CONCOURS ESIM Entrepreneur Industrie - Session 2001

Option MP

EPREUVE D'INFORMATIQUE

Durée : 4 heures

L'usage des calculatrices est autorisé. Les algorithmes demandés peuvent être codés en CAML ou en PASCAL.
Quand aucune directive n'est donnée le candidat peut établir une version récursive ou non. Quel que soit le choix du langage, les algorithmes doivent être écrits de la manière la plus courte possible, parfaitement lisible, avec une indentation convenable, sans aucune rature et en ***respectant scrupuleusement les notations introduites****. Ils doivent être documentés par des explications concises et précises sur les points qui le nécessitent.*
La lecture de l'annexe *correspondant au langage choisi est vivement conseillée avant d'aborder les questions de programmation.*
Les réponses qui ne respecteraient pas les consignes précédentes ne seront pas prises en considération.
Les questions sont assez souvent indépendantes les unes des autres, il est cependant conseillé de respecter la chronologie proposée.
Les parties I, II et III sont indépendantes.

I- Plus long plateau.

Soit $t_1, t_2, \cdots, t_n$ une suite de n de nombres réels, on appelle plateau une sous-suite de composantes consécutives égales entre elles. Plus précisément, si on note $t[k..l]$ un plateau ($k \leq l$), alors $\forall j, k \leq j \leq l \Rightarrow t_k = t_j$.

1) On suppose que les termes de la suite sont ordonnés selon l'ordre croissant, proposer un algorithme en *O(n)* qui permette de trouver l'indice *d* du début du plus long plateau ainsi que sa longueur *l*.

2) Les termes de la suite sont maintenant dans n'importe quel ordre, proposer un algorithme en *O(n)* qui permette de trouver l'indice *d* du début du plus long plateau ainsi que sa longueur *l*.

II- Calcul du déterminant d'une matrice à coefficients dans Z.

Soit une matrice A carrée à coefficients dans Z d'ordre n : $A = (a_{ij})_{\substack{1 \leq i \leq n \\ 1 \leq j \leq n}}$ dont on cherche à calculer le déterminant D.

- si n=1 alors $D=a_{11}$.
- si n>1 cherchons le premier indice h tel que $|a_{hn}| = \min_{1 \leq j \leq n} |a_{jn}|$ avec $a_{hn} \neq 0$. S'il n'existe pas un tel indice h, c'est que D=0, sinon pour chaque ligne $j \neq h$ on écrit la division euclidienne $a_{jn} = q_{jh} a_{hn} + r_{jh}$ avec $|r_{jh}| < |a_{hn}|$, et on soustrait à chaque ligne j, q_{jh} fois la ligne h.

1) Démontrer qu'en itérant ce processus on obtient nécessairement au bout d'un nombre fini d'itérations un seul élément non nul dans la dernière colonne.

2) Comment continuer l'algorithme?

3) Comment est modifié le déterminant de la matrice A lorsqu'on permute deux lignes entre elles?

4) Ecrire la fonction *determine_ligne(m,k);* qui étant donnée une matrice *m* à coefficients dans Z, dont la dernière colonne a pour indice *k*, calcule l'indice ligne *h* défini dans le préambule et affecte à la variable globale *nul* la valeur *true* si tous les éléments de la colonne *k* sont nuls, *false* sinon.

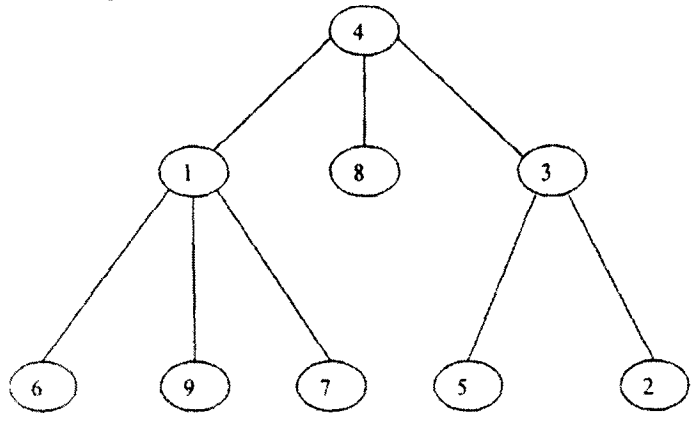
5) Ecrire la procédure *soustrait_ligne(m,k,h),* qui étant donnée une matrice *m* à coefficients dans Z, dont la dernière colonne a pour indice *k*, transforme la matrice *m* en enlevant à chaque ligne $j (j \neq h)$ q_{jh} fois la ligne h. De plus cette procédure affecte à la variable globale *fini* la valeur *true* si l'on a terminé l'étape *k*, c'est-à-dire si tous les éléments de la colonne *k* sont nuls sauf un, et la valeur *false* sinon.

