# M2 CCI – Algorithmique – Devoir surveillé 2

**Durée 2h, sans documents** **5 décembre 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. Répartition d'une liste en deux listes

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

**Q1 [8 points]**

**(i) Représentation contiguë**

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

MaxE : constante de type entier > 0 *{longueur maximum des séquences d'entiers}*
Réarranger : action (donnée-résultat T : tableau sur [1…MaxE] d'entiers,
donnée L : entier sur [0…MaxE])
*{Réarrange la séquence donnée par T et L 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.}*
Exemple :
si à l'état initial la séquence est représentée par [-2, -5, 3, -1, 4, 25, 0, -3, 4] de longueur 9,
alors à l'état final, la séquence est représentée par [3, 4, 25, 0, 4, -2, -5, -1, -3] de longueur 9.

— Donner une **réalisation itérative** de l'action **Réarranger**, sans utiliser d'autre tableau que le tableau **T**.

**(ii) Représentation chaînée, copie des éléments négatifs**

Les séquences sont maintenant 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**.

**(iii) Représentation chaînée, réarrangement en deux listes**

— Donner une **réalisation itérative** de l'action :

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.}*

## 2. Partition d'un ensemble de mots

On reprend l'exemple vu en cours : on considère la *partition* **P** d'un ensemble **E** de mots, selon l'initiale (la première lettre) de ces mots. **P** est un ensemble de *classes* **non vides** ; chaque classe de **P** est caractérisée par une lettre de l'alphabet : c'est le sous-ensemble de **E** de tous les mots ayant cette lettre pour initiale. Si une lettre n'est initiale d'aucun mot de **E**, il n'y a pas de classe associée à cette lettre dans **P**.

Les ensembles de mots sont représentés par des séquences de mots sans répétitions. Une partition est une séquence de classes, chaque classe étant un ensemble de mots.

### Q2 [5 points] Représentation contiguë

Les séquences sont sous **forme contiguë** dans des tableaux avec longueur explicite. **Aucun ordre n'est imposé sur ces séquences**. On utilise le lexique suivant :

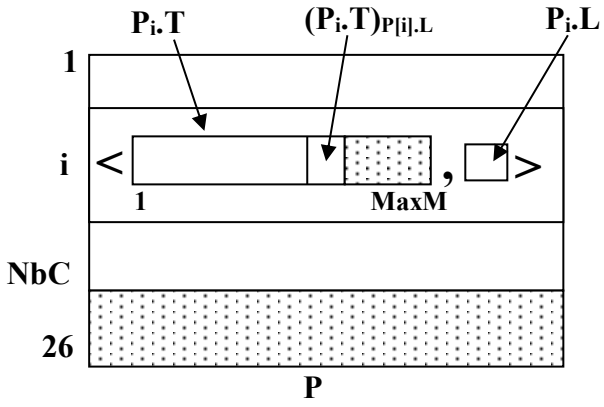
*{Les mots : **on fait abstraction de leur représentation**}*
Lettre : type caractère *{restreint aux lettres de l'alphabet}*
Mot : type séquence non vide de Lettre
*{M étant de type Mot, **premier(M)** dénote le 1er caractère de M. Pour affecter une valeur de type Mot à une variable de type Mot, on utilise le **symbole d'affectation usuel (←)**.}*

*{Les séquences de mots}*
MaxM : constante de type entier > 0 *{longueur maximum des séquences de mots}*
SeqMot : type <T : tableau sur [1…MaxM] de Mot ; L : entier sur [0…MaxM]>
*{ SM étant de type SeqMot, les mots de la séquence représentée par SM sont placés dans le tableau SM.T entre les indices 1 et SM.L. }*

*{L'ensemble E de tous les mots}*
E : SeqMot

*{La partition P de E : séquence de NbC classes (séquences de mots) non vides sous forme contiguë dans un tableau **P**}*
P : tableau sur [1…26] de SeqMot non vide
NbC : entier sur [0…26]

La figure ci-contre illustre la représentation de la partition : la classe située à l'indice **i** du tableau **P** est une séquence de $\mathbf{P_i.L}$ mots implantés de manière contiguë dans le tableau $\mathbf{P_i.T}$ et ayant tous la même initiale, de valeur $\mathbf{premier((P_i.T)_1)}$ (première initiale du premier mot de la $i^{ème}$ classe).




