n);
7    for i from 1 to n do Q[i] := 0 od;
8    for i from 1 to n1 do Q[i] := P1[i] od;
9    for i from 1 to n2 do Q[i] := Q[i]-P2[i] od;
10   return(Q)
11 end;
```

Question 4 : Si $P_1 = \sum_{k=1}^{d_1} a_k x^k$ et $P_2 = \sum_{k=1}^{d_2} b_k x^k$ et que $P_1 - P_2$ n'est pas le polynôme nul et a pour valuation v, alors $P_1(x) - P_2(x) \sim (a_v - b_v).x^v$. Donc à gauche de zéro, $P_1 - P_2$ est localement du signe de $(a_v - b_v)$ si v est pair et du signe contraire de $(a_v - b_v)$ si v est impair.

```
1  compare_neg := proc (P1::array, P2::array)
2    local Q, v;
3    Q := difference(P1, P2);
4    v := valuation(Q);
5    if v = 0 then return 0 fi;
6    if v mod 2 = 0
7       then
8         if 0 < Q[v] then return  1 else return -1 fi;
9       else
10        if 0 < Q[v] then return -1 else return  1 fi;
11   fi;
12   end;
```

Question 5 : on effectue un tri à bulles sur le tableau de polynômes.

```
echange := proc (t::array, i::posint, j::posint)
local tmp;
  tmp := t[i]; t[i] := t[j]; t[j] := tmp; return ()
end;

tri := proc (t::array)
local n, i, j;
  n := taille(t);
  for i from 1 to n-1 do
     for j from 1 to n-i do
        if compare_neg(t[j], t[j+1]) < 0
           then echange(t, j, j+1)
        fi;
     od;
  od;
 end;
```

Question 6 : Si $P_1 = \sum_{k=1}^{d_1} a_k x^k$ et $P_2 = \sum_{k=1}^{d_2} b_k x^k$ et que $P_1 - P_2$ n'est pas le polynôme nul et a pour valuation v, alors $P_1(x) - P_2(x) \sim (a_v - b_v).x^k$. Donc à droite de zéro, $P_1 - P_2$ est localement du signe de $(a_v - b_v)$.

```
compare_pos := proc (P1::array, P2::array)
 local Q, v;
  Q := difference(P1, P2);
  v := valuation(Q);
  if v = 0 then return 0 end if;
  if 0 < Q[v] then return 1 else return -1 end if;
end;

verifier_permute := proc (pi::array, t::array)
 local n, i; n := taille(pi);
 for i from 1 to n-1 do
  if not(compare_pos(t[pi[i]], t[pi[i+1]]))
        then return(false)
  fi;
 od;
 return(true)
 end;
```

II. Échangeurs de n polynômes

Question 7 : Les trois boucles imbriquées du programme permettent de tester tous les quadruplets (a, b, c, d) qui vérifient $1 \leqslant d \leqslant c \leqslant b \leqslant a \leqslant n$.

```
est_echangeur_aux := proc (pi::array, d::posint)
local a, b, c, n;
n := taille(pi);
for c from d+1 to n-2 do
   for b from c+1 to n-1 do
      for a from b+1 to n do
         if pi[c] < pi[a] and pi[a] < pi[d]
                          and pi[d] < pi[b] then return(false) fi;
         if pi[b] < pi[d] and pi[d] < pi[a]
                          and pi[a] < pi[c] then return(false) fi;
      od;
   od;
od;
return(true);
end;
```

Question 8 :

```
est_echangeur := proc (pi::array)
local d, n;
n := taille(pi);
for d from 1 to n do
   if not(est_echangeur_aux(pi, d))
       then return(false)
   fi;
od;
return(true);
end;
```

Question 9 : On calcule les valeurs $a(k)$ de façon itérative à l'aide d'un tableau à partir de la valeur connue $a(1) = 1$ (ligne 4). On calcule ensuite $a(i)$ à partir de $a(j)$ pour $1 \leqslant j \leqslant i-1$. On initialise la valeur à $a(i-1)$ à la ligne 6 et on ajoute $\sum_{j=1}^{i-1} a_j a_{i-j}$ grâce à la boucle des lignes 7 à 9.

```
1  nombre_echangeurs := proc (n::posint)
2  local tab, i, j;
3  tab := allouer(n);
4  tab[1] := 1;
5  for i from 2 to n do
6    tab[i] := tab[i-1];
7    for j from 1 to i-1 do
8       tab[i] := tab[i]+tab[j]*tab[i-j];
9    od;
10 od;
11 return tab[n];
12 end;
```

Question 10 : On applique directement la définition de l'énoncé.

```
1  decaler := proc(t::array, v::integer)
2  local i, n, u;
3  n := taille(t);
4  u[1] := v;
5  for i from 2 to n+1 do
6      if t[i-1] < v
7          then u[i] := t[i-1]
8          else u[i] := 1+t[i-1]
9      fi;
10 od;
11 return(u);
12 end;
```

Question 11 : Lignes 1 et 2, on crée le tableau `res` qui contiendra le résultat et un tableau `tmp` de stockage temporaire de même longueur. On initialise le tableau `res` (lignes 6 à 9). La boucle des lignes 11 à 27 permet de calculer les échangeurs de $\{1,\dots,k\}$ à partir des échangeurs de $\{1,\dots,k-1\}$. La double boucle des lignes 13 à 22 permet de créer par décalage des échangeurs potentiels de $\{1,\dots,k\}$. Dans le bloc `if...fi` des lignes 16 à 20, on vérifie que l'on a effectivement construit un échangeur de $\{1,..,n\}$, si c'est le cas, on le stocke dans le tableau temporaire `tmp`. Dans la boucle des lignes 24 à 26, on transvase les échangeurs trouvés dans le tableau des résultats `res`[1].

```
1  enumerer_echangeurs := proc (n::posint)
2  local res, tmp, i, k, v, m, t, compteur, tab_tmp, nb_echangeurs;
3  nb_echangeurs := nombre_echangeurs(n);
4  res := allouer(nb_echangeurs);
5  tmp := allouer(nb_echangeurs);
6  for i from 2 to nb_echangeurs do
7        t := allouer(1); t[1] := 0; res[i] := t
8  od;
9  t := allouer(1); t[1] := 0; res[1] := t
10 m := 1;
11 for k from 1 to n-1 do
12     compteur := 0;
13     for v from 1 to k+1 do
14         for i from 1 to m do
15             tab_tmp := decaler(res[i], v);
16             if est_echangeur(tab_tmp)
17                 then
18                     compteur := compteur+1;
19                     tmp[compteur] := tab_tmp
20             fi;
21         od;
22     od;
23     m := nombre_echangeurs(k+1);
24     for i from 1 to m do
25        res[i] := tmp[i]
26     od;
27 od;
28 return(res);
29 end;
```

Voici le résultat de `enumerer_echangeurs(4)` :

$[[1,2,3,4],[1,2,4,3],[1,3,2,4],[1,3,4,2],[1,4,2,3],[1,4,3,2],[2,1,3,4],$
$[2,1,4,3],[2,3,1,4],[2,3,4,1],[2,4,3,1],[3,1,2,4],[3,2,1,4],[3,2,4,1],$
$[3,4,1,2],[3,4,2,1],[4,1,2,3],[4,1,3,2],[4,2,1,3],[4,2,3,1],[4,3,1,2],$
$[4,3,2,1]]$

1. ligne 25, il faut en réalité écrire `res[i] := eval(tmp[i])` pour que ça marche en Maple