M2 CCI, M1 GEOMAS – Algorithmique – Examen

Durée 3h, sans documents **7 mars 2023**

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. À propos de "mots doubles" (séquences contiguës, traitement itératif)

Q1 [4 points]

Un mot est une séquence de caractères. On dit qu'un mot **M** est *double* si et seulement s'il existe un mot **x** tel que **M = x & x**. Autrement dit, la longueur d'un mot double **M** est nécessairement paire, soit **L=2K,** et $\mathbf{M_{[1...K]} = M_{[K+1...L]}}$. Par exemple, a**a**, abc**abc**, abcabc**abcabc** sont des mots doubles, respectivement de longueurs 2, 6 et 12 ; par contre, aab, abcabcabc ne sont pas des mots doubles.

Les mots sont donnés dans des tableaux **sous forme contiguë avec longueur explicite**.

(i) M est-il un mot double ?

On spécifie la fonction suivante :

Lmax : constante de type entier > 0 *{longueur maximum des mots considérés}*

EstDouble : fonction (T : tableau sur [1…Lmax] de caractère ; L : entier sur [0…Lmax]) → booléen

{vrai ssi le mot de longueur L représenté dans le tableau T est un mot double}

Exemples : EstDouble(['a','b','c','a','b','c'], 6) = vrai
EstDouble(['a','a','b'], 3) = faux
EstDouble(['a','b','b','a'], 4) = faux

— Donner une **réalisation itérative** de la fonction **EstDouble**.

(ii) Construction d'un mot double

On veut construire le mot double d'un mot donné. On spécifie l'action suivante :

ConstruireDouble : action (donnée T1 : tableau sur [1…Lmax] de caractère,
L1 : entier sur [0…Lmax],
résultat T2 : tableau sur [1…Lmax] de caractère,
L2 : entier sur [0…Lmax],
OK : booléen)

{Construit dans T2 le mot double de longueur L2, formé de la concaténation de deux instances du mot de longueur L1 représenté dans T1. OK a la valeur vrai si et seulement si cette opération est possible (c'est-à-dire si le double de L1 est inférieur ou égal à Lmax). Lorsque OK a la valeur faux, les valeurs de T2 et L2 ne sont pas pertinentes.}

Exemples :

ConstruireDouble(['a','b','c'], 3, T2, L2, OK) affecte à T2 la valeur ['a','b','c','a','b','c'], à L2 la valeur 6, et à OK la valeur vrai, si Lmax = 10.

ConstruireDouble(['a','b','c'], 3, T2, L2, OK) affecte à OK la valeur faux, si Lmax = 5, et ne donne pas de valeur ni pour T2 ni pour L2.

Pour cela, on doit recopier deux fois le mot donné. On procède par concaténation. On spécifie l'action intermédiaire suivante :

Concaténer : action (donnée T1 : tableau sur [1…Lmax] de caractère,
L1 : entier sur [0…Lmax],
donnée-résultat T2 : tableau sur [1…Lmax] de caractère,
L2 : entier sur [0…Lmax])

{Concatène le mot de longueur L1 représenté dans T1 après le mot de longueur L2 représenté dans T2. Pré-condition : L1+L2 ≤ Lmax. }

Exemple :
Si T2 contient [‘a’,’b’,’c’] et L2 = 3, alors Concatener([‘d’,’e’], 2, T2, L2) transforme T2 en [‘a’,’b’,’c’,’d’,’e’], et L2 devient 5.

— Donner une **réalisation itérative** de **Concaténer.**
— Donner une **réalisation itérative** de **ConstruireDouble** en utilisant **Concaténer**.

2. Répartition d'une liste en deux listes **(séquences contiguës, traitement itératif, séquences chaînées, traitement itératif et récursif)**

On étudie la répartition des éléments d'une séquence d'entiers selon leur signe.

Q2 [9 points]

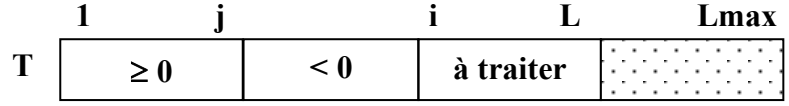
(i) Représentation contiguë, traitement itératif

Les séquences sont représentées sous forme contiguë dans un tableau avec longueur explicite. On donne le lexique suivant :

Lmax : constante de type entier > 0 *{longueur maximum des séquences d'entiers}*
Réarranger : action (donnée-résultat T : tableau sur [1…Lmax] d'entiers ;
donnée L : entier sur [0…Lmax]>)
{Réarrange les éléments de T donné de telle sorte qu'à l'état final, on trouve d'abord ses éléments positifs ou nuls, dans l'ordre initial, puis ses éléments négatifs, dans l'ordre initial. Par exemple, si à l'état initial T contient les éléments [-2, -5, 3, -1, 4, 25, 0, -3, 4], alors à l'état final, T contient les éléments [3, 4, 25, 0, 4, -2, -5, -1, -3]. La longueur L est inchangée entre état initial et état final.}

— Donner une réalisation **itérative** de l'action **Réarranger**.

