Université de Mons
FS/1/5684 – Algorithmique
Examen de première session – Partie pratique

Le 22 janvier 2010

Consignes

- Pour cette partie, vous avez le droit de consulter vos notes et tout ouvrage qui vous semble utile.
- Cette partie de l'examen dure 1 heure 45 minutes.
- Veillez à bien justifier vos réponses. Une réponse mal justifiée, même correcte, ne permet pas d'obtenir le maximum des points.
- Quand vous indiquez une complexité, veillez à bien expliquer ce que sont les paramètres qui apparaissent dans le $\mathcal{O}$. Par exemple, $\mathcal{O}(n^2)$ n'a aucun sens si n n'apparaît pas dans l'algorithme ou dans la définition de la structure qui est traitée…

Question 1 – 4 points

On considère la fonction suivante, qui a pour but de vérifier qu'un vecteur V est trié par ordre croissant :

début
  Booléen trié := *vrai* ;
  Entier $i := 2$;
  tant que $i \leq n$ *et trié* $=$ *vrai* **faire**
    si $V[i] < V[i-1]$ **alors** trié := *faux* ;
    $i := i+1$;
fin

Donc, si cette fonction est correcte, l'assertion suivante doit être vraie à la fin de la boucle :

$$\text{trié} = \textit{vrai} \quad \text{ssi} \quad \forall 2 \leq j \leq n : V[j] \geq V[j-1] \tag{1}$$

1. Parmi les assertions suivantes, lesquelles constituent un invariant de la boucle ci-dessus ? Justifiez vos réponses (pour les assertions qui sont un invariant, on ne vous demande pas de donner une preuve formelle mais *d'expliquer* pourquoi il s'agit d'un invariant).
   (a) $i \leq n+1$ et $\Big(\text{trié} = \textit{vrai}$ ssi $\forall 2 \leq j \leq i-1 : V[j] \geq V[j-1]\Big)$
   (b) $i < n+1$ et $\Big(\text{trié} = \textit{vrai}$ ssi $\forall 2 \leq j \leq i-1 : V[j] \geq V[j-1]\Big)$
   (c) trié $= \textit{vrai}$ ssi $\forall 2 \leq j \leq i : V[j] \geq V[j-1]$
   (d) $1 \leq i \leq n+1$ et $\Big(\text{trié} = \textit{vrai}$ ssi $\forall 1 \leq j < i-1 : V[j] \leq V[j+1]\Big)$
2. Parmi les assertions qui constituent un invariant lesquelles permettent de prouver que (1) est bien correcte à la fin de la boucle ? Donnez la preuve que (1) est correcte pour *un des invariants* qui le permettent.

Rappel important : nous avons utilisé la convention du cours de numéroter les cases à partir de 1.

Correction

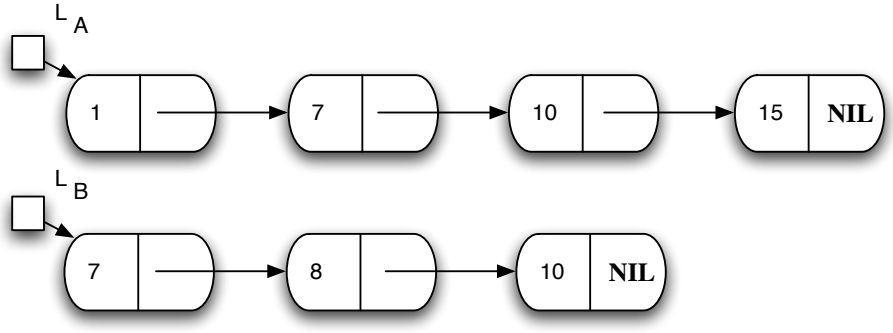
Seuls le (a) et le (d) sont des invariants. D'abord, remarquons qu'on a $i \leq n+1$ et non pas $i \leq n$, car, si le vecteur est trié, l'indice i atteindra $n+1$ lors du dernier tour de boucle. Cela disqualifie donc immédiatement (b). Ensuite, à chaque tour de boucle, on vérifie que $V[i] \geq V[i-1]$. Si ce n'est pas le cas, trié devient faux. Dans tous les cas, on incrémente i. Donc, à chaque début d'itération de la boucle, on a la garantie que la portion du vecteur qui va jusqu'à $i-1$ est triée : pour tout j compris entre 2 (valeur initiale de i) et $i-1$: $V[j] \geq V[j-1]$. C'est exactement l'assertion (a). L'assertion (d) affirme que $1 \leq i \leq n+1$, ce qui est toujours vrai (malgré le fait que i ne prendra jamais la valeur 1, mais cela ne rend pas l'invariant faux pour autant puisque $i \geq 2$ implique que $i \geq 1$). Ensuite, (d) donne une propriété sur le vecteur qui est la même que celle donnée par (a), mais écrite de manière différente. (d) est donc un invariant. Finalement, (c) n'est pas invariant, car il prétend que la portion *jusqu'à i inclus* est triée quand trié est vrai, ce qui est faux, étant donné qu'on incrémente i à la fin de la boucle. Par exemple, si V comprend deux cases $V[1] = 2$ et $V[2] = 1$ (et n'est donc pas trié), l'invariant est faux initialement. En effet, $i = 2$ initialement, trié vaut *vrai* initialement, et l'invariant affirme donc que $V[2] \geq V[1]$.
Pour faire les preuves demandées, il suffit de faire la conjonction entre la négation de la condition de la boucle, à savoir « $i \geq n+1$ ou trié $=$ *faux* », et l'invariant ; et de constater que le résultat implique (1). Ceci est possible tant avec (a) qu'avec (d).

