Corrigé TD2

Habiba Drias

Exercice 2.2

1) Ecrire la procédure d'insertion d'un élément dans un ensemble et calculer sa complexité.

```
Procédure insertion ;
entrée : TV : tableau d'entiers ; VR : tableau de booléens ; x : entier ;
sortie :  TV : tableau d'entiers ; VR : tableau de booléens ;

var p : entier ;
début       p :=1;
        tant que (p <= TV[max]) et (TV[p] ≠ x) faire p := p+1 ;
        si (TV[p] ≠ x) alors début TV[p] := x ;
                                   TV[max] := TV[max] +1 ;
                                fin ;
        VR[p] :=1 ;

fin
```

La complexité de la procédure insertion est *O*(*TV[max])* car au pire cas, *x* n'appartient pas à TV et la boucle aura un nombre d'itérations égal à *TV[max]*.

2) Ecrire la procédure de suppression d'un élément d'un ensemble et calculer sa complexité.

```
Procédure suppression ;
entrée : TV : tableau d'entiers ; VR : tableau de booléens ; x : entier ;
sortie :  TV : tableau d'entiers ; VR : tableau de booléens ;

var p : entier ;
début   p :=1;
        tant que (p <= TV[max]) et (TV[p] ≠ x) faire p := p+1 ;
        si (TV[p] = x) alors
                si (VR[p] =0) alors afficher ('x n'existe pas dans l'ensemble')
                                sinon VR[p] := 0
          sinon afficher ('x n'existe pas dans l'ensemble') ;
fin
```

La complexité de la procédure suppression est *O*(*TV[max])* car au pire cas, *x* n'appartient pas à TV et la boucle aura un nombre d'itérations égal à *TV[max]*.

Exercice 2.3

1) L'algorithme de fusion de deux listes triées

a. En utilisant la structure de tableau

Chacune des deux listes est représentée par deux tableaux : élément et suivant. Soient tête1 et tête2 les indices respectifs des premiers éléments des listes. L'algorithme de fusion est comme suit :

```
algorithme fusion ;
entrée : tête1, tête2 :entier ;
sortie : tête3 : entier ;
const max = 200 ;
var p1, p2, p3, libre : entier ;
    T1, T2, T3 : tableau [1..max] de
        enregistrement élément : entier ; suivant : 1..max fin;
début   p1 := tête1 ;
        p2 := tête2 ;
        libre := 1
        tant que (p1 ≠ -1) et (p2 ≠ -1) faire
        début   si (T1[p1].élément < T2[p2].élément) alors
                        début T3[libre].élément := T1[p1].élément ;
                                p1 := T1[p1].suivant ;
                        fin
                sinon début T3[libre].élément := T2[p2].élément ;
                                p2 := T2[p2].suivant ;
                        fin ;
                T3[libre].suivant := libre+1 ;
                Libre := libre+1 ;
        fin ;
        si (p1 = -1) alors
          tant que (p2 ≠ -1) faire
            début T3[libre].élément := T2[p2].élément ;
                T3[libre].suivant := libre+1 ;
                Libre := libre+1 ;
             fin  sinon
          tant que (p1 ≠ -1) faire
            début T3[libre].élément := T1[p1].élément ;
                T3[libre].suivant := libre +1;
                Libre := libre+1 ;
                fin ;
        si (libre = 1) alors tête3 := -1 sinon début T3[libre-1].suivant := -1 ;
                                                    tête3 := 1 ;
                                                fin ;
fin ;
```

b. En utilisant la structure dynamique de listes

```
algorithme fusion ;
entrée : tête1, tête2 : ↑p
sortie : tête3 : ↑p

var
        p : enregistrement élément : entier ; suivant : ↑p fin;
        p1, p2, p3, libre, prec : ↑p
début
        p1 := tête1 ;
        p2 := tête2 ;
        si (p1 ≠ nil) et (p2 ≠ nil) alors
                début nouveau(libre)
                        tête3 := libre
                fin sinon tête3 := nil ;
        tant que (p1 ≠ nil) et (p2≠ nil) faire
        début
                si (p1↑.élément < p2↑.élément) alors
                        début libre↑.élément := p1↑.élément ;
                                p1 := p1↑.suivant ;
                        fin
                sinon début libre↑.élément := p2↑.élément ;
                                p2 := p2↑.suivant ;
                        fin ;
                prec := libre ;
                nouveau(libre) ;
                prec↑.suivant := libre ;
        fin ;
        si (p1 = nil) alors
                tant que (p2 ≠ nil) faire
                début libre↑.élément := p2↑.élément ;
                        prec := libre ;
                        nouveau(libre) ;
                        prec↑.suivant := libre ;
                fin
        sinon   tant que (p1 ≠ nil) faire
                début libre↑.élément := p1↑.élément ;
                        prec := libre ;
                        nouveau(libre) ;
                        prec↑.suivant := libre ;
                fin ;
        libérer (libre) ;
        prec↑.suivant := nil ;
fin ;
```

2) L'algorithme parcourt les deux listes fournies en entrée linéairement pour construire une troisième liste qui contient les éléments des deux listes. Si *l1* et *l2* sont les longueurs respectives des deux listes, la complexité de l'algorithme serait *O(l1+l2)*.