Idée : dans un parcours de la séquence, l'élément courant est éventuellement déplacé en maintenant l'invariant illustré par la figure ci-dessous : les éléments déjà traités se trouvent entre les positions **1** et **i-1** ; parmi eux les éléments positifs ou nuls se trouvent entre **1** et **j** dans l'ordre initial et les éléments négatifs entre **j+1** et **i-1**, dans l'ordre initial.



(ii) Représentation chaînée, copie des éléments négatifs, traitement itératif

Les séquences sont maintenant représentées sous forme chaînée standard. On donne le lexique suivant :

adCelE : type pointeur de CelE
CelE : type <E : entier, Suc : adCelE>

CopierNeg : action (donnée TS : adCelE, résultat TN : adCelE)
{Construit une nouvelle liste de tête TN formée des éléments négatifs de la liste de tête TS dans l'ordre où ils sont dans la liste de tête TS. La liste de tête TS n'est pas modifiée.}

— Donner une réalisation **itérative** de l'action **CopierNeg**. Pour créer les cellules de la nouvelle liste, utiliser la primitive **Allouer**.

(iii) Représentation chaînée, réarrangement en deux listes, traitement récursif

— Donner une réalisation **récursive** de l'action suivante (toute solution itérative sera ignorée lors de la correction) :

Réarranger:action(donnée-résultat TS: adCelE ; résultat TP, TN: adCelE)
{Réarrange les liens entre les cellules de la liste d'adresse de tête TS de manière à les répartir en deux listes selon le signe des éléments.

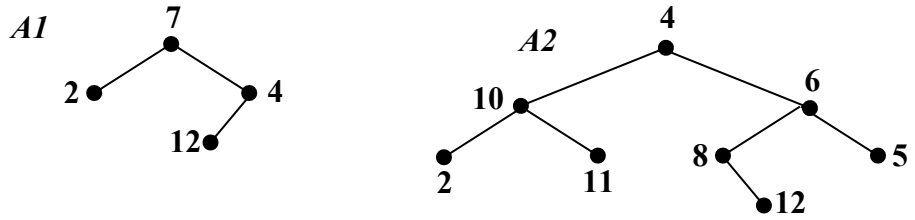
À l'état final :
- *la liste de tête TP est formée des cellules contenant les éléments positifs ou nuls dans le même ordre qu'au départ,*
- *la liste de tête TN est formée des cellules contenant les éléments négatifs dans le même ordre qu'au départ,*
- *TS vaut Nil.*

***L'algorithme procède par modification des liens de chaînage**.}*

3. Les *traversées paires* d'un arbre binaire [7 points] (arbres binaires abstraits et chaînés, traitement récursif)

Une *traversée* d'un arbre est un chemin conduisant *de sa racine à l'une de ses feuilles*. Une *traversée paire* d'un arbre binaire d'entiers positifs **A** est une traversée dont *tous les éléments sont des entiers pairs*.

Dans la figure suivante, les traversées de l'arbre ***A1*** sont les chemins [7, 2] et [7, 4, 12] : aucune des deux traversées n'est paire car les deux contiennent des entiers impairs. Dans l'arbre ***A2***, les traversées [4, 10, 2] et [4, 6, 8, 12] sont paires et les traversées [4, 10, 11] et [4, 6, 5] ne le sont pas.



Dans ce qui suit, on utilisera la fonction suivante que l'on supposera existante :

Pair : fonction (x : entier) → booléen *{vrai si et seulement si x est pair.}*

Q3 Une traversée paire

On spécifie la fonction suivante :

UneTP : fonction (A : arbre binaire d'entier > 0) → séquence d'entier pair

*{Séquence d'entiers de valeur égale à celle de **l'une des** traversées paires de A, ou la séquence vide si A n'a pas de traversée paire ou si A est vide.}*

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

Exemples : UneTP(A1) = [] UneTP(A2) = [4, 10, 2]

Q4 Toutes les traversées paires

On spécifie la fonction suivante :

LesTP : fonction (A : arbre binaire d'entier > 0) → séquence de (séquence non vide d'entier pair)

*{Séquence de **toutes** les traversées paires de A, ou la séquence vide si A n'a aucune traversée paire.}*

Exemples :

