M2 CCI – Algorithmique – Examen

Durée 3h, sans documents **5 mars 2024**

NE PAS RECOPIER les énoncés des questions. Ne pas perdre de temps à un soin excessif de la présentation. Les questions sont indépendantes. Le barème est indicatif.

1. Les plateaux d'une séquence d'entiers

Étant donnée une séquence d'entiers **non vide**, nommée **S**, un *plateau* de **S** est défini comme une sous-séquence d'éléments adjacents de **S** ayant une même valeur **h** et délimitée à gauche et à droite, soit par les extrémités de **S**, soit par un élément ayant une valeur différente de **h** ; **h** est appelée la *hauteur* du plateau.

Par exemple, dans la séquence **S = [4, 4, -1, 5, 5, 5, 0, 0, 7, 7, 2, 2, 2]**, on trouve successivement un plateau de **hauteur 4** et de **longueur 2**, puis un plateau de **hauteur -1** et de **longueur 1**, puis un plateau de **hauteur 5** et de **longueur 3**, etc…

Dans l'exemple ci-dessus, la longueur maximale des plateaux est **3**, et il y a **2** plateaux de **longueur 3**, l'un de **hauteur 5** et l'autre de **hauteur** 2.

Q1 [5 points]

La séquence non vide **S** est représentée sous forme contiguë dans un tableau **T**, avec longueur explicite **L**, selon le lexique suivant :

n : constante de type entier > 0
T : tableau sur [1…n] d'entiers
L : un entier sur [1…n]

(i) Étude du premier plateau

— Donner un algorithme qui affiche la hauteur et la longueur du premier plateau de la séquence représentée dans le tableau **T** et de longueur explicite **L**.

Dans l'exemple ci-dessus, le résultat affiché est : hauteur = 4, longueur = 2

(ii) Nombre de plateaux de longueur maximale.

— Donner un algorithme qui affiche le nombre de plateaux de longueur maximale de la séquence représentée dans le tableau **T** et de longueur explicite **L**.

Dans l'exemple ci-dessus, le résultat affiché est : nombre de plateaux de longueur max = 2.

(iii) Les hauteurs des plateaux de longueur maximale

On veut construire une séquence d'entiers formée des hauteurs des plateaux de longueur maximale de la séquence représentée dans le tableau **T** et de longueur explicite **L**.

Dans l'exemple ci-dessus, le résultat cherché est la séquence **[5, 2]** de longueur **2**.

— Donner les modifications qu'il faut apporter à l'algorithme obtenu en **Q1(ii)** pour qu'il construise cette séquence des hauteurs des plateaux de longueur maximale dans un tableau **TH** de longueur **LH**.

2. Somme de polynômes

On considère des polynômes en **x** à coefficients entiers, par exemple $\mathbf{-34x^4 + 2x^2 + 10x - 12}$. Un monôme, par exemple $\mathbf{-34x^4}$, est caractérisé par son coefficient (-34), et sa puissance de **x** (4). Un polynôme est décrit par une séquence de monômes *à coefficients non nuls et en ordre décroissant des puissances* ; un monôme est décrit par un couple <coefficient, puissance>. Ainsi, le polynôme de l'exemple ci-dessus est décrit par la séquence [<-34, 4>, <2, 2>, <10, 1>, <-12, 0>]. Le polynôme nul est représenté par la séquence vide.

Q2 [6 points]

On étudie la somme de deux polynômes. Par exemple, la somme des polynômes

$-34x^4 + 2x^2 + 10x - 12$ et $3x^5 + 4x^4 - 2x^2$ est $3x^5 - 30x^4 + 10x - 12$.

Les polynômes étant **sous forme de séquences triées**, leur somme est construite en appliquant le ***principe d'interclassement*** vu en cours.

(i) Etude fonctionnelle abstraite

On fixe le lexique suivant, faisant abstraction de la représentation des séquences :

Monôme : type <Coef : entier ≠ 0, Puis : entier ≥ 0 >

Polynôme : type séquence de Monôme en ordre décroissant des puissances

SomPol : fonction (P, Q : Polynôme) → Polynôme

{Polynôme valant la somme des polynômes P et Q.}

Exemple :

SomPol([<-34, 4>, <2, 2>, <10, 1>, <-12, 0>], [<3, 5>, <4, 4>, <-2, 2>])

= [<3, 5>, <-30, 4>, <10, 1>, <-12, 0>]

— Donner des **équations de récurrence** définissant la fonction **SomPol**.

