

Programmation avancée

Les Arbres

Walter Rudametkin

Walter.Rudametkin@polytech-lille.fr
<https://rudametw.github.io/teaching/>

Bureau F011
Polytech'Lille

7 mars 2016

1/24

Les arbres

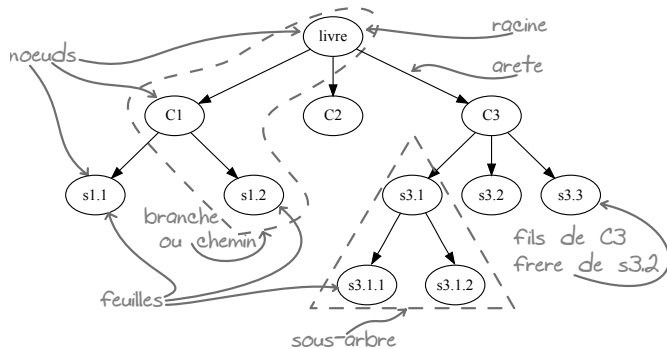
Collection d'informations hiérarchisées

Exemple

- ▶ Arbre généalogique, organigramme d'une entreprise, table des matières d'un livre
- ▶ Organisation d'informations dans une base de données, représentation de la structure syntaxique d'un programme dans les compilateurs

2/24

Les arbres: terminologie



3/24

Les arbres: définitions

- ▶ **Niveau d'un nœud** : nombre d'arêtes entre le nœud et la racine (ex : niveau de s3.2 = 2)
- ▶ **Hauteur d'un arbre** : niveau maximum de l'arbre (3 pour l'exemple)
- ▶ **Arbre ordonné** : l'ordre des fils de chaque nœud est spécifié
- ▶ **Degré d'un nœud** : nombre de fils que le nœud possède
- ▶ **Arbre n-aire** : les nœuds sont de degré n

4/24

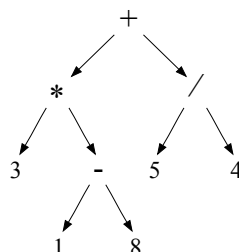
Les arbres binaires

Définition

$B = \emptyset \mid \langle R, G, D \rangle$

où $\begin{cases} R: \text{Noeud Racine} \\ G: \text{Sous-arbre gauche} \\ D: \text{Sous-arbre droite} \end{cases}$

Exemple



5/24

Le type arbre binaire

- ▶ **Déclaration** : A de type `ArbreBinaire de <T>`

Primitives

- ▶ `init_arbre(A)` : crée un arbre binaire vide
- ▶ `vide(A)` : teste si A vide
- ▶ `valeur(A)` : retourne la valeur de la racine
- ▶ `gauche(A)` : retourne le sous-arbre gauche de A
- ▶ `droite(A)` : retourne le sous-arbre droit
- ▶ `put_valeur(A, V)` : range la valeur de V à la racine
- ▶ `put_droite(A, D)` : D devient le sous-arbre droit de A
- ▶ `put_gauche(A, G)` : G devient le sous-arbre gauche de A
- ▶ `cons(R, G, D)` : construit l'arbre $\langle R, G, D \rangle$

6/24

Le type arbre binaire: exemple

- Fonction qui teste si un arbre est une feuille (1 seul nœud)

```

Fonction feuille(A)
  D : A : ArbreBinaire de <T>
  Si vide(A) Alors
    retourner (faux)
  Sinon
    retourner( vide(gauche(A)) et vide(droite(A)) )
Fsi
Ffonction
    
```

7/24

Le type arbre binaire: exemple

- Calcul du nombre de nœuds d'un arbre binaire

$$\text{nb_noeuds}(A) \begin{cases} A = \emptyset : & 0 \\ A = \langle R, G, D \rangle : & 1 + \text{nb_noeuds}(G) + \text{nb_noeuds}(D) \end{cases}$$

```

Fonction nb_noeuds(A)
  D : A : ArbreBinaire de <T>
  Si vide(A) Alors
    retourner (0)
  Sinon
    retourner ( 1 + nb_noeuds(G) + nb_noeuds(D) )
Fsi
Ffonction
    
```

