CONCOURS ESIM Entrepreneur Industrie - Session 2002

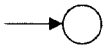
# Filière MP

# 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 parties I, II et III sont indépendantes.*

## I- Automates.

Notations :
On représentera un état initial par le schéma : 

et un état final par : 

**1)** On considère le langage $L_1$ des mots *m* construits sur l'alphabet {0,1} ayant un nombre pair de 1 et se terminant par 0, construire l'automate $A_1$ qui reconnaît les mots de $L_1$.

**2)** On considère le langage $L_2$ des mots *m* construits sur l'alphabet {0,1} ayant un nombre impair de 1 et se terminant par 1 (le dernier 1 n'intervenant pas pour le calcul de la parité), construire l'automate $A_2$ qui reconnaît les mots de $L_2$.

**3)** Déduire des questions précédentes l'écriture de l'automate $A_3$ non déterministe sans *ε-transition* (transition étiquetée par le vide) qui reconnaît les mots de $L_3 = L_1 \cup L_2$. Le dernier symbole d'un mot *m* appartenant à $L_3$ est appelé bit de parité. Ce bit est donc égal à 0 si le nombre de 1 contenus dans le mot *m* (hors bit de parité) est pair, il est égal à 1 sinon.

**4)** Construire l'automate déterministe $A_4$ à partir de $A_3$.

## II- Permutations et arbres binaires croissants.

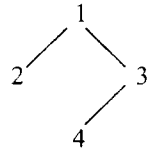
On appelle arbre **binaire croissant** un arbre binaire dont les nœuds sont étiquetés par des entiers de façon à ce que l'étiquette d'un nœud quelconque soit toujours strictement plus petite que celles qui étiquettent tous les descendants de ce nœud. Il existe un bijection entre les arbres binaires croissants étiquetés par les entiers 1, 2 ,... , *n* et les permutations de l'ensemble {1,2,...,*n*}.



**Tournez la page S.V.P.**

Algorithme de construction de l'arbre associé à une permutation:

La racine de l'arbre est étiquetée par 1 (le minimum). Tous les nombres à gauche (resp. à droite) de 1 dans la permutation sont placés dans le sous-arbre gauche (resp droit) de l'arbre. Et ainsi de suite récursivement.
Par exemple, à la permutation (2, 1, 4, 3) correspond l'arbre binaire croissant :




**1)** Donner un algorithme qui permet d'afficher la permutation associée à un arbre binaire croissant ayant n noeuds étiquetés par les entiers 1,2,...,*n*.

**2)** Ecrire la fonction *construire_arbre* qui renvoie pour résultat l'arbre binaire croissant associé à une permutation *p*, les arguments de cette fonction sont à préciser.

## III- Permutations et tableaux de Young.

Le but de cette partie est d'étudier quelques propriétés remarquables qui mettent en relation les permutations et certains tableaux de nombres entiers.
Un tableau de Young de forme $\lambda = (\lambda_1, \lambda_2, \cdots, \lambda_p)$, où $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_p > 0$, est un arrangement de $n = \lambda_1 + \lambda_2 + \cdots + \lambda_p$ entiers rangés dans un tableau dont les lignes sont justifiées à gauche, avec $\lambda_i$ éléments dans la ligne *i*, tel que les nombres de chaque ligne soient dans un ordre croissant de la gauche vers la droite, et les nombres de chaque colonne soient dans un ordre croissant du haut vers le bas. Par exemple,

| | | | |
|---|---|---|---|
| 1 | 3 | 9 | 16 |
| 2 | 6 | 10 | |
| 4 | 11 | 12 | |
| 13 | 14 | | |
| 19 | | | |

est un tableau de Young de forme (4,3,3,2,1).