(ii) Représentation chaînée

Dans ce qui suit, les séquences représentant les polynômes sont **sous forme chaînée (représentation standard)** et on étudie une action nommée **Sommer** :

adCelM : type pointeur de CelM

CelM : type <M : Monôme ; Suc : adCelM>

Sommer : action (donnée-résultat TP, TQ : adCelM)

{A l'état initial TP et TQ sont les têtes des listes représentant deux polynômes P et Q. À l'état final, TP est la tête de la liste représentant le polynôme somme des polynômes P et Q, et ***TQ vaut Nil****.*

***Conditions de réalisation** : l'algorithme procède par **modification des liens de chaînage** : le polynôme résultat dans TP est formé de cellules des polynômes donnés ; **les cellules non utilisées sont libérées**.}*

— Donner une **réalisation récursive** de l'action **Sommer**.

3. Dictionnaire arborescent

On considère un ***dictionnaire*** composé d'un ensemble de **mots distincts**, comme ci-dessous :

— A chaque mot correspond la séquence de ses caractères, terminée par le caractère *'*'*.

— Deux mots peuvent avoir un préfixe commun : par exemple les mots *arbre* et *arts* ont *ar* pour plus long préfixe commun.

— Il est possible qu'un mot complet soit le préfixe d'un autre mot. Par exemple, le mot *art* est le début du mot *arts*. Le caractère *'*'*, qui marque la fin des mots, permet alors de distinguer la fin du mot *art* de la suite du mot *arts*.

Le dictionnaire est représenté par un **arbre binaire de caractères**. La racine de l'arbre correspond au premier caractère du premier mot.

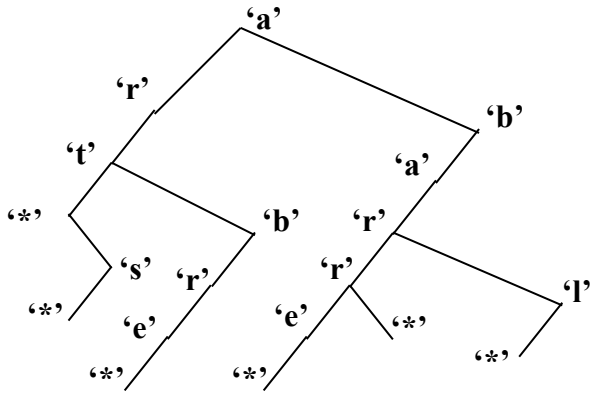
Chaque nœud peut avoir deux fils :

— Un fils gauche ou *successeur* : dans le mot *art*, la lettre '*a'* est suivie de la lettre '*r'*, suivie de la lettre '*t'*, suivie de *'*'*.

— Un fils droit ou *alternant* : après le préfixe *ar*, la lettre '*b'* (venant de *arbre*) est un alternant de la lettre '*t'* (venant de *art*), *'b'* est le fils droit de *'t'* (alternant de la lettre *'t'* après *ar*) ; après le préfixe *art*, la lettre '*s'* (venant du mot *arts*) est un alternant du caractère '**'* (marquant la fin du mot *art*),), *'s'* est le fils droit de *'*'* (alternant de la lettre *'*'* après *art*) ; etc.

Remarques : un sous-arbre gauche n'est vide que si la racine est une étoile (elle n'a pas de successeur, l'étoile est la fin d'un mot). Un sous-arbre droit n'est vide que s'il n'y a pas d'alternant à la racine.

Exemple d'un dictionnaire contenant les mots *art, arts, arbre, barre, bar, bal* **:**



Q3 [3,5 points] Etude fonctionnelle abstraite

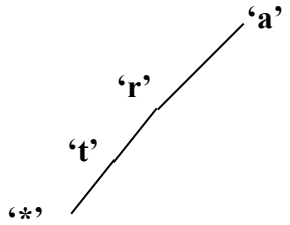
(i) Construction d'un dictionnaire formé d'un seul mot

On spécifie la fonction **LeDico** de construction d'un dictionnaire formé d'un seul mot.