LesTP(A1) = []
LesTP(A2) = [[4, 10, 2], [4, 6, 8, 12]]

Pour réaliser cette fonction **LesTP**, on spécifie une fonction supplémentaire :

PlusT : fonction (x : entier, SS : séquence de (séquence non vide d'entier))
→ séquence de (séquence non vide d'entier)

{Formée de toutes les séquences de SS auxquelles on a ajouté x en tête. On n'ajoute rien à SS si SS est vide.}

Exemples :

PlusT(1,[]) = []
PlusT(4,[[10,2],[6,8,12]]) = [[4,10,2], [4,6,8,12]]

— Donner des **équations de récurrence** définissant la fonction **LesTP**, en utilisant **PlusT**.

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

Q5 Copie d'une traversée paire

On veut copier l'une des traversées paires d'un arbre binaire d'entiers (n'importe laquelle parmi celles de l'arbre). Les séquences et les arbres sont sous forme chaînée standard. On donne le lexique suivant :

adNœud : type pointeur de Nœud *{représentation chaînée des arbres binaires d'entiers}*
Noeud : type <R : entier ; G, D : adNoeud>

adCel : type pointeur de Cel *{représentation chaînée des séquences d'entiers}*
Cel : type <E : entier, Suc : adCel>

CopierUneTP : action (donnée X : adNoeud ; résultat Y : adCel)
{Recopie l'une des traversées paires d'un arbre donné, s'il en existe. À l'état initial, X est l'adresse de la racine d'un arbre A ; à l'état final, Y est la tête d'une liste chaînée dont la valeur est une des traversées paires de A. En particulier, Y vaut Nil, si A n'a pas de traversée paire. L'arbre de racine A représenté par X n'est pas modifié par l'action.}

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

M2 CCI, M1 GEOMAS – Algorithmique

Des exemples de solutions **7 mars 2023**

1. À propos de "mots doubles"

Q1

(i) M est-il un mot double ? [1.5 points]

On vérifie que chaque élément de la première moitié du mot est égal à l'élément de même position dans sa deuxième moitié.

Version 1 : application d'un schéma de parcours

```
EstDouble(T, L) :
        EstD : booléen                                  {pour élaborer le résultat}
        K : entier sur [1…Lmax]

        si non EstPair(L) alors EstD ← faux
        sinon   K ← L/2
                EstD ← vrai
                pour i allant de 1 a K
                   EstD ← EstD et Ti=Ti+K
   retour : EstD
```

Version 2 : application d'un schéma de recherche

```
EstDouble(T, L) :
        EstD : booléen                                  {pour élaborer le résultat}
        K : entier sur [1…Lmax]
        i : entier sur [1…Lmax]

        si non EstPair(L) alors EstD ← faux
        sinon   K ← L/2
                i ← 1
                tant que i ≠ K+1 et puis Ti=Ti+K : i ← i + 1
                EstD ← i = K+1
   retour : EstD
```

(ii) Construction d'un mot double [1 point Concatener, 1.5 points ConstruireDouble]

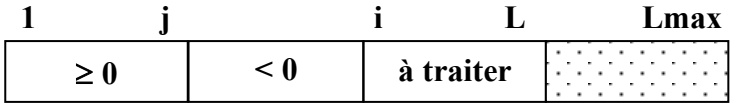
```
Concaténer(T1, L1, T2, L2) :                                    {L1 + L2 ≤ Lmax}
{ parcours de T1 et ajouts successifs de ses caractères en queue de T2. }
        pour i allant de 1 a L1
              L2 ← L2 + 1
              T2L2 ← T1i

ConstruireDouble(T1, L1, T2, L2, OK) :  {on concatène T1,L1 deux fois}
        si 2*L1 > Lmax alors OK ← faux
        sinon
              OK ← vrai
              L2 ← 0
              Concaténer(T1, L1, T2, L2)
              Concaténer(T1, L1, T2, L2)
```

2. Répartition d'une liste en deux listes

Q2

(i) Représentation contiguë [3 points]

Idée 1 : dans un parcours de la séquence, l'élément courant est éventuellement déplacé en maintenant l'invariant illustré par la figure ci-dessous : les éléments déjà traités se trouvent entre les positions **1** et **i-1** ; parmi eux les éléments positifs ou nuls se trouvent entre **1** et **j** dans l'ordre initial et les éléments négatifs entre **j+1** et **i-1**, dans l'ordre initial.
Si l'élément courant est positif ou nul, il est placé en position **j**, après libération de cette position par décalage à droite des éléments négatifs.




```
Réarranger(T,L) :
    x : entier ; j : entier sur [0…Lmax]                    {pour le décalage}
    j ← 0
    pour i allant de 1 à L
        si T_i ≥ 0 alors
            x ← T_i ;
            pour k allant de i-1 a j+1, pas -1
                T_{k+1} ← T_k
            j ← j+1
            T_j ←x
```