Question 2 – 6 points

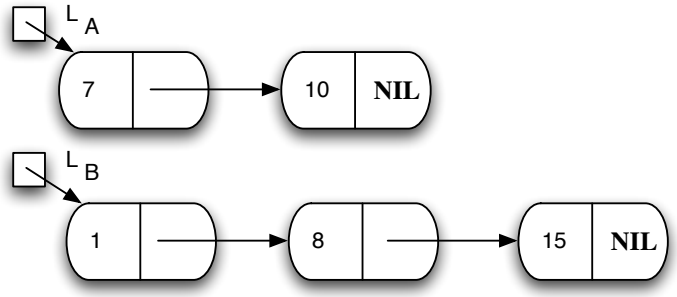
Dans cette question, on représentera des *ensembles de nombres naturels* à l'aide de *listes simplement liées et triées par ordre croissant*, contenant tous les éléments de l'ensemble à représenter. Comme il s'agit de représenter des ensembles, les listes seront *sans répétition*. On vous demande *deux choses* :

1. Écrire un algorithme qui reçoit deux telles listes L_A et L_B, qui représentent A et B, et qui modifie L_A et L_B de telle manière que L_A ne contienne que les éléments de $A \cap B$ et L_B ne contienne que les éléments de $(A \cup B) \setminus (A \cap B)$ (la différence symétrique de A et B). Autrement dit, après l'appel à votre algorithme, la liste L_A ne devra contenir que les éléments qui sont présents *dans les deux listes données*, et L_B ne devra contenir que les éléments qui ne sont présents *que dans une seule de deux listes données*.Votre solution devrait idéalement exploiter le fait que les listes sont triées.
2. Donner la complexité, en terme de $\mathcal{O}$, et en fonction des tailles des listes L_A et L_B, de votre solution.

Exemple Voici deux listes L_A et L_B qui représentent respectivement $A = \{1,7,10,15\}$ et $B = \{7,8,10\}$:



Après l'appel à l'algorithme, les listes devront être modifiées comme suit :



On voit bien que L_A contient alors $A \cap B = \{7,10\}$ et L_B contient $(A \cup B) \setminus (A \cap B) = \{1,7,8,10,15\} \setminus \{7,10\} = \{1,8,15\}$.

Correction

Une solution immédiate consiste à parcourir la liste L_A à l'aide d'un pointeur p_A et à chercher chaque information p_A.info dans L_B. Si l'information est trouvée, on supprime l'élément correspondant dans L_B et on avance le pointeur dans L_A. Sinon, on supprime l'élément pointé par p_A de L_A et on l'insère de manière triée dans L_B. Ces opérations peuvent être réalisées à l'aide de fonctions étudiées au cours. Malheureusement, ce faisant, on n'exploite pas le fait que les listes sont triées, et, comme il est nécessaire de parcourir L_B pour chaque élément de L_A, on obtient une complexité en $\mathcal{O}(\ell_A \times \ell_B)$, où ℓ_A et ℓ_B sont respectivement les tailles de L_A et L_B.