Représentation et conventions:

On représentera un tableau de Young par une matrice carrée de dimension *nmax*; pour simplifier l'écriture des algorithmes, on supposera que *nmax* est suffisamment grand pour ne pas avoir de tests de débordement à effectuer sur cette valeur et que le tableau est bordé par des 0 au dessus et à gauche (i.e. la ligne 0 et la colonne 0 sont remplies de 0), Les composantes non remplies (celles qui correspondent aux cases de dessous ou de droite dans le tableau) sont initialisées avec $\infty$ (dans la pratique avec *maxint* un nombre entier le plus grand possible).
On dit qu'une case $t_{i,j}$ d'un tableau de Young est une **extrémité** si et seulement si $t_{i,j+1} = \infty$ et $t_{i+1,j} = \infty$.

1) Ecrire la fonction booléenne *est_de_young* qui prend pour argument un tableau *T* et qui renvoie *vrai* si *T* représente un tableau de Young ,*faux* sinon. On s'attachera à effectuer le moins de comparaisons possibles.

**Dans tout ce qui suit, on suppose que les éléments d'un tableau de Young sont distincts.**

On s'intéresse à un algorithme de construction d'un tel tableau qui est basé sur une méthode d'insertion illustrée par l'exemple qui suit. Imaginons que l'on veuille insérer 8 dans le tableau précédent.

Algorithme d'insertion :
La méthode consiste toujours à placer le nombre à insérer dans la première ligne, ainsi 8 prend la place de 9, car 9 est le plus petit élément plus grand que 8 dans cette ligne. L'élément 9 est « chassé » dans la seconde ligne et prend la place de 10, puis 10 est placé dans la troisième ligne à la place de 11, qui prend la place 13, qui lui se propage sur la cinquième ligne à la place de 19. Finalement le procédé s'arrête en plaçant 19 sur la ligne 6 juste en dessous de 13.

Le tableau a été transformé en:

| | | | |
|---|---|---|---|
| 1 | 3 | 8 | 16 |
| 2 | 6 | 9 | |
| 4 | 10 | 12 | |
| 11 | 14 | | |
| 13 | | | |
| 19 | | | |

La case de coordonnées (6,1) a été rajouté au tableau.

**2)** Montrer que l'algorithme d'insertion décrit ci-dessus permet d'obtenir encore un tableau de Young.

**3)** Ecrire la fonction *insere* (procédure en PASCAL) qui prend pour argument un nombre entier *elmt* et qui l'insère dans un tableau de Young *T*. Cette procédure retournera pour résultat la paire $(x,y)$, coordonnées de la case qui a été rajoutée au tableau par ce procédé d'insertion.

## Algorithme de suppression :

L'algorithme reçoit en entrée un tableau *T* de Young ainsi que les coordonnées $x,y$ d'une case *extrémité*. Il transforme ce tableau en un tableau de même forme excepté pour la composante de coordonnées $x,y$ qui vaut $\infty$. Un élément *e* déterminé par l'algorithme est supprimé du tableau *T*.

Initialisations :

$e \leftarrow t_{x,y}$

$t_{x,y} \leftarrow \infty$

$j \leftarrow y$

$i \leftarrow x\text{-}1$

Itération :

On recherche dans la ligne $i$ à partir de la colonne $j$, en incrémentant $j$, le plus grand élément inférieur à *e*.

On échange alors *e* avec cet élément $t_{i,j}$, puis on décrémente $i$ et on cherche de la même manière à placer *e* sur cette nouvelle ligne et ainsi de suite.

Arrêt :

L'algorithme s'arrête lorsque l'on a atteint la ligne 0; *e* contient alors l'élément de la ligne 1 qui est « chassé » du tableau.

On peut démontrer assez facilement que ces *algorithmes d'insertion et de suppression sont rigoureusement les parfaits inverses l'un de l'autre*. Cette propriété nous servira par la suite pour établir que la correspondance Robinson-Schensted est bijective.

**4)** Ecrire la fonction *supprime* qui prend pour arguments un tableau de Young *T*, les entiers $x$ et $y$ coordonnées d'une composante qui est une *extrémité* de *T* et qui retourne pour résultat l'élément *e* qui a été « chassé » du tableau *T*. Le tableau *T* est bien sûr modifié par l'algorithme de suppression.

