Eléments de Corrections pour le concours 2015

ALSDD :

Exercice 2 (arbre de recherche binaire de tableaux)

1) Construction de l'arbre à partir M tableaux lus (pouvant avoir le même poids)
Pour représenter les tableaux ayant le même poids dans un arbre de recherche binaire on a 3 possibilités :
a. chaque nœud contient dans son champ 'info' une liste de tableaux ayant le même poids.
b. transformer la relation stricte '<' en '≤', c-a-d les valeurs ayant le même poids que le nœud courant seront dans son sous-arbre gauche.
c. transformer la relation stricte '>' en '≥', c-a-d les valeurs ayant le même poids que le nœud courant seront dans son sous-arbre droit.
Comme dans l'énoncé, il est spécifié que chaque nœud contient un seul tableau, on écarte donc l'option a.
Il reste alors le choix entre mettre les valeurs de même poids à gauche (b) ou alors à droite (c).Nous opterons pour cette dernière (l'option c) car elle facilite la solution de la dernière question (suivants inordre).
Algorithme de construction de l'arbre :

```
racine ← nil
POUR i=1,M
        Lire( V ) ;       // lecture d'un tableau de N éléments
        inserer( V, r )
FP

// insertion dans un arbre de recherche en prenant en compte les valeurs de même poids
inserer( V:tableau[N] , var r:ptr )
SI ( r = nil )
        r ← CreerNoeud( V )
SINON
   SI ( Non egal( V , Info(r ) )          // Pour ne pas insérer 2 fois le même tableau
        SI ( poids(V) < poids( Info(r) )
                p ← fg( r ) ;             // cas d'un poids <
                inserer( V, p ) ;         // donc insertion à gauche
                Aff-fg( r, p )
        SINON
                p ← fd( r ) ;             // cas d'un poids ≥
                inserer( V, p )           // donc insertion à droite
                Aff_fd( r, p )
        FSI
  FSI
FSI
```

2) Listage des tableaux de l'arbre en ordre croissant de leurs poids (parcours inordre)

```
inordre( r : ptr )
SI ( r <> nil )
        inordre( fg(r) ) ;
        Afficher( Info(r) ) ;    // affichage d'un tableau de N éléments
        inordre( fd(r) )
FSI
```

3) Requête à intervalle : lister tous les tableaux ayant un poids dans [p1, p2]

```
soit P une pile globale de ptr.
// recherche de p1 …
CreerPile( P ) ;
q ← r ;
trouv ← faux ;
TQ ( r <> nil && Non trouv )
        SI ( p1 < poids(Info(q)) )
                Empiler( P , q ) ;
                q ← fg(r)
        SINON
                SI ( p1 > poids(Info(q) )
                        q ← fd(q)
                SINON
                        trouv ← vrai
                FSI
        FSI
FTQ ;
SI ( Non trouv )
        q ← Suivant_inordre(q)
FSI ;
// affichage des nœuds appartenant à l'intervalle [p1, p2] …
stop ← faux ;
TQ ( q <> nil && Non stop )
        SI ( poids(q) ≤ p2 )
                Afficher( Info(q) ) ; // Affichage des éléments d'un tableaux
                q ← Suivant_inordre(q)
        SINON
                stop ← vrai
        FSI
FTQ

// algo pour le suivant inordre de q
Suivant_inordre( q:ptr) : ptr
SI ( fd(q) <> nil )
        q ← fd(q) ;
        TQ ( fg(q) <> nil )
                Empiler( P, q ) ;
                q ← fg(q)
        FTQ
SINON
        SI ( Non PileVide(P) )
                Depiler( P , q )
        SINON
                q ← nil
        FSI
FSI ;
return q
```

SFSD :

Exercice 3 (équilibrage du contenu de deux blocs consécutifs d'un fichier (TOF)

```
// lecture des blocs en entrée ...
Ouvrir( F, « ... », 'A') ;
LireDir(F, n, buf1 ) ;
LireDir(F, m, buf2 ) ;
// opération d'équilibrage en MC ...
n ← buf1.NB + buf2.NB ;
SI ( buf1.NB > buf2.NB )
        // de buf1 vers buf2
        q ← buf1.NB – (n div 2) ;      // nombre d'élément à transférer vers buf2
        // décalages dans buf2 ...
        j ← buf2.NB ;
        TQ ( j ≥ 1 )
                buf2.Tab[ j+q ] ← buf2.Tab[ j ] ;
                j ← j-1
        FTQ ;
        // transfert de buf1 vers buf2 …
        POUR j = 1 , q
                buf2.Tab[ j ] ← buf1.Tab[ buf1.NB-q+j ] ;
        FP ;
        // m-a-j des champs NB …
        buf1.NB ← buf1.NB – q ;
        buf2.NB ← buf2.NB + q
SINON
        //  de buf2 vers buf1
        q ← buf2.NB – (n div 2) ;      // nombre d'élément à transférer vers buf1
        // transfert de buf2 vers buf1 …
        POUR j = 1 , q
                buf1.Tab[ buf1.NB+j ] ← buf2.Tab[ j ] ;
        FP ;
        // décalages dans buf2 …
        POUR j = q+1 , buf2.NB
                buf2.Tab[ j-q ] ← buf2.Tab[ j ] ;
        FP ;
        // m-a-j des champs NB …
        buf1.NB ← buf1.NB + q ;
        buf2.NB ← buf2.NB - q
FSI ;
// écriture des blocs ...
EcrireDir( F, n, buf1 ) ;
EcrireDir( F, m, buf2) ;
Fermer(F)
```