Afin d'obtenir une solution plus efficace, il faudrait ne pas avoir à reparcourir les listes. Cela peut se faire en exploitant le fait que les listes sont triées. Sur l'exemple, si on place un pointeur p_A sur le premier élément de L_A et un pointeur p_B sur le premier élément de L_B, on voit que $1 = p_A.\text{info} < p_B.\text{info} = 7$. Comme les listes sont *triées*, on a la garantie que p_A.info ne sera jamais dans L_B, et ce, sans devoir la parcourir. On peut donc déplacer l'élément pointé par p_A vers L_B. À nouveau, cette insertion peut se faire sans devoir parcourir L_B, en tenant compte du fait qu'elle est triée, car il suffit d'insérer *avant* p_B (pour ce faire il faut garder un pointeur q_B qui pointe toujours sur l'élément avant p_B – on fait de même avec un pointeur q_A dans L_A pour pouvoir faire les suppressions de manière aisée). On peut maintenant avancer le pointeur p_A, et on obtient $p_A.\text{info} = p_B.\text{info} = 7$. Dans ce cas, il suffit de supprimer de L_B l'élément pointé par p_B, ce qui se fait aussi en temps constant si on a le pointeur q_B. Comme on a traité les éléments tant dans L_A que dans L_B, on avance les deux pointeurs. On a alors $10 = p_A.\text{info} > p_B.\text{info} = 8$. Dans ce cas, on est dans une situation symétrique du premier cas : le 8 ne se trouvera jamais dans L_A, et on peut donc l'ignorer dans L_B, puisqu'il y est correctement placé. Il suffit donc d'avancer les pointeurs p_B et q_B dans L_B.

Ces opérations sont a répéter jusqu'à ce qu'une des deux listes soit complètement traitée. Attention, dans ce cas, il peut rester des éléments dans l'autre liste, et il faut en tenir compte. Si $p_B = \textbf{NIL}$ mais $p_A \neq \textbf{NIL}$, il faut déplacer les éléments restants de L_A vers L_B. Par contre, si $p_A = \textbf{NIL}$ et $p_B \neq \textbf{NIL}$, il n'y a plus rien à faire.

Dans les trois cas considérés ci-dessus, les opérations effectuées sont en $\mathcal{O}(1)$ et on a la garantie d'avancer au moins un des deux pointeurs à chaque fois. Au pire cas, il faudra donc répéter $\ell_A + \ell_B$ fois ces opérations pour arriver à la fin des deux listes. Comme on répète $\ell_A + \ell_B$ fois des opérations en $\mathcal{O}(1)$, on a une complexité en $\mathcal{O}(\ell_A + \ell_B)$.

La correction est donnée ci-dessous. À noter que dans cette implémentation, on a choisi de commencer par insérer des éléments pré-tête dans les deux listes pour ne pas avoir de problèmes liés à la manipulation de la tête. Ces éléments sont supprimés en fin de traitement, afin de respecter la structure des listes originelles.

Correction

```
begin
    Elem * pA, pB, qA, qB ;
    /* Initialisation des pointeurs et insertion des éléments pré-tête.     */
    pA := LA ;
    pB := LB ;
    qA := new Elem ;
    qA.next := LA ;
    LA := qA ;
    qB := new Elem ;
    qB.next := LB ;
    LB := qB ;

    /* Boucle principale.                                                   */
    tant que pA ≠ NIL et pB ≠ NIL faire
        si pA.info = pB.info alors
            /* Suppression de l'élément pointé par pB dans LB.              */
            qB.next := pB.next ;
            delete pB ;
            pB := qB.next ;
            /* On avance les pointeurs dans LA.                             */
            qA := pA ;
            pA := pA.next ;
        sinon si pA.info < pB.info alors
            /* Déplacement de l'élément pointé par pA de LA vers LB. L'élément
               sera inséré entre qB et pB.                                  */
            qA.next := pA.next ;
            pA.next := pB ;
            qB.next := pA ;
            /* On réajuste qB et pA suite au déplacement.                   */
            qB := qB.next ;
            pA = qA.next ;
        sinon
            /* On avance les pointeurs dans LB.                             */
            qB := pB ;
            pB := pB.next ;

    /* Traitement des derniers éléments de LA, s'il y a lieu.               */
    si pA ≠ NIL alors qB.next := pA ;

    /* Suppression des éléments pré-tête.                                   */
    pA := LA ;
    LA := LA.next ;
    delete pA ;
    pB := LB ;
    LB := LB.next ;
    delete pB ;
end
```