SESSION 2000

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- Parcours en escalier dans un tableau.

Soit M une matrice de $n+1$ lignes et de $n+1$ colonnes ($n>0$), telle que $m_{i,j} = 2^i 3^j$ pour $0 \leq i \leq n$ et $0 \leq j \leq n$.

1) Décrire un algorithme de remplissage du tableau à deux dimensions représentant la matrice M.

2) Quel est le nombre total de multiplications de votre algorithme ?

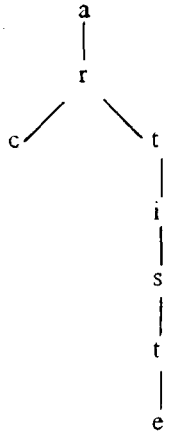
3) Ecrire la fonction *pgcd* qui a prend pour arguments une matrice M (telle que $m_{i,j} = 2^i 3^j$ pour $0 \leq i \leq n$ et $0 \leq j \leq n$) et quatre indices i, j, k, l dans $[0,n]$, et retourne pour résultat la valeur du plus grand commun diviseur de $m_{i,j}$ et de $m_{k,l}$.

4) On dispose d'une matrice M telle que $m_{i,j} = 2^i 3^j$ pour $0 \leq i \leq n$ et $0 \leq j \leq n$, décrire un algorithme qui étant donné un entier p, $1 \leq p \leq 2^n 3^n$, calcule les indices i, j du plus grand des minorants de p de la forme $2^i 3^j$.
On s'efforcera dans cette question de minimiser le nombre de comparaisons en exploitant la remarque qui suit.
Remarque : Si sur la ligne i_0 le plus grand des minorants de p se trouve en colonne j_0, alors sur la ligne i_0+1 le plus grand des minorants de p est :
- Soit l'élément qui se trouve sur en (i_0+1, j_0), si $m_{i_0+1, j_0} = 2m_{i_0, j_0}$ est inférieur ou égal à p.
- Soit l'élément qui se trouve en (i_0+1, j_0-1), si $m_{i_0+1, j_0} = 2m_{i_0, j_0}$ est strictement supérieur à p, car dans ce cas $m_{i_0+1, j_0-1} = \frac{2}{3} m_{i_0, j_0}$ est alors le plus grand des minorants de p sur la ligne i_0+1.

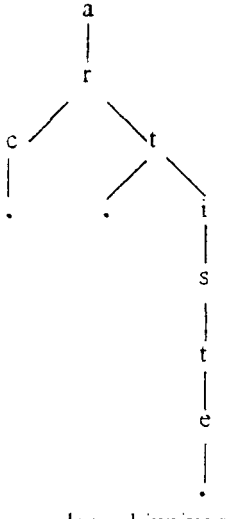
5) Combien de comparaisons mettant en jeu un élément de la matrice M effectue votre algorithme ?

II- Codage de mots en parties communes.

Le problème consiste à gérer un dictionnaire en codant les mots en partie commune à l'aide d'une arborescence.
Exemple : les mots *arc* et *artiste* seront codés par l'arbre suivant :



