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

Échangeurs de Polynômes

```
1  import numpy as np
2
3  def taille(a):
4      return(len(a)-1)
5
6  def allouer(n):
7      return(np.zeros(n+1))
```

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

```
1  def evaluation(P, v):
2      n = taille(P)
3      res = 0.
4      for i in range(n,0,-1):
5          res = (res + P[i])*v
6      return(res)
```

Question 2. On parcourt le tableau représentant le polynôme avec la boucle des lignes 2-4 et on retourne la première valeur pour laquelle $a_i \neq 0$. Si P est le polynôme nul, on retourne la valeur 0.

```
1  def valuation(P):
2      for i in range(taille(P)):
3          if P[i+1] <> 0:
4              return(i+1)
5      return(0)
```

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 (lignes 9-10). On duplique le polynôme P_1 dans Q (lignes 10-11) et enfin on lui soustrait P_2 (ligne 12-13).

```
1  def difference(P1, P2):
2      n1 = taille(P1); n2 = taille(P2)
3      if n2 < n1 :
4          n = n1
5      else :
6          n = n2
7      Q = allouer(n)
8      for i in range(n):
9          Q[i] = 0
10     for i in range(n1) :
11         Q[i+1] = P1[i+1]
12     for i in range(n2):
13         Q[i+1] = Q[i+1]-P2[i+1]
14     return(Q)
```

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 si $P_1 - P_2$ 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  def compare_neg(P1, P2):
2      Q = difference(P1, P2); v = valuation(Q)
3      if v == 0 :
4          return(0)
5      if v % 2 == 0:
6          if 0 < Q[v]:
7              return(1)
8          else :
9              return(-1)
10     else:
11         if 0 < Q[v]:
12             return(-1)
13         else :
14             return(1)
```

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

```
1  def echange(t, i, j):
2      tmp = t[i]
3      t[i] = t[j]
4      t[j] = tmp
5
6  def tri(t):
7      n = taille(t)
8      for i in range(n-1):
9          for j in range(1,n-i) :
10             if compare_neg(t[j], t[j+1]) < 0:
11                 echange(t, j, j+1)
```

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 $P_1(x) - P_2(x) \sim (a_v - b_v).x^v$. Donc à droite de zéro, $P_1 - P_2$ est localement du signe de $(a_v - b_v)$.

```
1  def compare_pos(P1, P2):
2      Q = difference(P1, P2)
3      v = valuation(Q)
4      if v == 0 :
5          return(0)
6      if 0 < Q[v] :
7          return(1)
8      else :
9          return(-1)
10
11 def verifier_permute(pi, t) :
12     n = taille(pi)
13     for i in range(1,n) :
14         if not(compare_pos(t[pi[i]], t[pi[i+1]])):
15             return(False)
16     return(True)
```

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 $1 \leqslant d < c < b < a \leqslant n$. On quitte le programme en retournant la valeur `faux` si l'un des triplet (a, b, c) ne convient pas (lignes 7 et 9). Si tous les triplets (a, b, c) conviennent, on retourne la valeur `vrai` (ligne 10).

```
1  def est_echangeur_aux(pi, d):
2      n = taille(pi)
3      for c in range(d+1,n-1):
4          for b in range(c+1,n) :
5              for a in range(b+1,n+1) :
6                  if pi[c] < pi[a] and pi[a] < pi[d] and pi[d] < pi[b] :
7                      return(False)
8                  if pi[b] < pi[d] and pi[d] < pi[a] and pi[a] < pi[c] :
9                      return(False)
10     return(True)
```

Question 8 : On utilise la fonction auxiliaire `est_echangeur_aux` pour savoir si la permutation π est un échangeur. Dans la boucle des lignes 3 à 5, on teste toutes les valeurs possibles de d. On quitte le programme si une valeur d ne convient pas (ligne 5). Si toutes les valeurs de d conviennent, on retourne la valeur `vrai` (ligne 6).

```
1  def est_echangeur(pi) :
2      n = taille(pi)
3      for d in range(1,n+1) :
4          if not(est_echangeur_aux(pi, d)) :
5              return(False)
6      return(True)
```

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 3). 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 5 et on ajoute $\sum_{j=1}^{i-1} a_j a_{i-j}$ grâce à la boucle des lignes 6 à 7.

```
1  def nombre_echangeurs(n) :
2      tab = allouer(n)
3      tab[1] = 1
4      for i in range(2,n+1) :
5          tab[i] = tab[i-1]
6          for j in range(1,i) :
7              tab[i] = tab[i]+tab[j]*tab[i-j]
8      return(int(tab[n]))
```

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

```
1  def decaler(t, v) :
2      n = taille(t)
3      u = allouer(n+1)
4      u[1] = v
5      for i in range(2,n+2) :
6          if t[i-1] < v :
7              u[i] = t[i-1]
8          else :
9              u[i] = 1+t[i-1]
10     return(u);
```

Question 11 : Lignes 3 et 4, 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 5 à 8). La boucle des lignes 10 à 21 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 18 permet de créer par décalage des échangeurs potentiels de $\{1,\dots,k\}$. Dans le bloc de l'instruction conditionnelle des lignes 16 à 18, 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 20 à 21, on transvase les échangeurs trouvés dans le tableau des résultats `res`.

```
1  def enumerer_echangeurs(n) :
2      nb_echangeurs = nombre_echangeurs(n)
3      res = [[] for i in range(nb_echangeurs+1)]
4      tmp = [[] for i in range(nb_echangeurs+1)]
5      for i in range(1,nb_echangeurs):
6          t = allouer(1)
7          t[1] = 1
8          res[i] = t
9      m = 1
10     for k in range(1,n) :
11         print(k)
12         compteur = 0
13         for v in range(1,k+2) :
14             for i in range(1,m+1) :
15                 tab_tmp = decaler(res[i], v)
16                 if est_echangeur(tab_tmp) :
17                     compteur = compteur+1
18                     tmp[compteur] = tab_tmp
19         m = nombre_echangeurs(k+1)
20         for i in range(1,m+1) :
21             res[i] = tmp[i]
22     return(res);
```

Voici le résultat de `enumerer_echangeurs(4)` :

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