CONCOURS ESIM Entrepreneur Industrie - Session 2003

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 parties I et II sont indépendantes.

I - Tranche de somme minimale.

Soit $V=(v_0, v_1, \cdots, v_n)$ un tableau dont les éléments sont des entiers relatifs indexés de 0 à n ($v_i \in \mathbb{Z}, 0 \le i \le n$). Une tranche T de V est une suite non vide d'éléments consécutifs de V, i.e. $T = (v_j, v_{j+1}, \cdots, v_k)$, avec $0 \le j \le k \le n$. On note $T=V(j..k)$ et E l'ensemble des tranches de V.

A toute tranche $T=V(j..k)$ de V on associe la somme $S(T)$ des éléments qui la composent : $S(T) = \sum_{i=j}^{k} v_i$.

Une tranche T de V est dite minimale si la somme qui lui est associée est minimale. Il peut ne pas y avoir unicité.
Le but de cette partie est de comparer divers algorithmes qui permettent de trouver une tranche minimale $T=V(d..f)=min\ \{ S(t), t \in E \}$.

1) Ecrire la fonction *somme* qui prend pour arguments un tableau V et deux nombres entiers j et k ($j \le k$) et qui calcule la somme associée à la tranche $V(j..k)$.

2) Ecrire, en utilisant la fonction *somme*, la fonction *tranche_min1* qui prend pour arguments un tableau V et un entier n (n désigne ici le nombre de composante du tableau V) et qui renvoie pour résultat la somme $S(T)$ associée à une tranche minimale T.

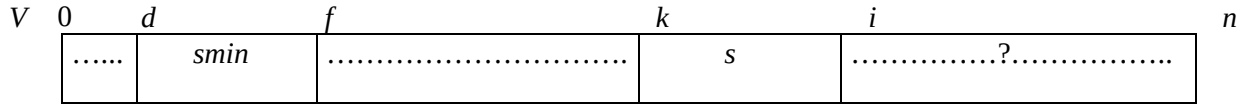
3) Déterminer en fonction de n, le nombre d'additions et de comparaisons effectuées suite à l'appel de la procédure *tranche_min1*.

4) Pour j fixé, $0 \le j \le n$. Ecrire la fonction *somme_min* ,prenant pour arguments un tableau V et deux entiers j et n, qui sans utiliser la fonction *somme* écrite auparavant, calcule la somme d'une tranche minimale dont le premier élément est v_j, i.e. $\min_{j \le k \le n} S(V(j..k))$.

5) Ecrire, en utilisant la fonction *somme_min*, la fonction *tranche_min2* qui prend pour arguments un tableau *V* et un entier *n* (*n* désigne ici le nombre de composantes du tableau *V*) et qui renvoie pour résultat la somme *S(T)* associée à une tranche minimale *T*.

6) Déterminer en fonction de *n*, le nombre d'additions et de comparaisons effectuées suite à l'appel de la procédure *tranche_min2*.

7) Soit le schéma :



Précisons la situation décrite par ce schéma. En supposant que toutes les composantes entre 1 et *i* ont été examinées, la tranche de somme minimale (en ne considérant que les éléments d'indices compris entre 0 et *i*) commence en *d*, se termine en *f* et a pour somme *smin*. Par ailleurs, parmi toutes les tranches qui s'achèvent en *i*, il en existe (au moins) une de somme minimale, on suppose qu'elle commence en *k* et a pour somme *s*.

a) Après incrémentation de *i* (*i*←*i*+1), quelles sont les deux seules valeurs à envisager, pour déterminer parmi toutes les tranches qui s'achèvent en i, celle qui est de somme minimale.

b) En déduire un algorithme, en O(*n*), permettant de trouver une tranche de somme minimale *T* du tableau *V*. Cet algorithme doit déterminer les indices *d* et *f* qui délimite la tranche minimale *T*, ainsi que *smin=S(T)* la somme qui lui est associée.

c) Ecrire la fonction *tranche_min3* correspondant à cet algorithme.