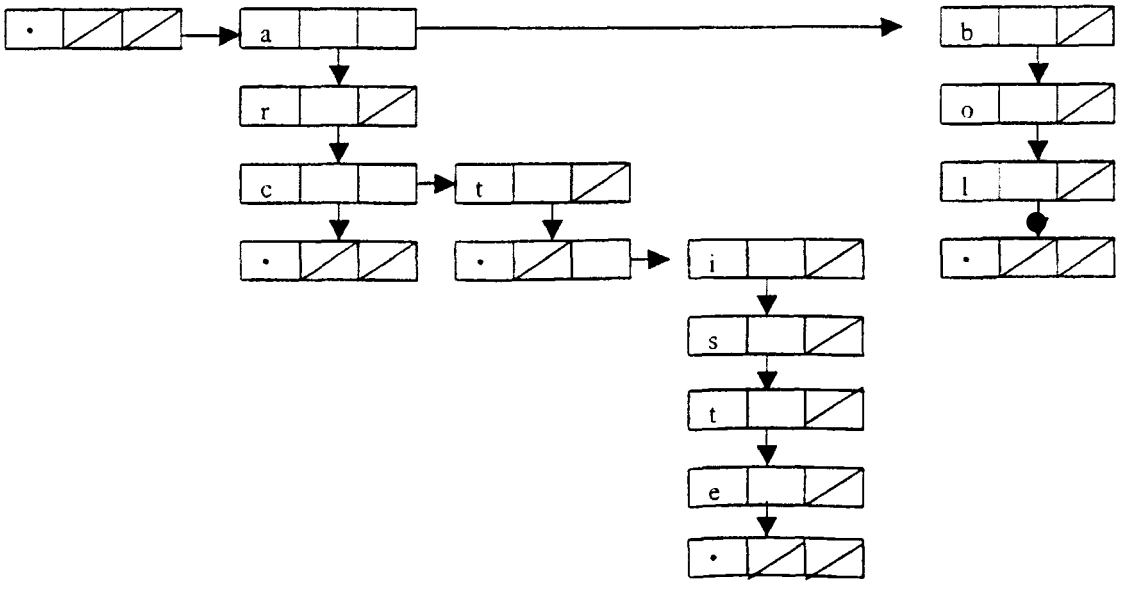
Afin de pouvoir coder avec une telle structure de données des mots qui sont préfixes d'autres mots déjà placés, on rajoute systématiquement un point à la fin de chaque mot. Ainsi les mots arc, art, artiste seront codés par :



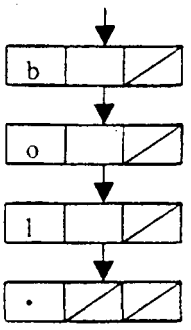
Il y a correspondance biunivoque entre les branches de l'arbre et les mots qui y ont été placés. Pour représenter une telle structure de données on associe à chaque nœud de l'arborescence un triplet dont la première composante contient le caractère associé au nœud, la seconde composante permet d'accéder au fils aîné de ce nœud et la troisième permet l'accès au frère cadet.

On suppose que la racine de l'arbre contient le caractère '.', ainsi dès le départ le dictionnaire contient le mot vide cela simplifie l'algorithme d'insertion car on évite de se poser la question de savoir si le dictionnaire est vide ou pas.

Le caractère '.' étant inférieur d'après l'ordre lexicographique à n'importe quelle lettre, on peut schématiser l'arborescence qui respecte l'ordre alphabétique et qui contient les mots *arc*, *art*, *artiste*, *bol* par :



Ecrire la fonction *creer_branche* qui prend pour argument un mot *m* et retourne pour résultat l'arbre réduit à la branche qui code le mot *m*. Si le mot est le mot *bol*, la fonction retourne pour résultat l'adresse de l'arbre représenté par la branche :



1) Ecrire la fonction *placer* qui a pour arguments un mot *m* et un arbre *a*, cette fonction retourne pour résultat l'arbre auquel le mot *m* a été ajouté. Dans cette question on ne suppose pas que l'arbre respecte l'ordre alphabétique, le mot *m* est simplement ajouté à l'arborescence et codé en partie commune sans tenir compte de l'ordre alphabétique.

2) Ecrire la fonction *inserer* qui a pour arguments un mot *m* et un arbre *a*, cette fonction retourne pour résultat l'arbre auquel le mot *m* a été ajouté. Ici on suppose que l'arbre *a* contient les mots rangés dans l'ordre alphabétique, l'arbre résultat après insertion du mot *m* doit posséder la même propriété.

3) Ecrire la procédure *affiche* qui affiche dans l'ordre alphabétique tous les mots contenus dans un arbre créé avec la fonction *inserer*. Le(s) paramètre(s) de cette procédure est (sont) à préciser

4) Ecrire la fonction booléenne *présent* qui a pour argument un mot *m* et arbre *a*. cette fonction renvoie pour résultat le booléen *vrai* si le mot *m* est présent dans l'arbre *a*, le booléen *faux* sinon.

5) Ecrire la séquence d'instructions qui permet de saisir en lecture des mots et de les insérer au fur et à mesure dans l'arbre dont la racine se nomme *dico*. La fin de la lecture intervient lorsque le mot *fin* est lu.

III- Expressions préfixées bien formées.