(ii) Représentation chaînée, copie des éléments négatifs [3 points]

Dans un *parcours* de la liste donnée, le résultat est construit par *ajouts successifs en queue* (ordre pertinent). La liste résultat peut être initialisée hors de l'itération (recherche du premier négatif), ou dans l'itération en la munissant d'un *élément fictif de tête* temporaire. C'est ce qui est fait ici :

```
CopierNeg(TS, TN)
    F, Q : dCelE                {élément fictif de tête du résultat et pointeur de queue}
    AC : adCelE                 {adresse courante du parcours}
    X : adCelE                  {pour la création des cellules du résultat}
    Allouer(F) ; Q ←F
    AC ← TS
    tant que AC ≠ Nil
        si AC↑.E < 0 alors
            Allouer(X) ; X↑.E ← AC↑.E ; Q↑.Suc ← X ; Q ← X
        AC ← AC↑.Suc
    Q↑.Suc ← Nil
    TN ← F↑.Suc
    Libérer(F)
```

Dans une autre version, sans fictif, il faudrait distinguer le cas de création du premier élément, et le cas des ajouts en queue des suivants.

(iii) Représentation chaînée, réarrangement en deux listes [3 points]
Version 1 : on réarrange la fin de la liste donnée, puis on place le premier élément dans la liste adéquate, en tête (ce qui conserve l'ordre initial).

```
Réarranger(TS, TP, TN) :
    si TS = Nil alors
        TP ← Nil
        TN ← Nil
    sinon   Réarranger(TS↑.Suc, TP, TN)
            si TS↑.E ≥ 0 alors
                    TS↑.Suc ← TP
                    TP ← TS
            sinon
                    TS↑.Suc ← TN
                    TP ← TN
            TS ← Nil
```

Version 2 : on place le premier élément dans la bonne liste selon son signe, puis on réarrange la fin de la liste donnée.

```
Réarranger(TS, TP, TN) :
    si TS = Nil alors
        TP ← Nil
        TN ← Nil
    sinon
            si TS↑.E ≥ 0 alors
               TP ← TS
               TS ← TS↑.Suc
               Réarranger(TS, TP↑.Suc, TN)
            sinon
               TN ← TS
               TS ← TS↑.Suc
               Réarranger(TS, TP, TN↑.Suc)
```

3. Les *traversées paires* d'un arbre binaire [7 points]

Q3 Une traversée paire [2 points]
Une traversée paire est obtenue en plaçant r en tête d'une traversée paire de G s'il en existe ou de D s'il en existe.

```
(1) UneTP(/ \) = [ ]
(2) UneTP(/G, r, D\) =  si non Pair(r) alors[ ]
                        sinon si EstVide?(G) et EstVide?(D) alors [r]
                              sinon   soit TG = UneTP(G)
                                      dans  si non EstVide?(TG) alors r∘TG
                                            sinon   soit TD = UneTP(D)
                                                    dans si EstVide?(TD) alors [ ] sinon r∘TD
```

Q4 Les traversées paires [2 points]
Les traversées paires de A sont obtenues en concaténant les traversées paires de G et les traversées paires de D et en plaçant r en tête de toutes ces traversées.

```
(1) LesTP(/ \) = [ ]
(2) LesTP(/G, r, D\) =  si non Pair(r) alors[ ]
                        sinon si EstVide?(G) et EstVide?(D) alors [ [r] ]
                              sinon   soit L = LesTP(G) & LesTP(D)
                                      dans    PlusT(r, L)
```

(1) PlusT(x, [ ]) = [ ]
(2) PlusT(x, s∘SS) = (x ∘ s) ∘ PlusT(x, SS)

Q5 Copie d'une traversée paire [3 points]

```
CopierUneTP(X, Y) :
    Z : adCel
    si X = Nil ou alors non Pair(X↑.R) alors Y ← Nil
    sinon   si X↑.G = Nil et X↑.D = Nil alors
                Allouer(Y)
                Y↑ ← <X↑.R, Nil>                          {création singleton}
            sinon CopierUneTP(X↑.G, Z)
                si Z ≠ Nil alors
                    Allouer(Y)
                    Y↑ ← <X↑.R ; Z>                       {ajout en tête}
                sinon CopierUneTP(X↑.R, Z)
                    si Z = Nil alors Y ← Nil
                    sinon
                       Allouer(Y)
                       Y↑ ← <X↑.R ; Z>                    {ajout en tête}
```