II - Mots de Lyndon

Dans cet énoncé, un mot est une suite finie d' entiers $u = (u_1, u_2, \cdots, u_n)$ où $n \geq 1$ et $u_i \in \{0,1\}$ pour *i=1,..., n*.
L' entier *n* désigne la longueur du mot *u*.
Si $u = (u_1, \cdots, u_n)$ et $v = (v_1, \cdots, v_m)$ sont des mots, on note *uv* le concaténé des mots *u* et *v*
$uv = (u_1, \cdots, u_n, v_1, \cdots, v_m)$.
Soient $u = (u_1, \cdots, u_n)$ et $v = (v_1, \cdots, v_m)$ deux mots.
Le mot *u* est un "début" de *v* si *n*<*m* et $u_i = v_i$ pour *i=1,..., n*.
Le mot *u* est une "fin" de *v* si *n*<*m* et $u_i = v_{m-n+i}$ pour *i=1,..., n*.
On pose $u \prec v$ *si u* est un début de *v* ou s' il existe un entier *k* tel que $u_i = v_i$ pour *i=1, ..., k-1* et $u_k < v_k$.
La relation $\preceq$ définie par $u \preceq v$ si et seulement si *u=v* ou $u \prec v$ s' appelle *l'ordre lexicographique.*
On pose *u*<<*v* si et seulement si $u \prec v$, *u* n'étant pas un début de *v*. Ainsi, $u \prec v$ si *u* est un début de *v* ou si *u*<<*v*.
Un *conjugué* d' un mot $u = (u_1, \cdots, u_n)$ est un mot de la forme $v = (u_i, \cdots, u_n, u_1, \cdots, u_{i-1})$ avec $2 \leq i \leq n$.
Un mot est un mot de Lyndon si $u \prec v$ pour tout conjugué *v* de *u*.
Un mot de longueur 1 est évidement un mot de Lyndon.

Etude Mathématique :

1) Démontrer que pour deux mots $u \neq v$, on a soit $u \prec v$, soit $v \prec u$.

2) Soit *u* un mot. Démontrer que s' il existe un mot qui est à la fois un début et une fin de *u*, alors *u* n' est pas un mot de Lyndon.

3) Démontrer qu' un mot u est un mot de Lyndon si et seulement si pour toute fin h de u, on a $u \prec h$.

4) Démontrer que si un mot u de longueur >1 est un mot de Lyndon, il existe deux mots de Lyndon f et g tels que u=fg et $f \prec g$ et, réciproquement, si f et g sont des mots de Lyndon tels que $f \prec g$, alors fg est un mot de Lyndon.

Conseils :

- Pour ($\Rightarrow$), on considérera g la fin de u de longueur maximale qui soit un mot de Lyndon. Un tel mot existe car les mots de longueur 1 sont des mots de Lyndon.

- Pour ($\Leftarrow$), on démontrera que $u=fg$ est plus petit que toutes ses fins. Si l'on note h une fin de u, on distinguera les cas $h=g$ et $h \neq g$.

Les résultats établis dans la partie mathématique pourront être utilisés, même s'ils n'ont pas été démontrés.

Partie informatique :

Dans cette partie les mots seront représentés par des chaînes de caractères.

5) Ecrire une fonction booléenne *inferieur* qui prend pour deux mots u et v, et renvoie pour résultat le booléen *true* si $u \prec v$, *false* sinon.

6) Ecrire la fonction *conjugue* qui a pour arguments un mot u et un entier i. Cette fonction rend pour résultat le conjugué du mot u qui débute par le suffixe commençant au $i^{ième}$ caractère.

7) Ecrire une fonction booléenne *lyndon* qui teste si un mot u, passé en paramètre, est un mot de Lyndon.

8) Soit $u = (u_1, u_2, \cdots, u_n)$ un mot. Une factorisation de Lyndon du mot u est une suite $u^{(1)}, u^{(2)}, \cdots, u^{(p)}$ de mots de Lyndon telle que $u = u^{(1)}u^{(2)}\cdots u^{(p)}$ et $u^{(1)} \succeq u^{(2)} \succeq \cdots \succeq u^{(p)}$.

