

Manipulation d'Arbre Binaire



Dossier de programmation – Projet N°2

TP arbre binaire : manipulation d'arbre binaire

Préambule :

Dans ce TP, je fais appel à des fonctions du projet précédent (Liste doublement chaînée) afin de trier les valeurs de l'arbre dans l'ordre croissant, et avec une allocation dynamique de la mémoire (permettant de limiter le nombre de nœuds de l'arbre en fonction de la machine, et non de manière statique comme le ferait un tableau intermédiaire).

Cette solution n'est pas la plus efficace (on aurait pu utiliser une liste simplement chaînée et effacer celle-ci au fur et à mesure que l'on accède aux valeurs de cette liste) mais elle permet de vérifier la portabilité de ces fonctions au sein d'un autre projet.

Les fonctions propres aux listes doublement chaînées ne seront pas décrites ici car elles sont strictement identiques au projet précédent, dont vous avez déjà le dossier de programmation.

Ordonner un arbre binaire :

Le principe est de récupérer toutes les valeurs de l'arbre dans une liste chaînée ordonnée, puis de parcourir l'arbre (parcours infixe) afin de remettre les valeurs dans les nœuds pour qu'elles soient dans l'ordre et que l'arbre soit ordonné. Le parcours infixe permet de lire les valeurs d'un arbre ordonné dans l'ordre croissant, on utilise ce principe, mais et affectant les nœuds au lieu de les lire. Cet arbre est ordonné mais n'est pas optimisé comme il aurait pu l'être.

En effet, Adel'son, Vel'skii et Landis (VLS) ont créé un algorithme permettant de construire un arbre binaire équilibré (ayant le même nombre de fils gauche que de fils droit) ce qui permet un parcours minimal de l'arbre de la racine à la feuille. Il n'est pas utilisé ici car il a été demandé de conserver la même structure d'arbre, et la structure de nœud imposée ne permet pas de l'appliquer (à moins de surcharger le code ou la structure).

Fonctions : listeValeurArbre parcourt l'arbre et dispose les valeurs rencontrées dans une liste ordonnée
ordonnerArbre récupère les valeurs de la liste et les remet (selon l'ordre infixe) dans l'arbre, conservant ainsi la même structure.

Fonction listeValeurArbre

Description: fait une liste doublement chaînée avec les valeurs de l'arbre, ces valeurs sont ordonnées

Entrées: a - arbre - L'arbre dont on liste les valeurs
l - pointeur sur TypeListeDC - la liste que nous allons remplir

Sorties: vide

Lexique Local des Fonctions :

insereEltListe (l : pointeur de TypeListeDC, valeur : infoNoeud)
C'est une fonction du projet précédent
Elle insère 'valeur' dans la liste 'l' de manière ordonnée
arbreExiste(a : arbre)
Retourne VRAI si a est différent de NIL

Idée de l'algorithme :

Si arbre non vide alors
listeValeurArbre(filsGauche(a))
insérer la valeur courante dans la liste de manière ordonnée
listeValeurArbre(filsDroit(a))
Fsi

listeValeurArbre(a : arbre ,l : pointeur de TypeListeDC) : ret vide

Si !arbreExiste(a) alors
listeValeurArbre(filsGauche(a),l)
insereEltListe(l,valeur(a))
listeValeurArbre(filsDroit(a),l)
Fsi

Fonction ordonnerArbre

Description: Parcours l'arbre de manière Infixée (pour les insérer dans l'ordre) et insère les valeurs

Entrées: a - arbre - L'arbre où on va copier les valeurs de la liste
l - pointeur sur TypeListeDC - la liste avec laquelle nous allons remplir l'arbre

Sorties: vide

Lexique Local des Fonctions :

arbreExiste(a : arbre)
Retourne VRAI si a est différent de NIL

Idée de l'algorithme :

Si l'arbre existe alors
 ordonnerArbre(filsGauche(a),l)
 la clé de l'arbre reçoit la première valeur de la liste (l plus petite)
 Si la liste n'est pas finie
 La tête de la liste pointe sur le successeur
 ordonnerArbre(filsDroit(a),l)
Fsi
Fsi

ordonnerArbre(a : arbre , l : pointeur de TypeListeDC) : ret vide

Si arbreExiste(a) alors //Si branche incomplète
 ordonnerArbre(filsGauche(a),l) // Pour aller tout a gauche
 a.↑clé ← l.↑tête↑clé //La racine correspond à la tête de la liste
 Si l.↑tête↑succ != CNULL) //Tant qu'il y a des valeurs dans la liste
 l.↑tête←l.↑tête↑succ
 ordonnerArbre(filsDroit(a),l) //La valeur suivante étant pour le fils droit
Fsi
Fsi

Insérer une valeur dans un arbre binaire ordonné :

L'insertion dans un arbre binaire ordonné est assez simple, la fonction a été faite en TD. Elle repose sur le fait que l'on insère une valeur uniquement dans les feuilles, simplifiant ainsi l'algorithme car on a plus à gérer les échanges de pointeurs entre père et fils afin d'échanger les nœuds dans l'arbre.

Fonctions : insereValeur est l'interface entre l'utilisateur et la fonction insertion
insertion est la fonction permettant d'insérer la valeur dans l'arbre

Fonction insererValeur

Description: demande à l'utilisateur la valeur à insérer dans l'arbre ordonné

Entrées: a - arbre - l'arbre dans lequel va s'effectuer l'insertion

Sorties: retourne un arbre dans lequel a été insère la valeur

Lexique Local des Variables :

val : infoNoeud – La valeur saisie par l'utilisateur

Lexique Local des Fonctions :

insertion (a : arbre , val : infoNoeud) : ret arbre

Idée de l'algorithme :

```
écrire « entrez la valeur à insérer »  
val ← lire  
a ← insertion(a, val)  
retour_de_insereValeur ← a
```

insererValeur(a : arbre) : ret arbre

```
écrire « Tapez la valeur a insérer: »  
val ← lire  
a ← insertion(a, val)  
retour_de_insererValeur ← a
```

Fonction insertion

Description: Insère une valeur dans un arbre ordonné

Entrées: a - arbre - l'arbre dans lequel on va insérer la valeur
val - infoNoeud - la valeur à insérer dans l'arbre

Sorties: retourne un arbre dans lequel val est insérée

Lexique Local des Fonctions :

arbreExiste(a : arbre)
Retourne VRAI si a est différent de NIL
valeur(a : arbre)
Retourne la valeur de la racine de l'arbre a

Idée de l'algorithme :

```
Si l'arbre n'existe pas alors
  a ← creerNoeud()
Si l'arbre existe
  On stocke 'val' dans le noeud courant
Fsi
Sinon
  Si val < valeur(a)
    filsGauche(a) ← insertion(filsGauche(a), val)
  sinon
    si val > valeur(a)
      filsDroit(a) ← insertion(filsDroit(a))
    sinon
      écrire « La valeur existe déjà »
    fsi
  fsi
fsi
```

arbre insertion(a : arbre , val : infoNoeud) : ret arbre

```
Si !arbreExiste(a) alors //Si premier élément (un arbre vide est bien orienté, non?)
  a ← creerNoeud()
Si arbreExiste(a) alors
  a.↑clé ← val
Fsi
Sinon
  Si val < valeur(a) )
    a.↑filsGauche ← insertion( filsGauche(a) , val )
  Sinon
    Si val > valeur(a) )
      a.↑filsDroit ← insertion( filsDroit(a), val )
    Sinon
      Si val = valeur(a) )
        Ecrire « La valeur existe déjà »
      Fsi
    Fsi
  Fsi
Fsi
Retour_de_insertion ← a
```