3) Pour représenter une liste chainée en assembleur, il faut réserver un espace mémoire contigu pour contenir les éléments de la liste en sachant qu'un élément est un enregistrement d'un entier et d'une adresse relative. Plus précisément, deux mots contigus seront nécessaires pour mettre l'entier et l'adresse de l'élément qui suit.

Exemple :

Soit *l* = {11, 12, 45, 14, 76} une liste, sa représentation en assembleur est la suivante :

10	11
11	12
12	45
13	14
14	76
15	-1

4) Bien sûr, seule la représentation de tableau convient pour l'assembleur. L'algorithme de fusion en assembleur à l'aide du jeu d'instructions donné au chapitre 1 est comme suit :

```
        MOV     tête1 , R1
        MOV     tête2 , R2
        MOV     tête3, R3
            MOV   R3, R0          / libre := 1
            ADD   #1, R1          tant que (p1 ≠ -1) et (p2≠-1) faire
            JZ    R1, L0           début
            ADD   #1, R2             si (T1[p1].élément < T2[p2].élément) alors
            JZ    R2, L0               début T3[libre].élément := T1[p1].élément ;
            SUB   #1, R1                       p1 := T1[p1].suivant ;
            SUB   #1, R2                fin
            SUB   (R2), (R1)
            JGT   (R1), L1
            ADD   (R2), (R1)
            MOV   (R1), (R3)
            ADD   #1, R1
            MOV   (R1), R1
            JMP   L2
L1:         ADD   (R2), (R1)
            MOV   (R2), (R3)    sinon début T3[libre].élément := T2[p2].élément ;
            ADD   #1, R2                    p2 := T2[p2].suivant ;
            MOV   (R2), R2              fin ;
L2 :        ADD   #2, R0          T3[libre].suivant := libre+1 ;
            ADD   #1, R3                  Libre := libre+1 ;
            MOV   R0, (R3)        fin ;
            ADD   #1, R3
L0 :        JZ    (R1), L4
L6 :        JZ    (R2), L3
            JMP   L5              si (p1 = -1) alors
L3 :        SUB   #1, (R2)        tant que (p2 ≠ -1) faire
            MOV   (R2), (R3)       début T3[libre].élément := T2[p2].élément ;
            ADD   #1, R2                 T3[libre].suivant := libre+1 ;
            MOV   (R2), R2               libre := libre+1 ;
            ADD   #1, R3            fin  sinon
            ADD   #2, R0
            MOV   R0, (R3)
            ADD   #1, R3
            ADD   #1, (R2)
            JMP    L6
L4 :        SUB   #1, (R1)
            MOV   (R1), (R3)       tant que (p1 ≠ -1) faire
            ADD   #1, R1             début T3[libre].élément := T1[p1].élément ;
            MOV   (R1), R1                 T3[libre].suivant := libre +1;
            ADD   #1, R3                    libre := libre+1 ;
            ADD   #2, R0               fin ;
```

```
        MOV   R0, (R3)
        ADD   #1, R3
        ADD   #1, (R1)
        JMP   L0
L5 :    SUB   #1, (R1)
        SUB   #1, (R2)
        SUB   tête3, R0        si (libre = 1) alors tête3 := -1 sinon
        JZ    R0, L7           début
        SUB   #1, R3
        MOV   #-1, (R3)                T3[libre-1].suivant := -1 ;
        MOV   tête3, R3                tête3 := 1 ;
        JMP   Fin              fin ;
L7 :    MOV   #-1, R3
Fin :   STOP                 fin ;
tête1 : 31
        tête1+2
        65
        tête1+4
        121
        -1
tête2 : 7
        tête2+2
        17
        tête2+ 4
        53
        tête2+6
        329
        -1
tête3 :
```

On suppose que les listes fournies en entrée apparaissent à la fin du programme respectivement aux adresses tête1 et tête2. A la fin de l'exécution de ce programme, tête3 contient le début de la liste résultat de la fusion des deux listes données en entrée.

5) La complexité de ce programme est identique à celle de l'algorithme de 1) à une constante multiplicative près. Elle est donc en *O(l1+l2)*.

Exercice 2.6

```
type liste : enregistrement clé : entier ;
                             suiv : ↑liste ;
             fin ;

procédure liste-miroir ;
entrée : une liste l : ↑liste ;
sortie : la liste miroir l_miroir : ↑liste ;

var prec, courant, succ : ↑liste ;
début
        si l ≠ nil alors
                début  prec := nil ;
                       courant := l ;
                       tant que courant↑.suiv ≠ nil faire
                              début   succ := courant↑.suiv ;
                                      courant↑.suiv := prec ;
                                      prec := courant ;
                                      courant := succ ;
                              fin ;
                       courant↑.suiv := prec ;
                       l_miroir := courant ;
                fin ;
fin ;
```

La complexité de la procédure est *O(n)* où *n* est la longueur de la liste.