**5)** Décrire un algorithme qui prend en entrée un tableau de Young *T* et qui retourne le tableau de Young *T* où l'élément du coin en haut à gauche a été supprimé du tableau *T*. Cet algorithme commencera par « vider » la composante du coin, puis propagera la case vide dans le tableau d'une manière que l'on précisera.

**6)** Ecrire la fonction *supprime_coin* qui retourne pour résultat la composante du coin $t_{1,1}$ en la supprimant du tableau de Young *T* et modifie le tableau T pour que celui-ci reste un tableau de Young.

**7)** Déduire et décrire de ce qui précède un algorithme de tri permettant de trier une suite de nombres entiers.



**Tournez la page S.V.P.**

Définition : Un tableau de Young ayant $n$ cases est dit **standard** s'il contient tous les nombres 1, 2, ..., n.

## Correspondance de Robinson-Schensted.

Cette correspondance établit une bijection entre l'ensemble des permutations de $\{1,2, ...,n\}$ et des couples de tableaux de Young standards de $n$ cases.
Soit la permutation

$$\pi = \begin{pmatrix} 1 & 2 & \cdots & n \\ p_1 & p_2 & \cdots & p_n \end{pmatrix}$$ où $p_1, p_2, \cdots, p_n$ sont les images de 1,2, ..., $n$ par $\pi$

on va lui associer un couple de tableaux (P,Q) où les tableaux P et Q sont de même forme.

### Algorithme de construction du couple (P,Q) :

P et Q sont initialement vides. Pour $i$ variant de 1 à $n$ on insère $p_i$ dans le tableau P en utilisant l'algorithme d'insertion présenté ci-dessus ; puis on place $i$ dans la case repérée par $(x,y)$, où $x$ et $y$ sont les coordonnées de la nouvelle case de P qui a été remplie (x et y sont calculés par la procédure *insere*).

Par exemple si la permutation donnée est $\begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 4 & 3 & 1 & 5 & 2 \end{pmatrix}$

| | P | Q |
|---|---|---|
| Insertion de 4 : | 4 | 1 |
| Insertion de 3 : | 3<br>4 | 1<br>2 |
| Insertion de 1 : | 1<br>3<br>4 | 1<br>2<br>3 |
| Insertion de 5 : | 1 5<br>3<br>4 | 1 4<br>2<br>3 |
| Insertion de 2 : | 1 2<br>3 5<br>4 | 1 4<br>2 5<br>3 |

Le tableau Q sert en fait à garder la trace de l'ordre dans lequel a été créé la nouvelle case dans P à chaque étape.

**8)** Montrer que le tableau Q est de même forme que le tableau P et que c'est aussi un tableau de Young.

Réciproquement, si P et Q sont deux tableaux de Young standards de $n$ cases de même forme, il est facile de leur faire correspondre une unique permutation. Il suffit pour $i = n, ..., 2, 1$ de prendre pour $p_i$ l'élément $e$ qui est chassé du tableau lorsqu'on applique l'algorithme de suppression sur le tableau P en supprimant la case de coordonnée (x,y), les valeurs $x,y$ sont telles que $Q_{x,y}=i$. Puisque les algorithmes d'insertion et de suppression sont les inverses l'un de l'autre la correspondance est bien bi-univoque.

**9)** Quelle est la permutation qui correspond aux tableaux ?

P=

| | |
|---|---|
| 1 | 3 |
| 2 | 4 |
| 5 | |

Q=

| | |
|---|---|
| 1 | 4 |
| 2 | 5 |
| 3 | |

Notations et définitions :

- Une *involution* est une permutation qui est égale à son inverse.