Par exemple, pour u=(1,0,1,0,0,1,0,1,1,0,0,1,0,0), une factorisation de Lyndon est :

(1), (0,1), (0,0,1,0,1,1), (0,0,1), (0), (0)

On construit une factorisation de Lyndon d' un mot $u = (u_1, \cdots, u_n)$ comme suit:
On part de la suite $u^{(1)}, u^{(2)}, \cdots, u^{(n)}$ où $u^{(i)} = (u_i)$ pour $i=1, 2, ..., n$.
Si $u^{(1)}u^{(2)}\cdots u^{(r)}$ est une suite de mots de Lyndon et s' il existe un indice k $(1 \leq k \leq r)$ tel que $u^{(k)} \prec u^{(k+1)}$, on construit une nouvelle suite de mots de Lyndon $v^{(1)}, v^{(2)}, \cdots, v^{(r-1)}$ en posant

$v^{(j)} = u^{(j)}$ pour $j=1, ..., k-1$.
$v^{(k)} = u^{(k)}u^{(k+1)}$.
$v^{(j)} = u^{(j+1)}$ pour $j=k+1, ..., r-1$.

On itère ce processus tant qu' il est possible de raccourcir la séquence. La séquence finale est une factorisation de Lyndon.

Ecrire la fonction *factorisation*, qui calcule une factorisation de Lyndon du mot u, passé en paramètre, par le procédé énoncé ci-dessus. Cette fonction renvoie comme résultat une liste de mots : la liste des facteurs.

Le but des questions qui suivent est d'obtenir tous les mots de Lyndon de longueur inférieure ou égale à un entier *n* donné. Pour cela on utilise la propriété établie à la question **4** de la partie mathématique et le fait que (0) et (1) sont des mots de Lyndon. A partir des mots de longueur 1, on obtient le seul mot de Lyndon de longueur 2 à savoir (0,1). Les mots de Lyndon de longueur 3 sont obtenus à partir de la concaténation de deux mots de longueur plus petite et dont la somme vaut 3, et ainsi de suite :

Longueur 1 : (0), (1)
Longueur 2 : (0,1)
Longueur 3 : (0,0,1), (0,1,1)
Longueur 4 : (0,0,0,1), (0,0,1,1), (0,1,1,1)
……

9) Donner la liste de tous les mots de Lyndon de longueur 5.

10) Ecrire la fonction *insere_mot_lst* qui prend pour arguments un mot *u* et une liste de mots *lst* classée suivant l'ordre lexicographique. Cette fonction renvoie pour résultat une liste classée de mots dans laquelle le mot *u* a été inséré. L'insertion doit avoir lieu seulement si le mot *u* n'appartient pas déjà à la liste *lst*, dans le cas contraire la liste reste inchangée.

11) Ecrire la fonction *insere_lst_lst* qui prend pour arguments deux listes de mots, *lst1* et *lst2*, la liste *lst2* étant classée suivant l'ordre lexicographique. Cette fonction renvoie pour résultat, une liste classée de mots obtenue après avoir inséré tous les mots de la liste *lst1* dans la liste *lst2*. L'insertion n'ayant lieu que si le mot à insérer n'est pas déjà présent.

12) Ecrire la fonction *fusionne_listes* qui a pour arguments deux listes *lst1* et *lst2* de mots de Lyndon. Cette fonction renvoie pour résultat la liste classée de tous les mots de Lyndon obtenus par concaténation d' un mot de la liste *lst1* avec un mot de la liste *lst2*. La concaténation de deux mots de Lyndon donne un mot de Lyndon seulement si la propriété établie à la question **4** est satisfaite. Par ailleurs, un même mot ne doit pas figurer plusieurs fois dans cette liste résultat.

