

Programmation avancée

Les Arbres

Walter Rudametkin

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

Bureau F011
Polytech Lille

CM7

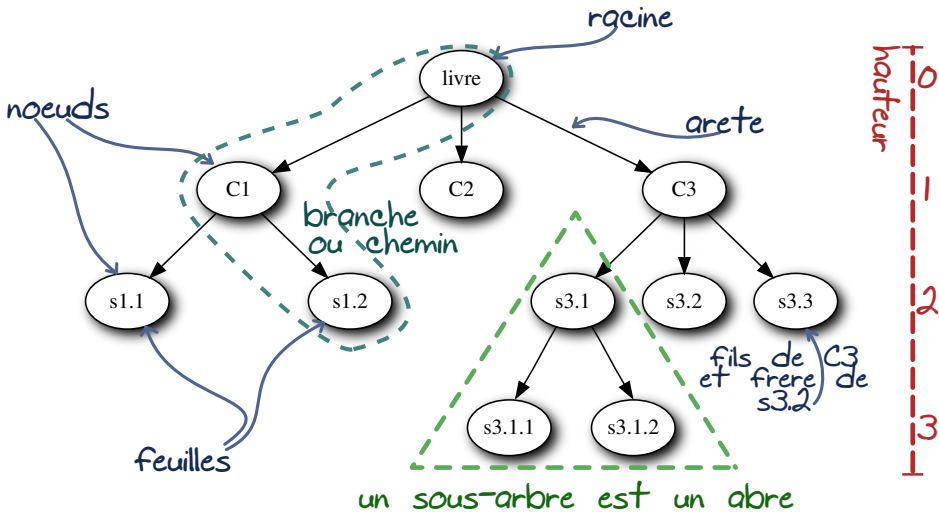
Les arbres

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

Les arbres: terminologie



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 sortant d'un nœud** : nombre de fils que le nœud possède
- ▶ **Arbre n-aire** : les nœuds sont de degré n

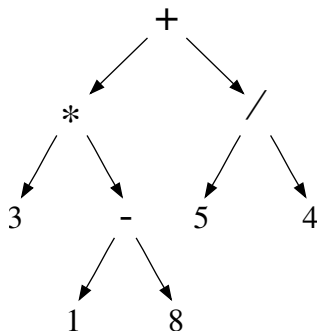
Les arbres binaires

Définition

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

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

Exemple



Le *type* arbre binaire

- ▶ **Déclaration** : A de type ArbreBinaire de $\langle T \rangle$

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(v, G, D)` : construit l'arbre $\langle v, G, D \rangle$

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

Le *type* arbre binaire: exemple

- Calcul du nombre de noeuds 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(gauche(A))
+ nb_noeuds(droite(A)))

Fsi

Ffonction

Algorithmes sur les arbres

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

- ▶ Préfixé
- ▶ Postfixé
- ▶ Infixé

Les arbres: parcours prefixé ou RGD

Parcours prefixé ou RGD

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

Les arbres: parcours prefixé ou RGD

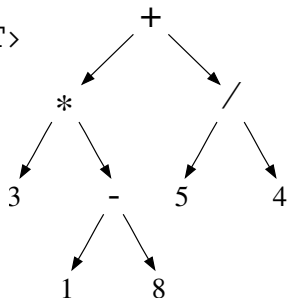
Action RGD(A)

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)

Les arbres: parcours postfixé ou GDR

Parcours postfixé ou GDR

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

Les arbres: parcours postfixé ou GDR

Action GDR(A)

D : A : ArbreBinaire de <T>

Si non vide(A) Alors

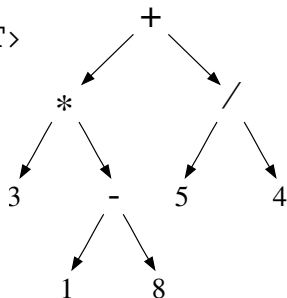
GDR(gauche(A))

GDR(droite(A))

traiter(valeur(A))

Fsi

Faction



Exemple :

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

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

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))