- On appelle *équerre* de coin (i,j) d'une case d'un tableau de Young , l'ensemble des cases qui se trouvent sur la même ligne à droite de $(i,j)$ et des cases qui se trouvent sur la même colonne en dessous de $(i,j)$. Une équerre peut éventuellement se réduire à une seule case (c'est le cas pour les cases *extrémités*). La *longueur d'une équerre* est égale aux nombres de cases qui la composent.
Voici un exemple d'équerre de coin (2,2) matérialisée par les *, sa longueur est de 3.

| | | |
|---|---|---|
| | | |
| | * | * |
| | * | |
| | | |

- Si P est un tableau de Young, on notera $P^T$ le tableau de Young transposé.
- On note $n_\lambda$ le nombre de tableaux de Young standard de forme $\lambda = (\lambda_1, \lambda_2, \cdots, \lambda_p)$, où $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_p > 0$, et $n = \lambda_1 + \lambda_2 + \cdots + \lambda_p$.

**10)** Ecrire la fonction *robinson* (procédure en PASCAL) qui prend pour argument une permutation *pi* de $n$ éléments et renvoie pour résultat l'unique paire $(P,Q)$ de tableaux de Young standards qui lui est associée par la correspondance de Robinson-Schensted.

**11)** Ecrire la fonction *produit_equerres* qui a pour arguments un tableau de Young $T$ et renvoie pour résultat le produit des longueurs de toutes les équerres figurant dans le tableau $T$.

Quelques résultats sur les tableaux de Young standards (utiles pour la suite) :

Propriété 1 : Le nombre $n_\lambda$ de tableaux de Young standards de $n$ cases ayant une forme $\lambda$ donnée est égal à $n$ ! divisé par le produit des longueurs de toutes ses équerres.

Propriété 2 : Si P est un tableau de Young standard correspondant à la permutation $\pi \begin{pmatrix} 1 & 2 & \cdots & n \\ p_1 & p_2 & \cdots & p_n \end{pmatrix}$, alors $P^T$ est le tableau de Young standard correspondant à la permutation miroir $\pi^* = \begin{pmatrix} 1 & 2 & \cdots & n \\ p_n & p_{n-1} & \cdots & p_1 \end{pmatrix}$.



**Tournez la page S.V.P.**

Propriété 3 : Si à la permutation $\pi = \begin{pmatrix} 1 & 2 & \cdots & n \\ p_1 & p_2 & \cdots & p_n \end{pmatrix}$ la correspondance de Robinson-Schensted associe le couple de tableaux de Young standards (P,Q), alors à la permutation inverse $\pi^{-1}$ correspond le couple (Q,P).

**12)** Montrer que le nombre de tableaux de Young standards que l'on peut former à partir de $\{1,2,...n\}$ est égal au nombre d'involutions de $\{1,2,...n\}$.

**13)** Une forme $\lambda$ d'un tableau de Young standard de $n$ éléments étant fixée, exprimer en fonction $n_\lambda$ le nombre de couple de tableaux de Young standards (P,Q) de forme $\lambda$. En déduire une relation entre $n!$ et les valeurs $n_\lambda$ associées aux formes $\lambda$ de tableaux de Young ayant $n$ cases.

Définition : Soit une permutation

$$\pi = \begin{pmatrix} 1 & 2 & \cdots & n \\ p_1 & p_2 & \cdots & p_n \end{pmatrix}$$

En remarquant que l'algorithme d'insertion insère toujours les éléments $p_i$ dans la première ligne du tableau P, on dira que $p_i$ est de *classe k* si $p_i$ est placé en ligne 1 colonne $k$ lors de son insertion dans le tableau P.
On peut remarquer qu'un élément $p_i$ est de classe 1 si et seulement si $p_i = \min\{p_1, p_2, \cdots, p_i\}$.

**14)** Montrer qu'un élément $p$ est de classe $k$ si et seulement si $k$ est le plus grand nombre d'indices $i_1 < i_2 < \cdots < i_k$ tels que $p_{i_1} < p_{i_2} < \cdots < p_{i_k} = p$.