13) On considère une variable globale *lynd*, qui désigne un tableau de *nmax* de listes de mots. Ecrire la procédure *remplir_lynd* qui prend pour argument un entier *n* et qui remplit les composantes de 0 à *n* du tableau *l*ynd de telle manière que la *i*-ème composante de ce tableau contienne la liste ordonnée de tous les mots de Lyndon de longueur *i*.
Cette procédure initialisera la première composante avec la liste ordonnée des mots de Lyndon de longueur 1 (la composante 0 étant initialisée avec la liste vide), et utilisera les fonctions écrites auparavant pour remplir les autres composantes du tableau *lynd*.

ANNEXE PASCAL

I) Tranche de somme minimale.

Déclarations :

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

En-tête des fonctions et des procédures demandées :

```
function somme(var V :tableau ;n :integer) :integer;
function tranche_min1(var V :tableau ;n :integer) :integer ;
function somme_min(var V :tableau ;j,n :integer) :integer ;
function tranche_min2(var V :tableau ;n :integer) :integer ;
function tranche_min3 (. . .) : integer ;
```

II) Mots de Lyndon.

On rappelle qu'en Pascal l'opérateur de concaténation de chaînes se note `+` et que la fonction `copy(ch,d,lg)` permet d'extraire de la chaîne `ch` la sous-chaîne qui débute au rang `d` et qui possède `lg` caractères.

Déclarations :

```
const
           nmax = 20 ;
    longueurmax = 10 ;
type
                mot = string[longueurmax] ;
          list_mots = ^doublet ;
            doublet = record
                        info : mot ;
                        suiv : listmot ;
                      end ;
      tab_list_mots = array [0..nmax] of list_mots ;
```

En-tête des fonctions et des procédures demandées :

```
function inferieur(u,v :mot) :boolean ;
function conjugue(u :mot) :mot ;
function lyndon(u :mot) :boolean ;
function factorisation(u :mot) :list_mots ;
function insere_mot_lst(u :mot ;lst :list_mots) :list_mots ;
function insere_lst_lst(lst1,lst2 :list_mots) :list_mots ;
function fusionne_listes(lst1,lst2 :list_mots) :list_mots ;
procedure genere(n :integer) ;
```

ANNEXE CAML

Si V est un vecteur, `V.(i)` désigne la composante de rang `i`.
En CAML les vecteurs sont indexés à partir de 0.

I) Tranche de somme minimale.

Fonctions demandées :

```
somme : int vect -> int -> int -> int = <fun>
tranche_min1 : int vect -> int -> int = <fun>
somme_min : int vect -> int -> int -> int = <fun>
tranche_min2 : int vect -> int -> int = <fun>
tranche_min3 : int vect -> int -> int*int*int = <fun>
```

II) Mots de Lyndon.

Rappels de quelques fonctions de la bibliothèque CAML :

`string_length: 'a vect -> int` renvoie le nombre d'éléments d'une chaîne.
`list length: 'a list -> int` renvoie le nombre d'éléments d'une liste.
`char_for_read:` char -> `string` renvoie une chaîne représentant le caractère donné.
`sub_string: string -> int -> int -> string` permet d'extraire une sous-chaîne.
`substring ch d lg` renvoie une nouvelle chaîne extraite de la chaîne `ch`, la sous-chaîne qui débute au rang `d` et qui possède `lg` caractères.
^ désigne l'opérateur de concaténation de chaînes.
@ désigne l'opérateur de concaténation de listes.

Valeurs globales :

```
let nmax = 20;;
let lynd = make_vect (nmax) [];;
```

Procédures et fonctions demandées :

```
inferieur : string -> string -> bool = <fun>
conjugue : string -> int -> string = <fun>
lyndon : string -> bool = <fun>
factorisation : string -> string list = <fun>
insere_mot_lst : string -> string list-> string list = <fun>
insere_lst_lst : string list -> string list-> string list = <fun>
fusionne_listes : string list-> string list-> string list = <fun>
genere : int -> unit = <fun>
```