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

Le 24 août 2008

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 – 2 points

Donnez la complexité de la fonction ci-dessous :

```
F(Entier n, Entier m)
début
    Entier i, j ;
    si n > m alors Afficher n et m ;
    sinon
        i := 1 ;
        tant que i ≤ n faire
            j := 1 ;
            tant que j ≤ 3 faire
                Afficher i et j ;
                j := j + 1 ;
            i := i + 3 ;
fin
```

Correction

La fonction est en $\mathcal{O}(n)$. En effet, si le premier test évalue à vrai, la fonction est en $\mathcal{O}(1)$. Sinon, on rencontre deux boucles imbriquées. La plus imbriquée exécute un nombre constant d'opérations constantes et est donc en $\mathcal{O}(1)$. La plus extérieure s'exécute au plus $n/3+1$ fois, et est donc en $\mathcal{O}(n)$. Comme on considère le pire cas, on garde $\mathcal{O}(n)$.

Question 2 – 5 points

Une *delta-liste* est une liste simplement liée qui est utilisée pour stocker une séquence d'éléments ordonnés dans le temps. Ce type de liste est couramment utilisé dans les pilotes de périphériques ou les systèmes d'exploitation qui doivent gérer des listes d'événements.

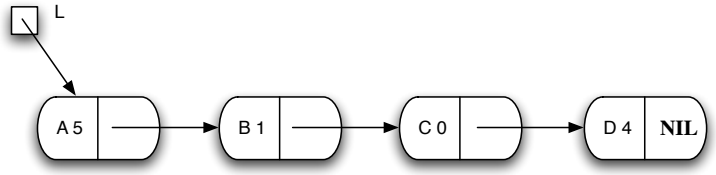
Nous considérerons ici des événements qui sont caractérisés par un nom (une lettre de l'alphabet) et un instant d'occurrence (entier). Par exemple, considérons la séquence d'événements suivants :

- L'événement `A` a lieu au temps 5
- L'événement `B` a lieu au temps 6
- L'événement `C` a lieu au temps 6 également
- L'événement `D` a lieu au temps 10

Pour constituer la delta-liste, on stocke cette séquence d'événements par ordre chronologique, dans une liste dont chaque élément contient, en plus du pointeur vers l'élément suivant :

1. un champ *nom* de type **Caractère** qui donne le nom de l'événement et
2. un champ *temps* de type **Entier** qui donne le temps qui sépare cet événement de son précédent.

Par exemple, la séquence ci-dessus peut être représentée par la delta-liste suivante (remarquez que l'ordre des événements `B` et `C` est indifférent puisqu'ils ont lieu au même moment) :

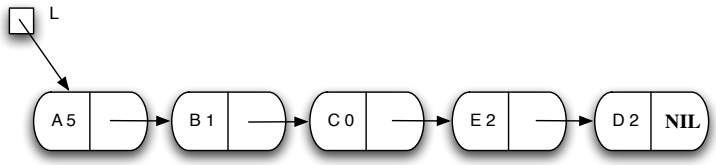


On vous demande d'écrire une fonction qui reçoit trois paramètres :

1. Un pointeur *L* vers la tête d'une delta-liste.
2. Un *nom* (de type **Caractère**) qui caractérise un événement.
3. Un temps *t* (de type **Entier**) qui indique à quel instant l'événement a lieu.

et qui insère, dans la delta-liste *L*, le nouvel événement indiqué. Après l'opération, la liste doit rester une delta-liste cohérente, et les instants auxquels ont lieu les autres événements ne doivent pas être modifiés.

Par exemple, si on insère l'événement `E` qui a lieu au temps 8 dans la delta-liste ci-dessus, on obtient :



Correction

L'observation cruciale dans cet exercice consiste à voir que la solution est une variation de l'*insertion triée* (voir syllabus, pp. 60 et 61). En effet, dans le cas de l'insertion triée étudiée au cours, on veut insérer une information (entière) de manière à ce que la séquence des informations soit croissante. Dans ce cas, on veut insérer un événement de manière ce que la séquence des instants d'occurrence des événements soit croissante. La différence principale tient dans le fait que les instants d'occurrence ne sont *pas* présents dans les éléments, et qu'il faut donc les recalculer.

```
Elem * InsertionTriée(Elem * L, Entier i)
début
    si EstVide(L) ou L.info ≥ i alors
        retourner InsèreTête(L, i) ;
    sinon
        Elem * q := L, p := L.next, r ;
        tant que p ≠ NIL et p.info < i faire
            q := p ;
            p := p.next ;
        r := new Elem ;
        r.next := p ;
        r.info := i ;
        q.next := r ;
        retourner L ;
fin
```