**(i) Construction de E à partir de P**

— Donner la **réalisation itérative** d'un algorithme qui construit l'ensemble **E** associé à une partition **P** donnée :

- Donnée fournie : le tableau **P** représentant la partition et sa longueur explicite **NbC**.
- Résultat à calculer : la variable **E** de type **SeqMot**

**(ii) Construction de P à partir de E**

— Donner la **réalisation itérative** d'un algorithme qui construit la partition **P** associée à un ensemble **E** donné :

- Donnée fournie : la variable **E** de type **SeqMot**.
- Résultats à calculer : le tableau **P** représentant la partition et sa longueur explicite **NbC**.

**Q3 [7 points] Représentation chaînée**

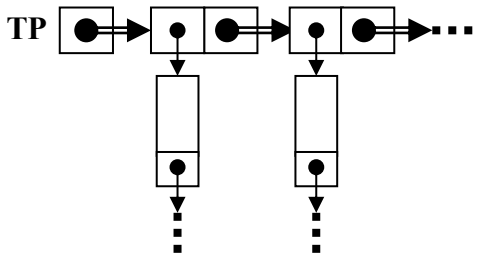
Les séquences sont maintenant sous forme de **listes chaînées** (représentation standard). **Aucun ordre n'est imposé sur ces listes**. **P** est une liste chaînée dont les éléments sont les têtes des listes chaînées représentant les classes. On utilise le lexique suivant :

*{cellules de mots}*
adCelM : type pointeur de CelM
CelM : type <M : Mot, Msuiv : adCelM>

*{cellules de classes}*
adCelC : type pointeur de CelC
CelC : le type <C : adCelM, Csuiv : adCelC>

*{accès à la partition P}*
TP : adCelC *{adresse de tête de la liste représentant P.}*

La figure ci-contre illustre la représentation de la partition **:** une flèche simple représente un pointeur de type **adCelM** et une flèche double représente un pointeur de type **adCelC**. **TP** contient l'adresse de tête de la séquence de classes. **TP↑.C** contient l'adresse de tête de la première classe de la partition ; **(TP↑.C)↑.M** contient le premier mot de la première classe.




**(i) Nombre de mots**

— Donner la **réalisation itérative** d'un algorithme qui affiche le nombre de mots d'une partition donnée :

- Donnée fournie : adresse de tête **TP** d'une liste de classes représentant une partition **P**.
- Résultat à calculer : nombre de mots de la partition **P**.

**(ii) Suppression d'un mot**

— Donner la **réalisation itérative** d'un algorithme qui supprime un mot donné dans une partition **P** donnée :

- Donnée fournie : une variable **M** de type **Mot**.
- Donnée-résultat fournie et à modifier : adresse de tête **TP** d'une liste de classes représentant une partition **P**.

Si le mot n'appartient pas à **P**, **TP** est inchangée. Si le mot **M** est le seul mot de sa classe, cette classe est supprimée. À l'état final, toutes les cellules non utilisées (de type **CelM** ou **CelC**) ont été libérées.

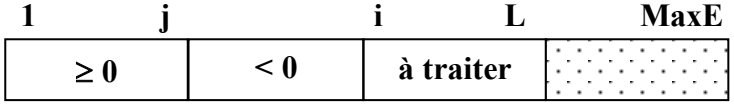
# M2 CCI – Algorithmique

**AL – DS 2 : des exemples de solutions** **5 décembre 2024**

## 1. Répartition d'une liste en deux listes

### Q1 (i) Représentation contiguë [3 points]

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…MaxE]                    {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
```

### Q1 (ii) Représentation chaînée, copie des éléments négatifs [2 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.

```
CopierNeg(TS, TN) :
    F, Q : adCelE                {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)
```

### Q1 (iii) Représentation chaînée, réarrangement en deux listes [3 points]

Même idée mais avec deux fictifs pour chacune des listes résultats.