**15)** Si P est le tableau correspondant à la permutation $\pi = \begin{pmatrix} 1 & 2 & \cdots & n \\ p_1 & p_2 & \cdots & p_n \end{pmatrix}$, montrer que le nombre de colonnes de P est la plus grande longueur $c$ d'une sous-suite croissante d'éléments $p_{j_1} < p_{j_2} < \cdots < p_{j_c}$, où $j_1 < j_2 < \cdots < j_c$; et que le nombre de lignes de P est égal à la plus grande longueur $l$ d'une sous-suite décroissante d'éléments $p_{i_1} > p_{i_2} > \cdots > p_{i_l}$, où $i_1 < i_2 < \cdots < i_l$.

**16)** Montrer que n'importe quelle permutation ayant plus de $n^2$ éléments possède une sous-suite monotone d'éléments de longueur supérieure à $n$.

**17)** Montrer qu'il existe des permutations de $n^2$ éléments qui ne possèdent aucune sous-suite monotone d'éléments de longueur supérieure à $n$.

**18)** Trouver une formule donnant le nombre exact de permutations de $\{1,2,..., n^2\}$ ne possédant pas de sous-suite monotone de longueur supérieure à $n$ (*conseil* : utiliser la propriété 1).

# ANNEXE PASCAL

## II) Permutation et arbre binaire croissant.

Déclarations

```
Const nmax = 20 ;

type
     permutation = array [1..nmax] of integer ;

          pnoeud = ^noeud ;

           noeud = record
                      info : integer ;
                     fg,fd : pnoeud
                   end ;
```

En-tête de la procédure demandée :

```
procedure construire_arbre(…) ;
```

## III) Tableaux de young.

Déclarations

```
const
            nmax = 20;
          maxint = 32767;
type
           tableau = array[0..nmax,0..nmax]of integer;

      permutation = array[1..nmax] of integer;
```

On supposera connue la procédure `init(var T :tableau)` qui créé un tableau de young vide, i.e. toutes les composantes du tableau passé en paramètre sont initialisées avec `maxint` , sauf la première ligne et la première colonne qui sont remplies avec des zéros.

En-tête des fonctions et des procédures demandées :

```
function est_de_young(var T : tableau) : boolean;

procedure insere (elmt : integer; var T : tableau; var x,y : integer);

function supprime(x,y : integer ;var T : tableau) :integer;

function supprime_coin (var T : tableau) : integer;

procedure robinson (pi : permutation; n : integer; var P,Q : tableau);

function produit_equerres( T : tableau) : integer ;
```



**Tournez la page S.V.P.**

# 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`.
En CAML les matrices ainsi que les vecteurs sont indexés à partir de 0 ;

`vect_length: 'a vect -> int` renvoie le nombre d'éléments du tableau donné.
`fst: 'a * 'b -> 'a` renvoie la première composante d'une paire.
`snd: 'a * 'b -> 'a` renvoie la seconde composante d'une paire.

## II) Permutation et arbre binaire croissant.

Valeurs globales :

```
Let pi=[|7;2;4;6;3;5;1|];;
```

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

En-tête de la procédure demandée :

```
construire_arbre : . . . -> arbre = <fun>
```

## III) Tableaux de Young.

Valeurs globales :

```
let maxint =int_of_float (2.**30.-.1.);;
let nmax = 21 ;;
let m = make_matrix nmax nmax maxint ;;
```

On supposera connue la procédure `init : int vect vect -> unit = <fun>` qui créé un tableau de young « vide », i.e. toutes les composantes du tableau passé en paramètre sont initialisées avec `maxint`, sauf la première ligne et la première colonne qui sont remplies avec des zéros.

Liste des fonctions demandées

```
est_de_young : int vect vect -> bool = <fun>

insere : int -> int vect vect -> int * int = <fun>

supprime : int -> int ->int vect vect -> int = <fun>

supprime_coin : int vect vect ->int = <fun>

robinson : int vect -> int vect vect * int vect vect = <fun>

produit_equerres int vect vect -> int = <fun>
```