LeDico : fonction (M : séquence de caractère) → arbre binaire de caractère

{Crée un dictionnaire constitué du seul mot M. ***Le mot M est marqué dans le dictionnaire par '*' comme dernier caractère.****}*

Exemple : pour le mot *art*, l'arbre construit est le suivant :



— Donner des **équations de récurrence** définissant la fonction **LeDico**.

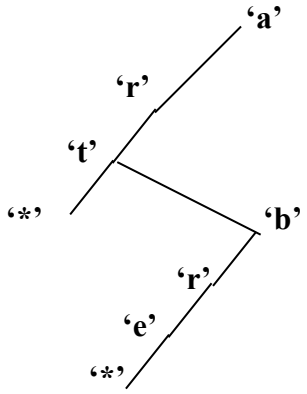
(ii) Ajout d'un mot à un dictionnaire

On spécifie la fonction **Plus** d'ajout d'un mot à un dictionnaire.

Plus : fonction (M : séquence de caractère, D : arbre binaire de caractère) → arbre binaire de caractère

{Construit le dictionnaire D' constitué de l'ajout du mot M au dictionnaire D. Le résultat est le dictionnaire D si le mot M lui appartient déjà.}

Exemple : l'ajout de *arbre* dans l'arbre précédent donne l'arbre suivant :



— Donner des **équations de récurrence** définissant la fonction **Plus**.

Q4 [3 points] Représentation chaînée

Les arbres binaires de caractères sont représentés sous forme chaînée.

Pour cela, on définit le lexique suivant :

adNœud : type pointeur de Nœud

Nœud : type <R : caractère ; G : adNœud ; D : adNœud>

Pour des raisons de simplification, on conserve par contre une représentation abstraite des mots, et donc les sélecteurs usuels sur les séquences (**premier**, **fin, EstVide?**, etc…).

(i) Construction d'un dictionnaire formé d'un seul mot

On spécifie l'action **CréerDico** de construction d'un dictionnaire chaîné formé d'un seul mot.

CréerDico : action (donnée M : séquence de caractère; résultat Z : adNœud)

*{À l'état final, Z est la racine de l'arbre représentant le dictionnaire contenant le seul mot M. **Le mot M est marqué dans le dictionnaire par '*' comme dernier caractère.**}*

— Donner une **réalisation récursive** de l'action **CréerDico**, s'inspirant des équations de **LeDico**.

(ii) Ajout d'un mot à un dictionnaire

On spécifie l'action **Ajout** d'ajout d'un mot à un dictionnaire.

Ajout : action (donnée M : séquence de caractère; donnée-résultat Z : adNœud)

{A l'état initial, Z est la racine d'un arbre représentant un dictionnaire. À l'état final, Z est la racine de l'arbre représentant le dictionnaire ajouté du mot M, s'il n'est pas déjà présent. S'il est déjà présent, Z est inchangé.}

— Donner une **réalisation récursive** de l'action **Ajout**, s'inspirant des équations de **Plus**.

4. Suppression dans une forêt

Étant donnés une forêt **F** et un élément **E**, on étudie la suppression de **tous** les arbres ou sous-arbres de **F** ayant **E** pour racine : un arbre de racine **E** est supprimé soit parce que c'est un arbre de la forêt **F**, soit parce que c'est l'un des sous-arbres des arbres de la forêt **F**.

Q5 [2,5 points]

(i) Étude fonctionnelle

On spécifie une fonction **Moins_F** :

Élément : type

Moins_F : fonction (E : Élément, F : Forêt d'Élément) → Forêt d'Élément

{Forêt construite à partir de F en supprimant tous les arbres ou sous-arbres de F ayant E pour racine. S'il n'en existe pas, Moins_F(E, F) = F.}

— Donner les **équations de récurrence** définissant la fonction **Moins_F**.

(ii) Représentation chaînée

Les forêts sont représentées sous forme chaînée :

Nœud : type <R : Élément ; Fils, Frère : adNœud>

adNœud : type pointeur de Nœud

On spécifie une action **Supprimer_F** :

Supprimer_F : action (donnée E : Élément ; donnée-résultat X : adNœud)

*{Supprime tous les arbres ou sous-arbres ayant E pour racine dans la forêt d'adresse X. **Les cellules non utilisées sont restituées à la mémoire libre**.}*

On spécifie de plus une action nommée **Libérer_F** :

Libérer_F : action (donnée-résultat X : adNœud)

{Restitue à la mémoire libre toutes les cellules de la forêt d'adresse X. À l'état final, X = Nil.}

— Donner une **réalisation récursive** de l'action **Supprimer_F** en utilisant **Libérer_F**.