8/24

Algorithmes sur les arbres

3 types de parcours pour effectuer un traitement sur tous les nœuds

- Préfixé
- Postfixé
- Infixé

9/24

Les arbres: parcours préfixé ou RGD

Parcours préfixé ou RGD

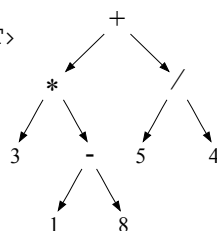
- Traiter la racine
- Traiter le sous-arbre gauche
- Traiter le sous-arbre droit

10/24

Les arbres: parcours préfixé ou RGD

```

Action RGD(A)
  D : A : ArbreBinaire de <T>
  Si non vide(A) Alors
    traiter(valeur(A))
    RGD(gauche(A))
    RGD(droite(A))
Fsi
Faction
    
```



Exemple :
 traiter(valeur(A)) = écrire(valeur(A))
 ⇒ + * 3 - 1 8 / 5 4 (notation préfixée)

11/24

Les arbres: parcours postfixé ou GDR

Parcours postfixé ou GDR

- Traiter le sous-arbre gauche
- Traiter le sous-arbre droit
- Traiter la racine

12/24

Les arbres: parcours postfixé ou GDR

Action GDR(A)

D : A : ArbreBinaire de <T>

Si non vide(A) Alors

GDR(gauche(A))

GDR(droite(A))

traiter(valeur(A))

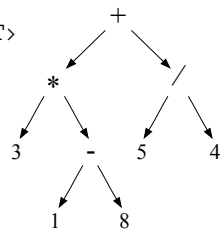
Fsi

Faction

Exemple :

traiter(valeur(A)) = écrire(valeur(A))

⇒ 3 1 8 - * 5 4 / + (notation postfixée)



13/24

Les arbres: parcours infixé ou GRD

- ▶ Traiter le sous-arbre gauche
- ▶ Traiter la racine
- ▶ Traiter le sous-arbre droit

Action GRD(A)

D : A : ArbreBinaire de <T>

Si non vide(A) Alors

GRD(gauche(A))

traiter(valeur(A))

GRD(droite(A))

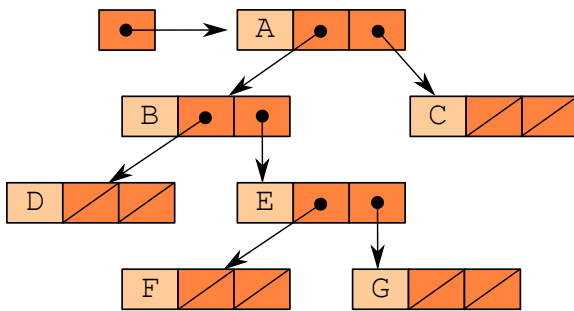
Fsi

Faction

14/24

Implantation des arbres binaires

Représentation par pointeurs



15/24

Implantation des arbres binaires

type ArbreBinaire = pointeur de Noeud

type Noeud = structure

val : <T>

gauche, droite: ArbreBinaire

fin

16/24

Implantation des arbres binaires

Soit A un ArbreBinaire

vide(A) ⇒ retourner(A = NULL)

init_arbre(A) ⇒ A ← NULL

valeur(A) ⇒ retourner(A↑•val)

gauche(A) ⇒ retourner(A↑•gauche)

droite(A) ⇒ retourner(A↑•droite)

put_valeur(A,V) ⇒ A↑•val ← V

put_gauche(A,G) ⇒ A↑•gauche ← G

put_droite(A,D) ⇒ A↑•droit ← D

cons(V,G,D) ⇒ allouer(A)