On se propose ici d'énumérer et de dénombrer les expressions préfixées d'une longueur fixée que l'on peut écrire avec un opérateur binaire noté + et une seule variable notée *a*. Par exemple les seules expressions préfixées bien formées de longueur 5 sont : *+a+aa* et *++aaa* correspondant respectivement aux expressions parenthésées *a+(a+a)* et *(a+a)+a*.
On associe à toute expression, construite avec les seuls symboles + et *a*, un mot $u=(u_1,u_2,\cdots,u_n)$ avec n≥1 et $u_i \in \{-1,1\}$ pour *i*=1,...,*n*. A chacun des deux symboles on associe son arité diminuée de 1. Ainsi à l'opérateur binaire + est associé le poids 1 et à la variable *a* (opérateur d'arité 0) le poids −1.
Aux deux expressions bien formées ci-dessus correspondent respectivement les mots : (1-11-1-1) et (11-1-1-1).
On démontre qu'un mot de longueur *n* sur $\{-1,1\}$ correspond à une expression préfixée bien formée si et seulement si

$$\sum_{i=1}^{n} u_i = -1 \text{ et } \sum_{i=1}^{k} u_i \geq 0 \text{ pour } 1 \leq k \leq n-1 \quad (1)$$

Par analogie, on dira que des mots qui possèdent la propriété ci-dessus qu'ils sont des **mots bien formés**.

Quel que soit le langage choisi un mot u est représenté dans la suite de cet énoncé par un **tableau d'entiers dont les composantes sont indexées à partir de 0**, la composante d'indice 0 contient la longueur du mot.
Ainsi le mot bien formé 1-1-1 est représenté par le tableau u, avec $u_0=3, u_1=1, u_2=-1, u_3=-1$.
La longueur d'un mot *u*, notée |*u*|, est le nombre de ses composantes.

1) Quelle est la parité de la longueur d'un mot bien formé ?

2) Démontrer que si $u=(u_1,u_2,\cdots,u_n)$ et $v=(v_1,v_2,\cdots,v_m)$ sont des mots bien formés, alors $w=(1,u_1,u_2,\cdots,u_n,v_1,v_2,\cdots,v_m)$ est un mot bien formé.

3) Démontrer que réciproquement, si $w = (w_1, w_2, \cdots, w_n)$ est mot bien formé (n>1) alors il existe un unique couple $u = (u_1, u_2, \cdots, u_l)$ et $v = (v_1, v_2, \cdots, v_r)$ de mots bien formés tels que $w = (1, u_1, u_2, \cdots, u_l, v_1, v_2, \cdots, v_r)$.

4) Ecrire la fonction booléenne *est_bien_forme*, qui a pour argument un mot u et pour résultat le booléen *vrai* si le mot u satisfait la propriété (1), *faux* sinon.

5) Ecrire suivant le langage choisi.:
   - Pour PASCAL, la procédure *construire* qui prend comme paramètres de données deux mots bien formés u et v et a pour paramètre résultat le mot bien formé w dont il est question en **2).**
   - Pour CAML, la fonction *construire* qui prend comme paramètres deux mots bien formés u et v et a pour résultat le mot bien formé w dont il est question en **2).**

6) Ecrire la fonction *fusion* qui a pour paramètres deux listes l_1 et l_2 de mots bien formés, et retourne pour résultat la liste de tous les mots bien formés obtenu par « construction » (à l'aide de la primitive *construire* qui est demandée à la question qui précède) à partir d'un mot de la liste l_1 suivi d'un mot de la liste l_2. Si la première liste a n éléments et la seconde p éléments, la liste résultat aura np éléments.

7) Ecrire la procédure *enumere* qui a pour argument un nombre entier p et pour effet de remplir le tableau de liste de mots t (global) de façon telle que la $j$ème composante de ce tableau contienne la liste de tous les mots bien formés de longueur exactement 2*j+1.

8) Soit $u = (u_1, u_2, \cdots, u_n)$ un mot tel que $\sum_{i=1}^{n} u_i = -1$ et i le plus petit indice ($1 \le i \le n$) tel que $\sum_{k=1}^{i} u_k$ soit minimum, démontrer que $v = (u_{i+1}, \cdots, u_n, u_1, \cdots, u_i)$ est l'unique mot bien formé déduit de u par une permutation circulaire.