<u>On ne demande pas de réaliser la fonction Libérer_F.</u>

M2 CCI – Algorithmique

AL – Examen : des exemples de solutions **5 mars 2024**

1. Les plateaux d'une séquence d'entiers

Q1

(i) [1.5 points]

La hauteur du premier plateau est la valeur du premier élément de la séquence donnée. On recherche le premier élément différent du premier.

```
i : entier sur [2…L+1]                          {indice courant pour la recherche}
i ← 2
tant que i ≠ L +1 et puis T_i = T_1
    i ← i + 1
Écrire("hauteur = ", T_1, ", longueur = ", i-1)
```

(ii) [2.5 points]

version 1

La limite entre deux plateaux est caractérisée par un couple d'éléments consécutifs de hauteurs différentes. Dans un parcours de cette suite de couples, on maintient la longueur maximale de plateau déjà rencontrée et le nombre de plateaux rencontrés ayant cette longueur. Ces quantités sont mises à jour au moment du traitement d'un couple de valeurs différentes

```
lg : entier sur [1…n]                          {longueur du plateau courant}
lgm : entier sur [0…n]                         {longueur maximale des plateaux déjà rencontrés}
nb : entier sur [0…n]                          {nombre de plateaux de longueur maximale déjà rencontrés}
nb ← 0 ; lgm ← 0 ; lg ← 1
i parcourant [2…L]
    si T_i = T_i-1 alors lg ← lg+1             {T_i-1 est la hauteur du plateau courant}
    sinon {traitement du plateau courant}
        selon lg, lgm
            lg < lgm :
            lg = lgm : nb ← nb+1
            lg > lgm : lgm ← lg ; nb ← 1
        lg ← 1
{traitement du dernier plateau}
selon lg, lgm
    lg < lgm :
    lg = lgm : nb ← nb+1
    lg > lgm : lgm ← lg ; nb ← 1
Écrire ("nombre de plateaux de longueur max = ", nb)
```

version 2

L'algorithme est construit en raisonnant sur la séquence des plateaux constituant la suite donnée. Comme dans la version 1, on parcourt la suite donnée en maintenant la longueur maximale de plateau déjà rencontrée et le nombre de plateaux rencontrés ayant cette longueur.

```
h : entier ; lg : entier                       {hauteur et longueur du plateau courant}
p : entier sur [1…n]                           {position du premier élément du plateau courant}
i : entier sur [2…n+1]                         {indice courant pour le parcours de T}
lgm : entier sur [0…n]                         {longueur maximale des plateaux déjà rencontrés}
nb : entier sur [0…n]                          {nombre de plateaux de longueur maximale déjà rencontrés}
nb ← 0
lgm ← 0
i ← 1
```

```
    tant que i ≠ L+1
        {i est la position du premier élément du prochain plateau}
            h ← T_i ; p ← i ; i ← i+1
            tant que i ≠ L+1 et puis T_i = h
                i ← i + 1
            lg ← i - p
            selon lg, lgm
                lg < lgm :
                lg = lgm : nb ← nb+1
                lg > lgm : lgm ← lg ; nb ← 1
Écrire ("nombre de plateaux de longueur max = ", nb)
```

(iii) [1 point]

TH : tableau sur [1…n] d'entiers ; LH : un entier sur [1…n]

Il s'agit de construire **TH** lors du traitement d'un plateau (**LH** remplace **nb**) :

```
    selon lg, lgm
            lg < lgm :
            lg = lgm : LH ← LH +1 ; TH_LH ← h                    {ajout en queue}
            lg > lgm : LH ← 1 ; TH_1 ← h
```

2. Somme de deux polynômes

Q2

(i) Equations de récurrence de SomPol [3 points]

