Chapitre 13 : Les Arbres

Introduction

Un arbre est une collection (ou un ensemble) de nœuds. Les nœuds sont reliés entre eux. Un arbre est accessible à travers la variable RACINE. La RACINE contient l'adresse du premier élément de l'arbre c'est à dire l'adresse du premier NOEUD. L'arbre est vide si RACINE = NIL sinon l'arbre existe c'est à dire l'arbre n'est pas vide.

Remarque

Par défaut un arbre est n-aire c'est à dire chaque nœud peut avoir au plus n fils.

Exemples

- ✓ *pour l'arbre BINAIRE, chaque nœud a au plus 2 FILS (UN FILS GAUCHE ET UN FILS DROIT),*
- ✓ *pour un arbre TERNAIRE, chaque nœud a au plus trois fils.*

Dans tout le cours, nous étudierons que les arbres BINAIRES.

I. La Topologie d'un Arbre Binaire

1- ***RACINE*** *: c'est le nœud qui permet d'accéder à l'arbre. Elle contient l'adresse du premier nœud de l'arbre. Ce nœud représente la porte d'entrée de l'arbre*
2- ***Noeud Fils*** *: c'est un nœud descendant. Ce nœud peut être gauche ou droit par rapport au NOEUD PARENT*
3- ***Noeud gauche*** *: c'est un nœud fils situé à gauche par rapport à son nœud parent*
4- ***Noeud droit*** *: c'est un nœud fils situé à droite par rapport à son nœud parent*
5- ***Feuille*** *ou* ***nœud final*** *ou* ***nœud de terminaison*** *: c'est un nœud fils et qui n'est pas père*
6- ***Noeud interne*** *ou* ***nœud intermédiaire*** *: c'est un nœud fils et père*
7- ***Taille*** *: c'est le nombre de nœuds que compose l'arbre*
8- ***Profondeur*** *: c'est le niveau maximal d'un arbre*
9- ***Sous-arbre*** *: c'est une partie d'un arbre à partir d'un nœud. C'est une sous - descendance d'un arbre. Un sous-arbre peut être gauche ou droit.*

II. Déclaration

Chaque nœud de l'arbre contient un ou plusieurs champs informations mais dispose exactement de 2 champs pointeurs (l'un pour sa descendance gauche et l'autre pour sa descendance droite).

Type nomNoeud = ↑Structure

Debut

info(s) : type(s)

noeudGauche, noeudDroit : nomNoeud

Fin

var Racine : nomNoeud

Exemple 1 *: Déclarer un Arbre binaire d'entiers*

Type Arbre = ↑structure

Debut

info : entier

nG, nD : Arbre

Fin

Var Racine : Arbre

Exemple 2 *: Déclarer un Arbre binaire de personnes. Personne (nom, prenom, genre, age)*

Type PERSONNE = structure

DEBUT

nom, prenom : chaine

genre : 'M', 'F'

age : entier

Fin

Type ArbrePersonnes = ↑structure

Debut

info : PERSONNE

nG, nD : ArbrePersonnes

Fin

Var Racine : ArbrePersonnes

Remarques

- ✓ *Si nG = NIL alors le nœud n'a pas de descendance Gauche (Pas de sous-arbre Gauche)*
- ✓ *Si nD = NIL alors le nœud n'a pas de descendance Droite (Pas de sous-arbre Droit)*

- ✓ *Si nG = NIL et nD = NIL alors le nœud est une Feuille*

III. Les méthodes de parcours d'un arbre

Il existe 3 méthodes de parcours possibles pour un arbre :

- ✓ *Le parcours préfixé encore appelé parcours pré-ordre*
- ✓ *Le parcours infixé encore appelé parcours symétrique*
- ✓ *Le parcours postfixé encore appelé parcours post-ordre*

Toutes ces méthodes peuvent être effectuées de la Gauche vers la Droite ou de la Droite vers la Gauche.

