Structures de données, IMA S6

Arbres Binaires

d'après un cours de N. Devésa, Polytech'Lille.

Laure Gonnord
http://laure.gonnord.org/pro/teaching/
Laure.Gonnord@polytech-lille.fr

Université Lille 1 - Polytech Lille

Février 2011

Introduction

Jusqu'à présent, on a vu :

- Les listes contiguës (tableaux).
- Les listes chaînées simples, doublement chaînées ...
- Des variantes des listes : piles, files

▶ Pour l'instant, aucune SD ne permet la hierarchie.

Exemples classiques

Quelques exemples classiques :

- Arbre généalogique
- Organigramme d'une entreprise
- Table des matières d'un livre
- Les expressions arithmétiques
- Un programme !

Terminologie - 1

Dessin !

- Noeuds, racine, feuilles, arêtes
- Fils, père, frère
- Sous-arbre
- Branche, chemin

Terminologie - 2

- Niveau d'un noeud : nombre d'arêtes entre le noeud et la racine
- Hauteur d'un arbre : niveau max des feuilles de l'arbre
- Arbre ordonné (ordre entre chaque fils)
- Degré d'un noeud : nombre de fils
- Arbre n-aire : les noeuds (hors feuilles) sont de degré n.

Arbre Binaire - Définition

Définition récursive d'un arbre binaire (grammaire) :

```
type Noeud
type Arbre ::= Vide | <Noeud,Arbre,Arbre>
```

▶ Dans `<Noeud,Arbre,Arbre>`, le premier arbre désigne le sous-arbre gauche (G) et le deuxième le sous-arbre droit (D). Exemple : expression arithmétique.

Arbre Binaire - Déclaration et primitives

Arbre Binaire dont les feuilles sont de type $< T >$:
A : **Arbre Binaire de** *T*

Primitives sur les arbres binaires (avec **effets de bord**) :

- `initArbre(A)` crée un arbre binaire (AB) vide A.
- `estVide(A)` teste si A est vide
- `<T> valeur(A)` retourne la valeur de la racine.

Arbre Binaire - Primitives 2

Autres primitives

- `ArbreBinaire getGauche(A)` retourne le sous-arbre gauche (pas de copie) (de même `AB getDroite(A)`).
- `putValeur(A,V)` range la valeur V (de type $< T >$ à la racine de A.
- `putDroite(A,D)` : D devient le sous-arbre droit (de même `putGauche`).
- `cons(R,G,D)` construit l'arbre $< R, G, D >$.

Premières remarques

On ne connait rien sur l'implémentation, on utilise les primitives de base.

Remarque : la structure **récursive** d'un AB nous invite à écrire des fonctions récursives.

AB = 1 feuille ?

Fonction *estFeuille(ab) :Booléen*
| **D**: ab : ArbreBinaire
| **Si** *estVide(ab)* **alors**
| | **Retourner** *faux*
| **Sinon**
| | **Retourner** *estVide(getGauche(ab))et estVide(getDroite(ab))*
| **Fsi**
FFonction

Nombre de noeuds d'un AB

Algorithme récursif :

- Si $ab = Vide$: résultat = 0
- Si $ab =< R, G, D >$: 1 + nb noeuds(G) + nb neuds (D).

Fonction *nbNoeuds(ab) :Entier*
| **D**: ab : ArbreBinaire
| **Si** *estVide(ab)* **alors**
| | **Retourner** *0*
| **Sinon**
| | **Retourner** *1 + nbNoeuds(getGauche(ab))+nbNoeuds(getDroite(ab))*
| **Fsi**
FFonction

Hauteur d'un AB

Algorithme récursif :

- Si $ab = Vide$: résultat = 0
- Si $ab =< R, G, D >$: ?

Fonction *hauteurAB(ab) :Entier*
|
|
|
|
FFonction

Les Parcours - 1

But : Effectuer un traitement sur chacun des noeuds d'un AB.
Deux possibilités :

- Parcours en profondeur : récursif (ou pile !), 3 variantes.
- Parcours en largeur (à l'aide d'une file).

▶ les parcours en profondeur sont les plus simples à implémenter.

