

# ESISAR MAN EAI 2005

## Contrôle continu (CS318) - CORRECTIONS

I. M. BILASCO, 20 novembre 2005

### Exercice 1 (30 min)

On considère deux tableaux A et B de n, respectivement m éléments ( $0 \leq m, n \leq 1000$ ). On dit que A est contenu dans B si tous les éléments de A figurent au moins une fois dans B.

A: (1, 2, 3)      B: (1, 1, 2, 3, 4)      A est contenu dans B.

A: (1, 1, 2, 2, 3)      B: (1, 2, 3)      A est contenu dans B. (même si dans A le 1 se trouve en double du fait qu'il apparaît au moins une fois dans B ... A est contenu dans B)

A: (1, 2, 3)      B: (1, 2, 2, 4, 5)      A n'est pas contenu dans B (il manque le 3)

Ecrire une fonction qui décide si A est contenu dans B. Pour optimiser la recherche des éléments on trie l'un des deux tableaux. Lequel des deux tableaux il vaut mieux trier? Donner aussi l'algorithme de tri utilisé.

```
FONCTION aDansB(nbA:entier;a:tableau d'entiers; nbB:entier; b:tableau d'entiers):booleen;  
{ renvoie vrai si tout element de a[1..nbA] appartient au b[1..nbB]}
```

```
VAR
```

```
    i:entier;
```

```
DEBUT
```

```
    trier(nbB,B); //ce qui nous permettra de faire des recherches  
                // dichotomiques pour chaque element de A
```

```
    i<-1;
```

```
    Faire
```

```
        pos<-positionDeXDansTab(a[i],1,nbB,b);
```

```
        i<-i+1;
```

```
    Tant que (pos != -1) && (i<=nbA)
```

```
FIN
```

```
FONCTION positionDeXDansTab(x:entier;li,lf:entier;a:tableau d'entiers):entier;
```

```
{ soit  $li \leq lf$ , a[li..lf]-trié. la fonction renvoie i si existe i tq:  $li \leq i \leq lf$  ai!=x, -1 sinon }
```

```
VAR
```

```
    pivot:entier;
DEBUT
    Faire
        pivot<-(li+lf) DIV 2;
        Si (a[pivot]<x) alors li<-pivot+1;
        Sinon lf<-pivot;
        FinSi
    Tant que (li<lf);
    Si (a[lf]=x) alors
        Renvoyer lf;
    Sinon
        Renvoyer -1;
    FinSi
FIN
```

## Exercice 2 (20 min)

Ecrire un algorithme de recherche *trichotomique* d'un élément  $X$  dans un vecteur trié contenant  $n$  éléments, sur le principe suivant:

*Comparer  $X$  avec l'élément à la position  $n/3$  puis le comparer éventuellement avec l'élément en position  $2*n/3$ ; si l'on n'a pas trouvé  $X$ , la recherche se poursuit alors sur une liste qui contient trois fois moins d'éléments que la liste d'origine.*

Construire des fonctions de recherche *trichotomique* telles que:

- dans la première on applique l'algorithme de manière récursive
- dans la deuxième on donne la version itérative.

FONCTION trichotomique(x:entier;li, lf:entier;a:tableau d'entier):entier;

{ soit  $li \leq lf$ ,  $a[li..lf]$  trié. renvoie  $i$  si  $li \leq i \leq lf$  et  $a_i = x$  sinon  $-1$  }

VAR

    pivot1, pivot2 : entier;

DEBUT

    Faire

        pivot1  $\leftarrow li + (lf - li) \text{ DIV } 3$ ;

        pivot2  $\leftarrow lf - (lf - li) \text{ DIV } 3$ ;

        si  $(a[\text{pivot1}] < x)$  alors

            si  $(a[\text{pivot2}] < x)$  alors

$li \leftarrow \text{pivot2} + 1$ ;

            sinon

$li \leftarrow \text{pivot1} + 1$ ;