1. Le parcours en profondeur

A. Le parcours en profondeur récursif

a. Le parcours préfixé récursif

Il implémente le principe du parcours préfixé. Il peut être fait de la gauche vers la droite ou de la droite vers la gauche.

***Exemple** :*

Soit un arbre binaire de personnes, écrire un module qui affiche les personnes en utilisant le parcours préfixé gauche → droite.

Personne (nom, prenom, age, sexe)

Solution 1

```
Type Personne = Structure
Debut
        nom, prenom : chaine
        age : entier
        sexe : 'M','F'
Fin
Type Arbre = ↑structure
Debut
        info : Personne
        G, D : Arbre
```

Fin

Var Racine : Arbre

Procedure PrefixeGaucheDroite(Donnee Racine:Arbre)

var p : Arbre

Debut

p ← Racine

Si (p != NIL) Alors

Ecrire "Le nom de la personne est ", p↑.info.nom

Ecrire "Le prenom de la personne est ", p↑.info.prenom

Ecrire "L'age de la personne est ", p↑.info.age

Ecrire "Le sexe de la personne est ", p↑.info.sexe

PrefixeGaucheDroite(p↑.G)

PrefixeGaucheDroite(p↑.D)

FinSi

Fin

Solution 2

Type Arbre = ↑Structure

Debut

nom, prenom : chaine

age : entier

sexe : 'M','F'

G, D : Arbre

Fin

Var Racine : Arbre

Procedure PrefixeGaucheDroite(Donnee Racine:Arbre)

var p : Arbre

Debut

p ← Racine

Si (p != NIL) Alors

Ecrire "Le nom de la personne est ", p↑.nom

Ecrire "Le prenom de la personne est ", p↑.prenom

Ecrire "L'age de la personne est ", p↑.age

Ecrire "Le sexe de la personne est ", p↑.sexe

PrefixeGaucheDroite(p↑.G)

PrefixeGaucheDroite(p↑.D)

FinSi

Fin

b. Le parcours infixé récursif

Il utilise le principe déjà vu pour le parcours préfixé. Il peut se faire de la gauche vers la droite ou de la droite vers la gauche.

Exercice d'application *:*

Soit un arbre binaire d'entiers, écrire un module qui affiche les valeurs de l'arbre en utilisant le parcours infixé gauche → droite.

Solution

Type Arbre = ↑Structure

Debut

info : entier

G, D : Arbre

Fin

var R : Arbre

Procedure InfixeGaucheDroite(Donnee R:Arbre)

var p : Arbre

Debut

p ← R

Si (p!= NIL) Alors

InfixeGaucheDroite(p↑.G)

Ecrire "La valeur est ", p↑.info

```
        InfixeGaucheDroite(p↑.D)
    FinSi
Fin
```

c. Le parcours postfixé récursif

Idem pour les autres algos récursifs, il implémente son principe.

Exercice d'application *:*

Soit un arbre de caractères, écrire un module qui affiche les valeurs de l'arbre en utilisant le parcours postfixé Gauche → Droite

Solution

```
Type Arbre = ↑Structure
Debut
        info : caractère
        G, D : Arbre
Fin
var R : Arbre
Procedure PostFixeGaucheDroite(Donnee R:Arbre)
var p : Arbre
Debut
    p ← R
    Si (p != NIL) Alors
            PostFixeGaucheDroite(p↑.G)
            PostFixeGaucheDroite(p↑.D)
            Ecrire "La valeur est ", p↑.info
    FinSi
Fin
```

B. Le parcours en profondeur itératif

Il nécessite l'utilisation de piles. Ces piles permettent de sauvegarder les nœuds parcourus. Toutes les méthodes vues peuvent être parcourues mais différemment.

a. Le parcours préfixé itératif

Il utilise le même principe mais de façon itérative grâce à l'utilisation de boucles et de piles de nœuds.