```
Réarranger (TS, TP, TN) :
    FP, QP : adCelE              {élément fictif de tête et pointeur de queue pour TP}
    FN, QN : adCelE              {élément fictif de tête et pointeur de queue pour TN}
    ACS : adCelE                                         {adresse courante du parcours}
```

```
Allouer(FP) ; FP↑.Suc ← Nil ; QP ← FP
Allouer(FN) ; FN↑.Suc ← Nil ; QN ← FN
tant que TS ≠ Nil
        ACS ← TS
        TS ← TS↑.Suc
        si ACS↑.E ≥ 0 alors
                QP↑.Suc ← ACS ; QP ← ACS
        sinon
                QN↑.Suc ← ACS ; QN ← ACS
        ACS↑.Suc ← Nil
TP ← FP↑.Suc
TN ← FN↑.Suc
Libérer(FP)
Libérer(FN)
```

## 2. Partition d'un ensemble de mots

**Q2 (i) Représentation contiguë, construction de E pour P donnée [2 points]**

*{Chaque classe de P (parcours de P) est recopiée dans E (parcours d'une classe) en procédant par ajout en queue de E.}*

```
{pré-condition : Σ_{i=1 à P.L} P_i.L ≤ MaxM}
E.L ← 0
pour i allant de 1 à NbC
        pour j allant de 1 à P_i.L
                E.L ← E.L + 1
                (E.T)_{E.L} ← (P_i.T)_j
```

**Q2 (ii) Représentation contiguë, construction de P pour E donné [3 points]**

*{Aucun ordre n'est imposé (entre les classes, dans une classe). Les classes de P sont non vides.}*

*{Pour chaque mot de E (parcours), l'insérer dans P. Pour cela, soit I l'initiale du mot courant : s'il existe dans P (recherche) une classe dont les mots commencent par I, ajouter le mot courant en queue de cette classe ; sinon créer une nouvelle classe avec ce mot (ajout en queue dans P).}*

```
MC : Mot                        {mot courant dans le parcours de E}
j : entier sur [1…27]           {pour la recherche dans P}

NbC ← 0
pour i allant de 1 à E.L
        MC ← (E.T)_i
        j ← 1
        tant que j≠ NbC + 1 et puis premier((P_j.T)_1) ≠ premier(MC)
           j ← j+1
        si j ≠ NbC + 1 alors                    {ajout en queue de la classe j}
                P_j.L ← P_j.L + 1
                (P_j.T)_{P[j].L} ← MC
        sinon                                   {ajout en queue de P d'une classe singleton}
                NbC ← NbC + 1
                P_{NbC}.L ← 1
                (P_{NbC}.T)_1 ← MC
```

**Q3 (i). Nombre de mots de P [2 points]**

Pour chaque classe de **P**, compter son nombre de mots (deux *parcours emboîtés*).

```
ACC : adCelC                {adresse de la classe courante}
AMC : adCelM                {adresse du mot courant}
NbM : entier ≥ 0            {nombre de mots}
NbM ← 0

ACC ← TP
tant que ACC ≠ Nil
    AMC ← ACC↑.C
    tant que AMC ≠ Nil
        NbM ← NbM + 1
        AMC ← AMC↑.Msuiv
    ACC ← ACC↑.Msuiv
```

**Q3 (ii). Suppression d'un mot [5 points]**

On cherche s'il existe dans **P** (*recherche*) une classe correspondant à l'initiale du mot cherché. Si c'est le cas, on cherche le mot dans cette classe. S'il est présent, on le supprime et si la classe devient vide, elle est supprimée. On prend en compte les cas de suppression en tête.

```
{données : X de type Mot, donnée-résultat :  TP tête de liste de la partition}
AMC, AMP : adCelM           {adresses du mot courant et du mot précédent}
ACC, ACP : adCelC           {adresses de la classe courante et de la classe précédente}

{recherche de la classe de l'élément courant}
ACC ← TP
tant que ACC ≠ Nil et puis premier(X) ≠ premier((ACC↑.C)↑.M)
    ACP ← ACC
    ACC ← ACC↑.Csuiv
si ACC ≠ Nil alors {recherche de M dans la classe courante}
    AMC ← ACC↑.C
    tant que AMC ≠ Nil et puis M ≠ AMC↑.M
        AMP ← AMC
        AMC ← AMC↑.Msuiv
    si AMC ≠ Nil alors {supprimer le mot d'adresse AMC}
        si AMC = ACC↑.C alors {suppression en tête de classe}
            ACC↑.C ← AMC↑.Msuiv
            si ACC↑.C = Nil alors {suppression de la classe d'adresse ACC}
                si ACC = TP alors {suppression première classe}
                    TP ← ACC↑.Csuiv
                sinon
                    ACP↑.Csuiv ← ACC↑.Csuiv
                Libérer(ACC)
        sinon
            AMP↑.Suc ← AMC↑.Msuiv
        Libérer(AMC)
```