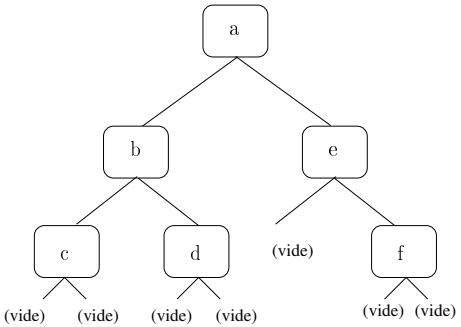
Les Parcours - 2

Parcours en profondeur :

- Traitement de la racine
- Appel récursif sur chacun des sous-arbres

▶ L'effet va dépendre de la place des appels récursifs.

- Préfixe : racine, gauche, droite
- Infixe : gauche, racine, droite
- Suffixe (ou postfixé) : gauche, droite, racine

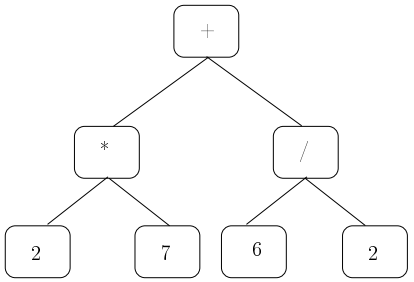


Les Parcours - 3

Parcours en profondeur suffixe pour l'évaluation d'une expression :

Fonction *evalexpr(ab) :Entier*

FFonction



Attention ! cet arbre n'a pas le même type de noeuds.

La Recherche - 1

Constat : Dans un arbre binaire classique, chercher un élément revient à le chercher dans le sous-arbre gauche ET le sous-arbre droit.

▶ Et si on **triait** ?

Définition d'un ABR

Définition récursive $A =< R, G, D >$ est un **arbre binaire de recherche (ABR)** si :

- Tout noeud n_g de G vérifie $valeur(n_g) \leqslant R$
- Tout noeud n_d de G vérifie $R < valeur(n_d)$
- G et D sont des ABR

Remarque : on appelle aussi "arbre ordonné".

Les parcours :

- GRD (infixe) voit les valeurs par ordre croissant
- DRG voit les valeurs par ordre décroissant

La Recherche - 2

Du coup, la recherche dans un ABR :

Fonction *rechercheElem(ab,elem) : Booléen*
D: ab : ArbreBinaire d'entiers (ordonné)
D: elem : Entier

FFonction

- Coût de la recherche = **hauteur de l'arbre**
- Techniques de constructions d'ABR équilibrés

Ajout dans un ABR - avec effet de bord

Ajout(A,V) :

- Si A est vide ?
- Si $A =< R, G, D >$?

Action *ajoutElem(ab,elem)*
D/R: ab : ArbreBinaire d'entiers (ordonné)
D: elem : Entier

FAction

- Passage en D/R de ab (cf ajout dans une liste triée).

D'autres algorithmes/variations classiques

- ajout d'un élément à la racine d'un ABR
- arbres binaires équilibrés par construction : les AVL (cf `http://fr.wikipedia.org/wiki/Arbre_AVL`
- les **tas** : utilisation pour implémenter les files de priorité et aussi trier.

Biblio :
- Chapitre 4 de `http://www.enseignement.polytechnique.fr/profs/informatique/Jean-Jacques.Levy/poly/polyx-cori-levy.ps.gz`
- Chapitres 12 et 13 de l'excellent livre "Introduction à l'algorithmique" (Cormen/Leseison/Rivest/Stein).

Quelques implémentations classiques (en C)

Suivant l'usage, un AB peut être représenté par :

- Un **chaînage** (similaire à une liste chaînée, mais chaque noeud a deux fils)
- Un tableau qui stocke les noeuds, la racine à première position, les autres noeuds ont la position qu'on veut.
- Un tableau où les noeuds ont une position définie à l'avance. La racine est à la position 1. Le noeud à la position p est le père des noeuds des positions $2p$ et $2p+1$.

Implémentation par chaînage - 1

type ArbreBinaire = **pointeur de** *Noeud*
type Noeud = **Structure**
| val : <T>
| gauche : ArbreBinaire
| droit : ArbreBinaire
FStruct

Alors on a :

- `estVide(A)` :
  Retourner $(A = NULL)$
- `initArbre(A)` :
  $A \leftarrow NULL$

Implémentation par chaînage - 2

Autres fonctions de base :

- `valeur(A)` :
  Retourner $(A\uparrow).val)$
- `getGauche(A)` :
  Retourner $(A\uparrow).gauche)$
- `putValeur(A,V)` ?
- `cons(R,G,V)` ?