Exercice d'application 1 :

Soit un arbre de caractères, écrire un module qui affiche les valeurs de l'arbre en utilisant le parcours préfixé Gauche → Droite

Solution

```
Type Arbre = ↑Structure
Debut
        info : caractères
        g, d : Arbre
Fin
Type Pile = ↑Structure
Debut
        info : Arbre
        suiv : Pile
Fin
var R : Arbre
Procedure PrefixeGaucheDroite (Donnée R:Arbre)
var p : Pile
    S : Arbre
Debut
    initPile(p)
    Si (R = NIL) Alors
            Ecrire "L'arbre est vide"
```

```
    Sinon
        S ← R
        TantQue (S != NIL ou pileVide(p) = faux) Faire
            TantQue (S != NIL) Faire
                Ecrire "La valeur est ", S↑.info
                Empiler(S, p)
                S ← S↑.g
            FinTantQue
            Depiler(p, S)
            S ← S↑.d
        FinTantQue
    FinSi
Fin
```

Exercice d'application 2 *: Parcours d’un arbre binaire*

Soit un arbre binaire de pointeur racine R dont chaque enregistrement possède les champs suivants :

*- Le champ **noProduit** de type chaine qui représente un numéro du produit*

*- Le champ **libelle** de type chaine qui représente le libellé d’un produit*

*- Le champ **prix** de type entier qui représente le prix du produit*

*- Le champ **quantité** de type entier qui représente la quantité en stock de ce produit*

*- Le champ **g** de type pointeur (adresse fils gauche)*

*- Le champ **d** de type pointeur (adresse fils droit)*

L’arbre binaire est quelconque. Ecrire l’algorithme suivant :

Procédure statistiques (Donnée R ; résultats NbProd, Total, PC) qui détermine les valeurs suivantes :

- NbProd : nombre total de produits

- Total : le montant total des produits de l’arbre

- PC : pourcentage de produits dont le montant dépasse 500 000 FCFA.

Solution

```
Type Arbre = ↑Structure
Debut
        noProd, libelle : chaine
        prix, quantite : entier
        g, d : Arbre
Fin
Type Pile = ↑Structure
Debut
        info : Arbre
        suiv : Pile
Fin
var R : Arbre

Procedure statistiques (Donnée R : Arbre
                        Résultats NbProd, Total : entier
                                  PC : réel)
var p : Pile
    S : Arbre
    cpt : entier
Debut
        NbProd ← 0
        Total ← 0
        cpt ← 0
        initPile(p)
        Si (R = NIL) Alors
                Ecrire "L'arbre est vide"
        Sinon
                S ← R
```

```
        TantQue (S != NIL ou pileVide(p) = faux) Faire
            TantQue (S != NIL) Faire
                NbProd ← NbProd + 1
                Total ← Total + (S↑.prix * S↑.quantite)
                Si ((S↑.prix * S↑.quantite)>500 000) Alors
                    cpt ← cpt + 1
                FinSi
                Empiler(S, p)
                S ← S↑.g
            FinTantQue
            Depiler(p, S)
            S ← S↑.d
        FinTantQue
        PC ← (cpt*100)/NbProd
    FinSi
Fin
```

b. *Le parcours infixé itératif*

Il se fait avec l'utilisation d'une pile qui va sauvegarder les nœuds. La pile sera associée à un pointeur de parcours qui sera initialisée à Racine.
Le principe défini ci-dessus sera appliqué.

Exercice d'application *:*
Soit un arbre de caractères, écrire un module qui affiche les valeurs de l'arbre en utilisant le parcours infixé itératif gauche → droite.