$lf \leftarrow \text{pivot2}$ ;

            finsi

        sinon

$lf \leftarrow \text{pivot1}$ ;

        fin si

    Tant que  $li < lf$

    si  $a[lf] = x$  alors

        renvoyer  $lf$ ;

    sinon

        renvoyer  $-1$ ;

    finsi

FIN

FONCTION trichotomiqueR(x:entier;li, lf:entier;a:tableau d'entier):entier;

```

{ soit  $li \leq lf$ ,  $a[li..lf]$  trié. renvoie  $i$  si  $li \leq i \leq lf$  et  $a_i = x$  sinon  $-1$  }
VAR
    pivot1, pivot2 : entier;
DEBUT
    si ( $li < lf$ ) alors
        pivot1  $\leftarrow -li + (lf - li) \text{ DIV } 3$ ;
        pivot2  $\leftarrow -lf - (lf - li) \text{ DIV } 3$ ;
        si ( $a[pivot1] < m$ ) alors
            si ( $a[pivot2] < m$ ) alors
                renvoyer trichotomiqueR( $x, pivot2 + 1, lf, a$ );
            sinon
                renvoyer trichotomiqueR( $x, pivot1 + 1, pivot2, a$ );
            finsi
        sinon
            renvoyer trichotomiqueR( $x, li, pivot1, a$ );
        fin si
    si ( $li = lf$ ) alors
        si  $a[lf] = x$  alors
            renvoyer  $lf$ ;
        finsi;
    fin si;
renvoyer  $-1$ ;
FIN

```

### Exercice 3 (20 min)

Soit L une liste chaînée. Ecrire des procédures telles que:

- dans la première on supprime toutes les occurrences d'un élément donné  $x$ .
- dans la deuxième, pour chaque élément de la liste, on ne laisse que sa première occurrence

### Exercice 4 (30 min)

Un ensemble peut être représenté sous la forme d'une liste chaînée sans doublons.

Ecrire des procédures telles que:

- dans la première on construit un ensemble à partir d'une liste chaînée.

procedure construireEnsemble([in]l:Liste;[out]ens:Liste);

{ rôle: construit un ensemble à partir d'une liste en éliminant les doublons

état d'entrée: l liste avec au moins un élément

état de sortie: ens un ensemble (une liste) qui contient tous les éléments de la liste l sans les doublons

}

- dans la deuxième on construit l'union des deux ensembles.

procedure unionEns([in]ens1, ens2:Liste;[out]ens:Liste);

{ rôle: calcule l'union des deux ensembles

état d'entrée: ens1, ens2 deux ensembles (listes sans doublons)

état de sortie: ens un ensemble (une liste) qui contient tous les éléments des ensembles ens1, ens2 sans doublons

}

- dans la troisième on construit l'intersection des deux ensembles.

procedure intersectionEns([in]ens1, ens2:Liste;[out]ens:Liste);

{ rôle: calcule l'intersection des deux ensembles

état d'entrée: ens1, ens2 deux ensembles (listes sans doublons)

état de sortie: ens un ensemble (une liste) qui contient les éléments communs de ens1 et ens2

}

CORRECTION commune pour les exercices 3 et 4

TYPE

Noeud=structure

val:entier;

suiv: pointeur vers Noeud;

fin structure;

Liste=pointeur vers Noeud;

```
PROCEDURE supprimer(x:entier;[in/out]L:Liste);  
{ rôle: elimine les éléments dont la valeur est égale à x  
état d'entrée: l - liste, x - entier  
état de sortie: l - liste sans x = (l1...li...) qq soit  $l_i.val \neq x$   
}
```

VAR

q:Liste;

DEBUT

q<-L;pq<-NULL;

tant que q!=NULL faire

si (q^.val=x) alors

si (pq=NULL) alors

q<-q^.suiv;

Liberer L; L<-q;

sinon

pq^.suiv<-q^.suiv;