A↑•val ← V

A↑•gauche ← G

A↑•droit ← D

17/24

Les arbres binaires ordonnées

Rappel

- ▶ Liste contiguë : recherche dichotomique en $O(\log_2 n)$
Ajout / suppression en $O(n)$
- ▶ Liste chaînée : recherche en $O(n)$
Ajout / suppression en temps constant

Arbre binaire ordonné (ou arbre de recherche)

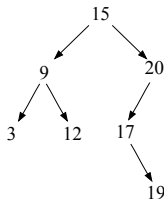
- ▶ Recherche / ajout / suppression : même efficacité
- ▶ Au mieux (arbre équilibré) en $\log_2(n)$

18/24

Les arbres binaires ordonnés: définition

Soit $A = \langle R, G, D \rangle$, A est ordonné si

- ▶ Pour tout nœud nd de G , $\text{valeur}(nd) \leq R$
- ▶ Pour tout nœud nd de D , $\text{valeur}(nd) > R$
- ▶ G et D sont des arbres ordonnés



- ▶ Parcours GRD d'un arbre ordonné \Rightarrow par ordre croissant
- ▶ Parcours DRG d'un arbre ordonné \Rightarrow par ordre décroissant

19/24

Recherche dans un arbre binaire ordonné

Recherche associative d'un élément V

- ▶ $A = \emptyset \Rightarrow$ non trouvé
- ▶ $A = \langle R, G, D \rangle$
 - ▶ $R = X \Rightarrow$ trouvé
 - ▶ $X < R \Rightarrow$ rechercher X dans G
 - ▶ $X > R \Rightarrow$ rechercher X dans D

Coût de la recherche

- ▶ Dans tous les cas $\leq n$
- ▶ Au mieux $\log_2(n)$ si l'arbre est équilibré \Rightarrow techniques de construction d'arbres équilibrés

20/24

Recherche dans un arbre binaire ordonné

Fonction `existe(X, A) : booléen`

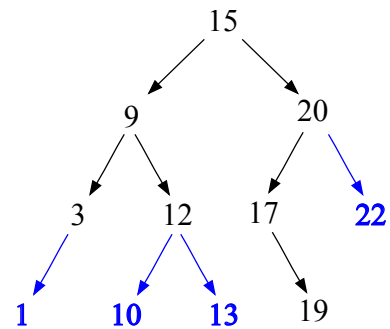
```

D : X : <T> ;
  A : ArbreBinaire
Si vide(A) Alors
  retourner(faux)
Sinon
  Si X=valeur(A) Alors
    retourner(vrai)
  Sinon
    Si X < valeur(A) Alors
      retourner(existe(X,gauche(A)))
    Sinon
      retourner(existe(X,droite(A)))
  Fsi
Fsi
Ffonction
  
```

21/24

Ajout dans un arbre binaire ordonné

Solution simple : ajout en feuille



22/24

Ajout dans un arbre binaire ordonné

Ajout(A, V) :

- ▶ $A = \emptyset \Rightarrow A = \langle V, \emptyset, \emptyset \rangle$
- ▶ $A = \langle R, G, D \rangle$
 - ▶ $V \leq R \Rightarrow$ ajouter V dans `gauche(A)`
 - ▶ $V > R \Rightarrow$ ajouter V dans `droite(A)`
- ▶ Utilisation du passage de A en D/R pour établir le lien père/fils
- ▶ cf : algorithme récursif d'ajout d'un élément dans une liste chaînée

23/24

Ajout dans un arbre binaire ordonné

Action `ajout(V, A)`

```

D : V : <T> ;
D/R : A : ArbreBinaire de <T>
Si vide(A) Alors
  A ← cons(V, ∅, ∅)
Sinon
  Si V ≤ valeur(A) Alors
    ajout (V, gauche(A))
  Sinon
    ajout (V, droite(A))
  Fsi
Fsi
Faction
  
```

24/24