6) Ecrire la fonction *determinant(m,n),* qui étant donnée une matrice *m*, d'ordre *n*, à coefficients dans Z, calcule son déterminant en programmant l'algorithme établi à la question **2)**.

III- Arbres binomiaux et tas binomiaux.

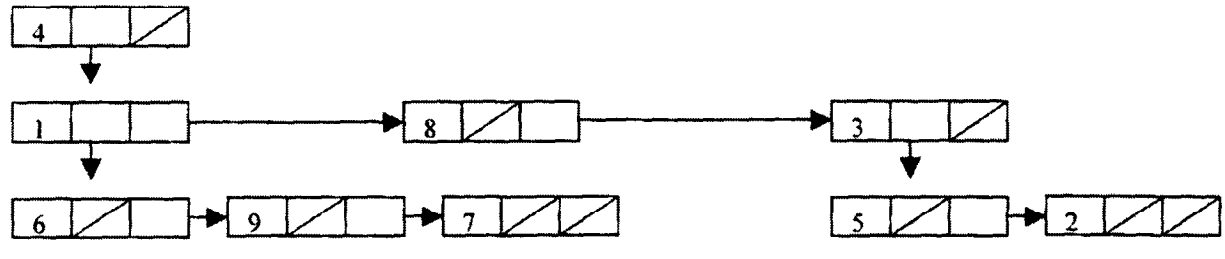
Rappels de définitions et notations :
On appelle le **degré** d'un nœud le nombre de ses fils.
Un nœud de degré 0 est dit **terminal** ou **feuille**.
On dit qu'un nœud *x* est un **ascendant** de *y* si et seulement si *x* est le père de *y* ou *x* est un ascendant du père de y.
Le **niveau** d'un nœud est le nombre de ses ascendants. La racine d'un arbre est donc de niveau 0.
On appelle **branche** une suite de nœuds $x_1, x_2, \cdots, x_k$ telle que x_{i+1} soit fils de x_i pour $1 \leq i < k$, x_1 est la racine de l'arbre et x_k est une feuille. La **longueur** d'une branche possédant *k* nœuds est *k*-1.
La **hauteur** d'un arbre est la longueur de sa plus longue branche.
Un **arbre ordonné** est un arbre dans lequel les fils de chaque nœud sont ordonnés. Il existe un premier fils (**fils aîné**), un deuxième fils, ..., et un dernier fils.

Dans la suite de ce sujet un arbre ordonné est représenté à l'aide de deux liens sur chaque nœud, ***fils*** et ***frere***. Le lien ***fils*** permet d'accéder au fils aîné du nœud, le lien ***frere*** permet d'accéder au premier frère (frère cadet) du nœud.
Par exemple l'arbre :

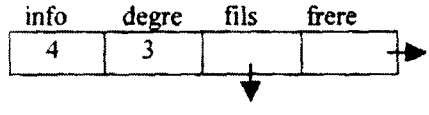


(figure 1)

est représenté avec la structure ***fils-frere*** par :



Dans la suite de cette partie les nœuds comporteront quatre champs :*info, degre, fils* et *frere*. Les champs *info* et *degre* sont de type entier et indique respectivement l'information contenue dans le nœud et son degré.



Un **arbre binomial** est défini par récurrence : l'arbre binomial B_0 d'ordre 0 ne contient qu'un seul nœud, l'arbre binomial B_k d'ordre *k* est construit à partir de deux arbres binomiaux d'ordre *k*-1 en plaçant la racine de l'un comme fils aîné de la racine de l'autre, l'ancien fils aîné devenant frère cadet.

1) Représenter schématiquement les arbres binomiaux B_0, B_1, B_2, B_3.
2) Montrer que l'arbre binomial B_k d'ordre *k* possède 2^k nœuds et que sa hauteur est *k*.
3) Montrer que la racine de l'arbre binomial B_k d'ordre *k* est de degré *k*.
4) Montrer que dans l'arbre binomial B_k il y a exactement C_k^i nœuds au niveau *i* pour *i*=0,1,..., *k* (c'est de cette propriété que vient le nom arbre binomial).
5) Montrer qui si les fils de la racine d'un arbre binomial d'ordre *k* sont numérotés *k*-1, *k*-2, ..., 1, 0 en partant du fils aîné, alors le fils n° *i* est la racine d'un sous-arbre binomial d'ordre *i*.
6) Ecrire la fonction *fusionne_arbre* qui a pour arguments deux arbres binomiaux *a* et *b* (supposés d'ordre *k*), cette fonction retourne pour résultat l'arbre binomial d'ordre *k*+1 fusion des arbres *a* et *b* par le procédé indiqué dans la définition.
7) Ecrire la procédure récursive *affiche_arbre* qui a pour argument un arbre *a* et un entier *niv* (le paramètre *niv* indique le niveau du sous-arbre à afficher). Cette procédure doit afficher, en pré ordre les champs *info* de tous les nœuds de l'arbre *a* avec une même indentation pour les nœuds d'un même niveau.
   En supposant que l'on décale de cinq espaces vers la droite chaque fois que le niveau augmente de un, voici le résultat produit par l'affichage de l'arbre de la **figure 1 :**

```
4
---- 1
--------- 6
--------- 9
--------- 7
---- 8
---- 3
--------- 5
--------- 2
```