Solution
Type Arbre = ↑Structure
Debut

```
        info : entier
        g, d : Arbre
Fin
Type pile=↑Structure
Debut
        info : Arbre
        suiv : pile
Fin
var R : Arbre
Procedure parcoursInfixeIteratif(Donnee R:Arbre)
var p : pile
    S : Arbre
Debut
    initPile(p)
    Si (R = NIL) Alors
            Ecrire "L'arbre est vide"
    Sinon
            S ← R
            Tantque (S != NIL ou pileVide(p)=faux) Faire
                    Tantque (S != NIL) Faire
                            empiler(S, p)
                            S ← S↑.g
                    FinTantQue
                    Depiler(p, S)
                    Ecrire "La valeur est ", p↑.info
                    S ← S↑.d
            FinTantQue
    FinSi
Fin
```

c. Le parcours postfixé itératif

Il nécessite l'utilisation de deux (2) piles. Une pile pour les nœuds et une pile pour les sens des nœuds. Par défaut, chaque nœud sera empilé avec la mention Gauche.
Si on dépile le nœud alors sa mention sera également dépilée et si c'est Gauche alors le nœud sera ré-empilé avec la mention Droite. Si la mention du nœud est Droite alors le nœud sera traité et son pointeur à NIL.

Exercice d'application

Soit un arbre binaire d'entiers, écrire un module qui affiche les valeurs en utilisant le parcours postfixé itératif gauche → droite

Solution

```
Type Arbre = ↑Structure
Debut
        info : entier
        g, d : Arbre
Fin
Type pileN=↑Structure
Debut
        info : Arbre
        suiv : pileN
Fin
Type pileS=↑Structure
Debut
        info : 'G', 'D'
        suiv : pileS
Fin
var R:Arbre
Procedure parcoursPostfixeGD(Donnee R:Arbre)
var pN : pileN
```

```
    pS : pileS
    S : Arbre
    sens : caractères
Debut
        Si (R=NIL) Alors
                Ecrire "L'arbre est vide"
        Sinon
                S ← R
                initPile(pN)
                initPile(pS)
                TantQue (pileVide(pN) = faux ou S != NIL) Faire
                        TantQue (S!=NIL) Faire
                                Empiler(S, pN)
                                Empiler('G', pS)
                                S ← S↑.g
                        FinTantQue
                        Depiler(pN,S)
                        Depiler(pS, sens)
                        Si (sens='G') Alors
                                Empiler(S, pN)
                                Empiler('D', pS)
                                S ← S↑.d
                        Sinon
                                Ecrire "La valeur est ", S↑.info
                                S ← NIL
                        FinSi
                FinTantQue
        FinSi
```

Fin

2. Le parcours en largeur

Il nécessite l'utilisation d'une file d'attente. Cette file va nous permettre de placer les nœuds à traiter. Les nœuds seront traités par niveau.

***Exercice d'application** :*

Soit un arbre binaire de caractères, écrire un module qui affiche les valeurs de l'arbre en utilisant le parcours en largeur.

Solution

```
Type Arbre = ↑Structure
Debut
        info : caractere
        g, d : Arbre
Fin
Type File = ↑Structure
Debut
        info : Arbre
        suiv : File
Fin
var R:Arbre
Procedure AfficheLargeur(Donnee R:Arbre)
var Tete, Queue : File
        S : Arbre
Debut
        Si (R = NIL) Alors
                Ecrire "L'arbre est vide"
        Sinon
              initFile(Tete, Queue)
```

```
            S ← R
          Enfiler(S, Tete, Queue)
            TantQue (fileVide(Tete, Queue) = faux) Faire
                  Defiler(Tete, Queue, S)
                  Ecrire ("La valeur est ", S↑.info)
                  Si (S↑.g != NIL) ALors
                        Enfiler(S↑.g, Tete, Queue)
                  Sinon
                        Si (S↑.d != NIL) Alors
                              Enfiler(S↑.d, Tete, Queue)
                        FinSi
                  Finsi
            FinTantQue
      FinSi
Fin
```

IV. ***Les arbres ordonnés***

Un arbre est ordonné si les nœuds sont classés soit par ordre croissant soit par ordre décroissant. Un arbre ordonné est encore appelé un arbre de recherche. Dans un arbre de recherche, les doublons sont interdits.

