Eléments de Corrections pour le concours 2017

ALSDD :

Exercice 2 (Arbre de recherche binaire)

- L'insertion d'une valeur X suivie d'une valeur Y dans un arbre de recherche binaire R est différente de l'insertion de Y suivie par X, car les arbres obtenus sont différents (même s'ils contiennent les mêmes valeurs). Les emplacements des nœuds associés à X et Y dans l'arbre résultat diffère selon l'ordre des opérations d'insertion. Donc on en déduit que l'insertion n'est pas commutative.

- Affichage de toutes les valeurs supérieures à une valeur donnée V dans un arbre de recherche binaire R. Pour limiter au maximum le nombre de nœuds visités, on adoptera alors l'algorithme de la requête à intervalle :

```
Soit P une pile de pointeurs globale
CreerPile(P) ;
// recherche de V …
q ← R ;   trouv ← faux ;
TQ ( q <> nil && Non trouv )
        SI ( V < Info(q) )
                Empiler( P , q ) ;
                q ← fg(q)

        SINON
                SI ( V > Info(q) )
                        q ← fd(q)
                SINON
                        trouv ← vrai
                FSI
        FSI
FTQ ;
// Parcours des suivant inordre jusqu' à la fin …
q ← Suiv_inordre(q) ;
TQ ( q <> nil )
        Afficher ( Info(q) ) ;
        q ← Suiv_inordre(q)
FTQ

// la fonction Suiv_inordre donne le prochain nœud à être visité en inordre
Suiv_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
```

- Pour vérifier si le parcours infixé (c-a-d inordre) d'un arbre donne les valeurs en ordre croissant, on réalise le parcours inordre et on vérifie que la valeurs courante est toujours supérieure ou égale à la valeurs précédente :

```
CreerPile(P)
q ← racine ;
Continu ← vrai ;
premiere_val ← vrai ;
ordonné ← vrai ;
TQ ( Continu && ordonné )
        TQ ( q <> nil )
                Empiler( P , q ) ;
                q ← fg(q)
        FTQ ;
        SI ( Non PileVide(P) )
                Depiler( P , q )
                SI ( premiere_val )
                        premiere_val ← faux ;
                SINON
                        SI ( prec > Info(q) )   ordonné ← faux   FSI
                FSI ;
                prec ← Info(q) ;
                q ← fd(q)
        SINON
                Continu ← faux
        FSI
FTQ ;

SI ( ordonné )
        Afficher(« les valeurs sont ordonnées »)
SINON
        Afficher(« échec »)
FSI
```

Une deuxième solution, consiste à vérifier récursivement que l'arbre de racine r est bien un arbre de recherche. Par exemple on effectue un parcours postordre et on vérifie que les sous-arbres gauche et droit sont bien des arbres de recherche et vérifie aussi que la plus grande du sous-arbre gauche est inférieure ou égale à la racine et que cette dernière est aussi inférieure ou égale à la plus petite valeur du sous-arbre droit

```
// cette fonction retourne vrai si les éléments de r sont ordonnés
// elle retourne aussi dans Vmin (resp. dans Vmax) la plus petite (resp. la plus grande) valeur
// du sous-arbre gauche (resp. droit) de r
```

```
ordonné( r:ptr, var Vmin:entier, var Vmax:entier ) : booleen
var locales
        min1, min2, max1, max2 : entier
SI ( r <> nil )
        min1 ← Info(r) ;        // ces initialisations sont
        max1 ← min1 ;           // nécessaires dans le cas
        min2 ← min1 ;           // où le fg ou le fd de r
        max2 ← min1 ;           // sont à nil.

        SI ( ordonné( fg(r), min1, max1 ) && ordonné(fd(r), min2, max2)  )
                SI ( max1 ≤ Info(r) && Info(r) ≤ min2 )
                        Vmin = min1 ;
                        Vmax =max2 ;
                        return vrai
                SINON
                        Vmin = min1 ;
                        Vmax =max2 ;
                        return faux
                FSI
        SINON
                Vmin = min1 ;
                Vmax =max2 ;
                return faux
        FSI
SINON
        // cas d'un arbre vide (c-a-d r = nil)
        return vrai
FSI
```