Liberer q;

q <- pq^.suiv;

fin si

sinon

pq<-q;

q<-q^.suiv;

fin si

fin tant que

FIN

```
PROCEDURE supprimerDoublons([in/out]L:Liste);
```

```
{ rôle: supprime les doublons d'une liste
```

```
état d'entrée: l - liste
```

```
état de sortie: l - liste sans doublons = (l1...li...lj...) qq soit i et j  $l_i.val \neq l_j.val$ 
```

```
}
```

VAR

q:Liste;

DEBUT

q<-L;

tant que q!=NULL faire

supprimer(q^.val,q^.suiv);

```

        q<-q^.suiv;
    fin tant que;
FIN

```

```

PROCEDURE construireEnsemble([in]l:Liste; [out]ens:Liste);
{rôle: construit un ensemble à partir d'une liste en éliminant les doublons
état d'entrée: l liste
état de sortie: ens un ensemble (une liste) qui contient tous les éléments de la liste
l sans les doublons
}
DEBUT
    copierListe(l,ens);
    supprimerDoublons(ens);
FIN

```

```

PROCEDURE copierListe([in]src:Liste; [out]dest:Liste)
{rôle: cree une copie d'une liste
état d'entrée: src une liste, dest une liste vide
état de sortie: dest contient les mêmes éléments que src (et dans le même ordre)
}
VAR
    q,r:Liste;
DEBUT
    //creation d'une sentinelle;
    Allouer(dest);r<-dest;
    //copier la liste element par element
    L<-q;
    tant que q!=NULL alors
        Allouer(r^.suiv);
        r<-r^.suiv;
        r^.val<-q^.val;
        q<-q^.suiv;
    fin tant que;
    r^.suiv<-NULL;

    //effacer la sentinelle
    r<-dest; dest<-dest^.suiv; Libérer r;

FIN

```

```

PROCEDURE union([in]ens1,ens2:Liste:[out]ens:Liste)
{rôle: calcule l'union des deux ensembles
état d'entrée: ens1, ens2 deux ensembles (listes sans doublons) et ens liste vide (=NULL)
état de sortie: ens un ensemble (une liste) qui contient tous les éléments de ens1 et ens2 sans
doublons
}

```

VAR

q,r:Liste;

DEBUT

si (ens1=NULL) alors copierListe (ens2,ens);

sinon si (ens2=NULL) alors copierListe (ens1,ens);

sinon

copierListe(ens1,ens);

copierListe(ens2,r);

//connecter les deux listes ens et r. ainsi ens contient tous les elements

//a. aller au dernier element de ens

q<-ens;

tant que q^.suiv!=NULL

q<-q^.suiv;

fin tant que;

//b. relier la liste r au dernier element de ens => ens=(ens+r)

q^.suiv<-r;

//construire l union en supprimant les doublons

supprimerDoublons(ens);

fin si

FIN

```

PROCEDURE intersection([in]ens1,ens2:Liste:[out]ens:Liste)

```

```

{rôle: calcule l'intersection des deux ensembles

```

```

état d'entrée: ens1, ens2 deux ensembles (listes sans doublons) et ens liste vide (=NULL)

```

```

état de sortie: ens un ensemble (une liste) qui contient les éléments communs de ens1 et ens2

```

```

}

```

VAR

q,r:Liste;

DEBUT

si (ens1=NULL) ou (ens2=NULL) alors

ens=NULL;

```

sinon
    //creation sentinelle
    Allouer(r);ens<-r;

    q<-ens1;
    tant que q!=NULL faire
        //on ne copie que si l'element appartient aussi a ens2
        si contient(q^.val,ens2) alors
            allouer(r^.suiv);
            r<-r^.suiv; r^.val=q^.val;
        fin si
    fin tant que
    //on ferme la liste
    r^.suiv=NULL;

    //on supprime la sentinelle
    r<-dest; dest<-dest^.suiv; Liberer r;
fin si
FIN

FONCTION contient(x:entier;ens:Liste):booleen;
VAR
    q:Liste;
DEBUT
    q<-ens;
    tant que (q!=NULL) et (q^.val!=x) faire
        q<-q^.suiv;
    fin tant que
    renvoyer q!=NULL;
FIN

```