La relation d'ordre est de la sorte ***NG < NP < ND***

Le traitement d'un arbre ordonné applique la dichotomie.

1. La recherche d'une valeur dans un arbre ordonné

La recherche d'une valeur applique la dichotomie. L'arbre de recherche est toujours trié soit par ordre croissant soit par ordre décroissant.

Exercice d'application *:*

Soit un arbre ordonné d'entiers, écrire un module qui recoit une valeur puis recherche si la valeur est présente ou pas dans l'arbre.

Solution

```
Type Arbre = ↑Structure
Debut
        info : entier
        g, d : Arbre
Fin
Var R : Arbre
Fonction RechercheDichotomique(Données Val : entier
                                        R : Arbre) : booleen
Var Trouve : booleen
        S : Arbre
Debut
    Trouve ← Faux
    Si (R = NIL) Alors
            Ecrire "L'arbre est vide"
    Sinon
            S ← R
            TantQue (S != NIL et Trouve = faux) Faire
                    Si (S↑.info = Val) Alors
                            Trouve ← vrai
                    Sinon
                            Si (S↑.info > Val) Alors
                                    S ← S↑.g
                            Sinon
                                    S ← S↑.d
                            FinSi
                    FinSi
            FinTantQue
    FinSi
    Retourner Trouve
```

Fin

2. Insertion de valeur dans un arbre ordonné

L'insertion dans un arbre de recherche respecte les contraintes définies :

- *Pas de redondance de valeurs c'est à dire pas des doublons*
- *Les relations d'ordre entre nœuds doivent être vérifiées*

Exercice d'application *:*

Soit un arbre binaire ordonné d'entiers, écrire un module qui recoit une valeur entière puis insère la valeur dans l'arbre de sorte que l'arbre reste trié.

Solution

```
Type Arbre = ↑Structure
Debut
        info : entier
        g, d : Arbre
Fin
Var R : Arbre
Procedure Insertion(Donnée Val : entier
                    Donnée/Résultat R : Arbre)
var S, svg, pVal : Arbre
    Trouve : booleen
Debut
    Trouve ← RechercheDichotomique (Val, R)
    Si (Trouve = vrai) Alors
        Ecrire "Pas de redondance autorisée donc insertion impossible"
    Sinon
        Allouer (pVal)
        pVal↑.info ← Val
        pVal↑.g ← NIL
```

```
        pVal↑.d ← NIL
        S ← R
        Si (R = NIL) alors
            R ← pVal
        Sinon
            TantQue (S != NIL) Faire
                svg ← S
                Si (S↑.info < Val) Alors
                    R ← S↑.d
                Sinon
                    S ←S↑.g
                FinSi
            FinTantQue
            Si (svg↑.info < Val) Alors
                svg↑.d ← pVal
            Sinon
                svg↑.g ← pVal
            FinSi
      FinSi
  FinSi
Fin
```

3. Création d'arbre ordonné

La création d'un arbre ordonné est la répétition de l'insertion en respectant les contraintes définies.

Exercice d'application *:*

Soit un tableau de 150 entiers, écrire un module qui crée l'arbre ordonné correspondant. L'arbre ne doit contenir que les nombres pairs du tableau ;

Solution

```
Const N=150
Type Tab = tableau[1..N] entier
Type Arbre =↑Structure
Debut
        info : entier
        g, d : Arbre
Fin
var T : Tab
    R : Arbre
Procedure CreationArbreOrdonne(Données T : Tab, N : entier
                                        Résultat R : Arbre)
var i : entier
Debut
    R ← NIL
    Pour i ← 1 à N Faire
        Si (T[i] mod 2 = 0) Alors
            Insertion(T[i], R)
        FinSi
    FinPour
Fin
```