(1) SomPol([], []) = []
(2) SomPol(m $_0$ P, [])= m $_0$ P
(3) SomPol([], m $_0$ P) = m $_0$ P
(4) SomPol(m1 $_0$ P1, m2 $_0$ P2) =
    selon m1.Puis, m2.Puis
        (4.1) m1.Puis > m2.Puis : m1 $_0$ SomPol(P1, m2 $_0$ P2)
        (4.2) m1.Puis < m2.Puis : m2 $_0$ SomPol(m1 $_0$ P1, P2)
        (4.3) m1.Puis = m2.Puis :
            soit SC = m1.Coef + m2.Coef, SP=SomPol(P1,P2)
            dans si SC = 0 alors SP sinon <SC, m1.Puis> $_0$ SP

(ii) Réalisation récursive de Sommer [3 points]

```
Sommer(TP, TQ) :
    Z : adCelM
    selon TP, TQ :
{1, 2}   TQ = Nil : {rien}
{3}      TP = Nil et TQ ≠ Nil : TP ← TQ ; TQ ← Nil
{4}      TP ≠ Nil et TQ ≠ Nil
            selon TP↑.M.Puis, TQ↑.M.Puis
{4.1}               TP↑.M.Puis > TQ↑.M.Puis : Sommer(TP↑.Suc, TQ)
{4.2}               TP↑.M.Puis < TQ↑.M.Puis :
                        Sommer(TP, TQ↑.Suc)
                        TQ↑.Suc ← TP ; TP ← TQ      {ajout en tête du premier de Q}
                        TQ ← Nil
{4.3}               TP↑.M.Puis = TQ↑.M.Puis :
                        Sommer(TP↑.Suc, TQ↑.Suc)
                        TP↑.M.Coef ← TP↑.M.Coef + TQ↑.M.Coef
                        si TP↑.M.Coef = 0 alors
                            Z ← TP ; TP ← TP↑.Suc ; Libérer(Z)
                        Libérer(TQ) ; TQ ← Nil
```

{variante pour le cas 4.2 :
Z ← TQ↑.Suc ; TQ↑.Suc ← TP ; TP ← TQ ; TQ ← Z ; Sommer (TP↑.Suc, TQ)

3. Dictionnaire arborescent

Q3 Etude fonctionnelle
(i) Construction d'un dictionnaire ne comportant qu'un seul mot [1 point]
(1) LeDico([]) = //'*'\\
(2) LeDico(c_oM) = //LeDico(M), c\\

(ii) Ajout d'un mot à un dictionnaire [2,5 points]
(1) Plus (M, /\) = LeDico(M)
(2) Plus ([], /G, r, D\) = si r='*' alors /G, r, D\ sinon /G, r, Plus(['*'], D)\
(3) Plus (c_oM, /G, r, D\) = *{c ne peut pas être l'étoile}*
si c=r alors /Plus(M, G), r, D\ sinon /G, r, Plus (c_oM, D)\

Q4 Réalisation chaînée
(i) Construction d'un dictionnaire ne comportant qu'un seul mot [1,5 points]

```
CreerDico (M, X) :
  Si EstVide?(M) alors
     Allouer (X)
     X↑.R ← '*'
     X↑.G ← Nil
     X↑.D ← Nil
  Sinon
     Allouer (X)
     X↑.R ← premier(M)
     CreerDico (fin(M), X↑.G)
     X↑.D ← Nil
```

(ii) Ajout d'un mot à un dictionnaire [1,5 points]

```
Ajout (M, X) :
  Si X = Nil alors
     CreerDico (M,X)
  Sinon
     Si EstVide?(M) alors
        Si X↑.R ≠ '*' alors
           Ajout (['*'], X↑.D)
     sinon
        si  X↑.R = premier(M) alors
           Ajout (fin(M), X↑.G)
        Sinon
           Ajout (M, X↑.D)
```

4. Suppression dans une forêt

Q5
(i) Étude fonctionnelle [1 points]
(1) Moins_F(E, [ ]) = [ ]
(2) Moins_F(E, <r, F1>$_o$F2) = si r=E alors Moins_F(E, F2)
sinon <r, MoinsF(E, F1)> $_o$ Moins_F(E, F2)

(ii) Représentation chaînée [1,5 points]

```
Supprimer_F(E, X) :
    Z : adNœud
    si X ≠ Nil alors
        Supprimer_F(E, X↑.Frere)
        si X↑.R ≠ E alors
            Supprimer_F(E, X↑.Fils)
        sinon {suppression du premier arbre}
            Z ← X
            X ← X↑. Frère
            Libérer_F(Z↑.Fils)
            Libérer(Z)
```