Un **tas binomial** est une liste d'arbres binomiaux telle qu'il y a au plus un arbre binomial dont la racine soit d'un degré donné et dans chaque arbre l'information d'un nœud est supérieure ou égale aux informations des nœud ancêtres. Ainsi la racine contient toujours la plus petite information.
Pour représenter un tas binomial on utilise le lien *frere* des nœuds racine pour chaîner entre eux les arbres binomiaux qui composent la liste.

8) Combien d'arbres binomiaux y-a-t-il dans un tas binomial totalisant *n* nœuds ?

9) Ecrire la fonction *fusionne* qui a pour argument deux tas binomiaux *tb1* et *tb2* ordonnés suivant les degrés croissants des racines et donne pour résultat la liste ordonnée par degré croissant fusion des tas *tb1* et *tb2*. Ici la liste résultat ne représente probablement pas un tas binomial car elle peut posséder des arbres dont les racines sont de même degré. On remarquera qu'il peut y avoir au plus deux racines de même degré.

10) Donner, en français, le plus clairement possible, un algorithme qui permette de fusionner deux tas binomiaux. Cet algorithme reçoit en entrée deux tas binomiaux *tb1* et *tb2* et donne à la sortie le tas binomial *tb* fusion des tas *tb1* et *tb2*. A partir de la liste d'arbres obtenue par fusion (question 9) l'algorithme consiste à fusionner (*fusionne_arbre*, question 6) entre eux les arbres de la liste dont les racines ont le même degré jusqu'à ce qu'il n'y ait qu'une seule racine au plus par degré. On s'efforcera de n'effectuer qu'un seul parcours de liste.

ANNEXE PASCAL

Rappels :

- La fonction abs a pour résultat la valeur absolue de l'expression passée en paramètre.
- L'opérateur infixé de division entière est noté div.

II) Calcul de déterminant.

Déclarations.

```
const
           nmax=10 ;
type
         matrice_entier :array[1..nmax,1..nmax]of integer ;
var
           a :matrice_entier ;
           n :integer ;
    nul,fini :boolean ;
```

En-tête des fonctions et des procédures demandées :

```
function determine_ligne(var m :matrice_entier ;k :integer) :integer ;

procedure soustrait_ligne(var m :matrice_entier ;k,l :integer) ;

function determinant (var m :matrice_entier ;n :integer) :integer ;
```

III) Arbres binomiaux.

Déclarations .

```
type
     pnoeud = ^nœud ;

       nœud = record
                degre,info : integer ;
                fils,frere : pnoeud ;
              end ;
```

En-tête des fonctions et des procédures demandées :

```
function fusionne_arbre(a,b :pnoeud) :pnoeud ;

procedure affiche_arbre(tb :pnoeud) ;

function fusionne(tb1,tb2 :pnoeud) :pnoeud ;
```

ANNEXE CAML

Rappels de quelques fonctions de la bibliothèque CAML :

Si m est une matrice, `m.(i).(j)` désigne la composante qui se trouve sur la ligne `i` et la colonne `j`.

`print_int : int -> unit` imprime un entier, en décimal.

`print_string : string -> unit` imprime une chaîne.

`Print_newline :unit ->unit` imprime un saut de ligne.

`abs : int -> int` renvoie la valeur absolue de son argument

II) Calcul de déterminant.

Valeurs globales :

```
let n =5 ;;
let m = make_matrix n n 0 ;;

let fini = ref (false) ;;

let nul = ref (false) ;;
```

Liste des fonctions et procédures demandées

```
determine_ligne : int vect vect -> int -> int = <fun>

soustrait_ligne : int vect vect -> int ->int -> unit = <fun>

determinant : int vect vect -> int -> int = <fun>
```

III) Arbres binomiaux.

```
type arbre =
    nil
  | noeud of int*int*arbre*arbre;;
```

Liste des fonctions et procédures demandées :

```
fusionne_arbre : arbre -> arbre -> arbre = <fun>

affiche_arbre : arbre -> int -> unit = <fun>

fusionne : arbre -> arbre -> arbre = <fun>
```