Repartons de l'algorithme de l'insertion triée. La principale modification à apporter est au niveau de la condition du **Tant que**, où $p.\text{info} < i$ n'est plus valide. On va donc maintenir une variable tq qui contiendra, à tout moment, l'instant d'occurrence de l'élément pointé par q, ce qui nous permettra aussi d'obtenir l'instant d'occurrence de l'élément pointé par p en calculant $tq + p.\text{temps}$. On obtient la valeur correcte de tq en y maintenant *somme de toutes les différences de temps* rencontrées dans la liste lors du parcours (plus précisément tous les élements avant q, celuci inclus). Sur l'exemple ci-dessus, on place initialement le pointeur q sur la tête et on initialise t à 5. En faisant avancer le pointeur q vers l'élément « `B` », on ajoute 1 à tq qui vaut 6, soit le temps auquel `B` a lieu. Ensuite, q pointe vers l'élément « `C` », et tq vaut $6+0=6$. A ce moment, l'expression $tq + p.\text{temps}$ vaut $6+4=10$ (en effet p pointe vers « `D` »), ce qui est > 8. On sort alors de la boucle, et on constate que q pointe effectivement sur l'élément représentant le dernier événement qui a lieu avant celui à insérer.

À la sortie de la boucle, on sait que l'on doit insérer le nouvel élément après q. On crée donc un nouvel élément r, que l'on insère entre q et p. Il reste deux questions à résoudre : quel est l'information que l'on va stocker dans $r.\text{temps}$? et comment va-t-on modifier l'information de p ?

L'information à stocker dans $r.\text{temps}$ est la distance temporelle qui sépare l'événénement représenté par r de son précédent, q. Or, l'événénement représenté par r a lieu à l'instant t (paramètre de la liste), et l'événénement représenté par q à l'instant tq (on l'a calculé). On stocke donc dans $r.\text{temps}$ l'information $t - tq$.

l'information à stocker dans $p.\text{temps}$ est la distance temporelle qui sépare l'événement représenté par p de son précédent, représenté par r. L'événement représenté par p a lieu à l'instant $tq + p.\text{temps}$ et celui représenté par r, à l'instant t. On stocke donc $tq + p.\text{temps} - t$ dans $p.\text{temps}$. Attention, ceci n'est à faire que si $p \neq \textbf{NIL}$! En effet, si le nouvel événement a insérer dans la liste a lieu après tous les autres, q pointera vers le dernier élément de la liste et on insérera en fin.

La solution est présentée sur la page suivante.

Correction

```
Nœud * InsertionDelta(Nœud * L, Caractère n, Entier t)
début
    si EstVide(L) ou L.temps ≥ t alors
        retourner InsèreTête(L, t) ;
    sinon
        tq := L.temps ;
        Elem * q := L, p := L.next, r ;
        tant que p ≠ NIL et tq + p.temps < i faire
            q := p ;
            tq := tq + p.temps ;
            p := p.next ;
        r := new Elem ;
        r.nom := n ;
        r.next := p ;
        r.temps := t − tq ;
        q.next := r ;
        si p ≠ NIL alors  p.temps = tq + p.temps − t ;
        retourner L ;
fin
```

Question 3 – 3 points

Donnez un algorithme récursif qui reçoit un arbre binaire de recherche A, ainsi que deux bornes bi et bs, et qui affiche toutes les valeurs contenues dans A qui sont comprises dans l'intervalle $[bi, bs]$ (bornes comprises, donc). Votre algorithme devra afficher ces valeurs dans l'ordre croissant.

Correction

Pour afficher les valeurs d'un arbre binaire de recherche dans l'ordre naturel (croissant), on utilise le parcours infixe, à savoir :

```
Infixe(Nœud *A)
début
    si A ≠ NIL alors
        Infixe(A.fg) ;
        Afficher A.info ;
        Infixe(A.fd) ;
fin
```

Pour n'afficher que les informations contenues entre bi et bs, il suffit d'ajouter un test autour de l'affichage :

```
InfixeBiBs(Nœud *A, Entier bi, Entier bs)
début
    si A ≠ NIL alors
        InfixeBiBs(A.fg, bi, bs) ;
        si bi ≤ A.info ≤ bs alors Afficher A.info ;
        InfixeBiBs(A.fd, bi, bs) ;
fin
```

On peut encore améliorer cet algorithme en évitant des appels récursifs qui ne sont pas nécessaires. Si l'information stockée à la racine d'un arbre est $< bi$, on sait que ce sera également le cas de toutes les informations stockées à gauche de la racine, et il n'est donc pas nécessaire de parcourir le sous-arbre gauche. De même, on ne parcourra pas le sous-arbre droit d'un arbre contenant une information $> bs$ dans la racine :

```
InfixeBiBs2(Nœud *A, Entier bi, Entier bs)
début
    si A ≠ NIL alors
        si A.info ≥ bi alors InfixeBiBs2(A.fg, bi, bs) ;
        si bi ≤ A.info ≤ bs alors Afficher A.info ;
        si A.info ≤ bs alors InfixeBiBs2(A.fd, bi, bs) ;
fin
```