9) On peut démontrer (on ne le demande pas) que si u et v sont deux mots déduits l'un de l'autre par une permutation circulaire tels que $\sum_{i=1}^{n} v_i = \sum_{i=1}^{n} u_i = -1$ alors $u \neq v$. En déduire le nombre C_k de mots bien formés de longueur $2k+1$.

Rappels de quelques fonctions standard de PASCAL :

- La fonction `length` a pour résultat la longueur de la chaîne passée en paramètre.
- L'opérateur `infixé` de concaténation de chaînes est noté +.
- La fonction `copy(ch, i, lg)` renvoie pour résultat la sous-chaîne extraite de la chaîne `ch` qui commence au caractère de rang `i` et qui a pour longueur `lg`.
- Si `ch` est une variable chaîne, ch[i] désigne le ième caractère de la chaîne `ch`.

I) Parcours en escalier dans un tableau.

Déclarations :

```
type
     matrice = array[0..20,0..20] of longint ;
```

En-tête de la fonction demandée :

```
Function pgcd (m :matrice ;i,j,k,l :integer) :integer ;
```

II) Codage de mots en parties communes.

Déclarations :

```
Type
     pnoeud = ^noeud;
      noeud = record
                 info : char ;
                 fils,frere :pnoeud
              end ;
Var
     mot : string ;
    dico : pnoeud;
```

En-tête des fonctions et procédures demandées :

```
Function creer_branche(m :string) :pnoeud;
Function placer(m :string ;a :pnoeud) :pnoeud;
Function inserer(m :string ;a :pnoeud) :pnoeud;
Procedure affiche.....;
Function present(m :string ;a :pnoeud) :boolean;
```

III) Expressions préfixées bien formées.

Déclarations :

```
Type
                mot = array[0..21] of integer;
        liste_mots = ^doublet ;
           doublet = record
                        info : mot ;
                        suiv : liste_mots
                     end ;
   tableau_de_listes_mots = array[0..10] of liste_mots ;

Var
    t : tableau_de_listes_mots ;
```

En-tête des fonctions et des procédures demandées :

```
Function est_bien_forme(u :mot) : boolean ;
Procedure construire(u,v :mot ;var v :mot) ;
Function fusion(l1,l2 :liste_mots) :liste_mots ;
Procedure enumere (p :integer) ;
```

On suppose disponible la fonction `conc(l1,l2 :liste_mots)` qui réalise la concaténation de deux listes.

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`.
Si `s` est une chaîne `s.[i]` désigne le caractère de rang `i` de la chaîne `s`.

`string_length : string -> int` renvoie la longueur de la chaîne donnée.
`sub_string : string -> int -> int -> string, sub_string s d l` renvoie une nouvelle chaîne de longueur `l` contenant les caractères de la chaîne `s` à partir du caractère numéro `d`.
`préfix ^ : string -> string -> string, s1^s2` renvoie une nouvelle chaîne concaténée des chaînes `s1` et `s2`.
`read_line : unit -> string` lit des caractères sur l'entrée standard jusqu'à rencontrer un saut de ligne. Renvoie la chaîne de tous les caractères lus, sans le caractère saut de ligne final.
`préfix @ : 'a list -> 'a list -> 'a list, l1 @ l2` renvoie une nouvelle liste concaténée des listes `l1` et `l2`.
`list_length : 'a list ->int` renvoie le nombre d'éléments de la liste donnée.

I) Parcours en escalier dans un tableau.

fonction demandée :

```
pgcd :'a vect vect -> int -> int-> int -> int -> 'a = <fun>
```

II) Codage de mots en parties communes.

Type et valeur globales :

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

let dico = ref(noeud('.',nil,nil));;
```

liste des fonctions et procédures demandées :

```
creer_branche : string -> arbre = <fun>
placer : string -> arbre -> arbre = <fun>
inserer : string -> arbre -> arbre = <fun>
affiche :.....-> unit = <fun>
present : string -> arbre -> bool = <fun>
```

III) Expressions préfixées bien formées.

Valeur globale :

```
Let u= make_vect 20 0 ;;
```

Liste des fonctions et procédures demandées :

```
est_bien_forme : int vect -> bool =< fun>
construire : int vect -> int vect -> int vect = <fun>
fusion : int vect list -> int vect list -> int vect list = <fun>
enumere : int -> int vect list vect = <fun>
```