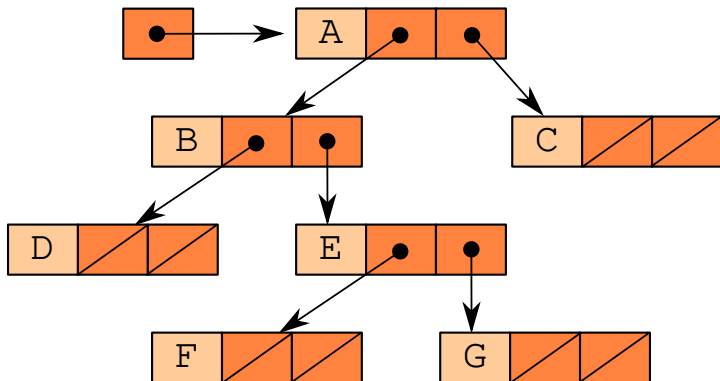
GRD(droite(A))

Fsi

Faction

Implantation des arbres binaires

Représentation par pointeurs



Implantation des arbres binaires

```
type ArbreBinaire = pointeur de Noeud
```

```
type Noeud = structure  
    val : <T>  
    gauche, droit: ArbreBinaire
```

```
fin
```


Implantation des arbres binaires

Soit A un ArbreBinaire

<code>vide(A)</code>	\Rightarrow	<code>retourner(A = NULL)</code>
<code>init_arbre(A)</code>	\Rightarrow	<code>A \leftarrow NULL</code>
<code>valeur(A)</code>	\Rightarrow	<code>retourner(A\uparrow•val)</code>
<code>gauche(A)</code>	\Rightarrow	<code>retourner(A\uparrow•gauche)</code>
<code>droite(A)</code>	\Rightarrow	<code>retourner(A\uparrow•droite)</code>
<code>put_valeur(A,v)</code>	\Rightarrow	<code>A\uparrow•val \leftarrow v</code>
<code>put_gauche(A,G)</code>	\Rightarrow	<code>A\uparrow•gauche \leftarrow G</code>
<code>put_droite(A,D)</code>	\Rightarrow	<code>A\uparrow•droit \leftarrow D</code>
<code>cons(v,G,D)</code>	\Rightarrow	<code>allouer(A)</code>
		<code>A\uparrow•val \leftarrow v</code>
		<code>A\uparrow•gauche \leftarrow G</code>
		<code>A\uparrow•droit \leftarrow D</code>

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

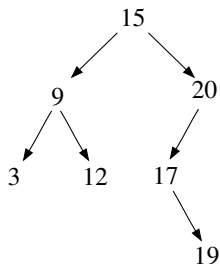
Arbre binaire ordonné (ou arbre de recherche)

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

Les arbres binaires ordonnées: définition

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

- ▶ Pour tout nœud nd de G , $va\text{leur}(nd) \leq R$
- ▶ Pour tout nœud nd de D , $va\text{leur}(nd) > R$
- ▶ 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

Recherche dans un arbre binaire ordonné

Recherche associative d'un élément X

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

Recherche dans un arbre binaire ordonné

Fonction existe(A, X) : 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(gauche(A), X))

Sinon

retourner(existe(droite(A), X))

Fsi

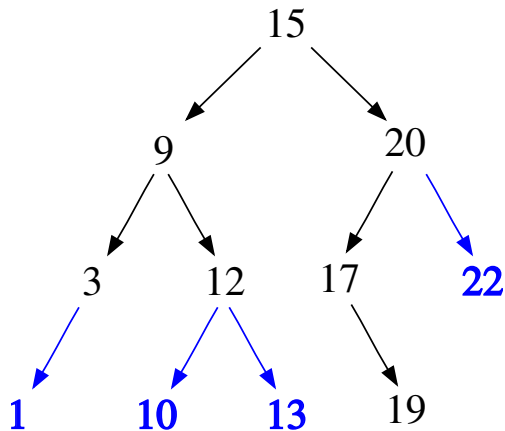
Fsi

Fsi

Ffonction

Ajout dans un arbre binaire ordonné

Solution simple : ajout en feuille



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

Ajout dans un arbre binaire ordonné

Action ajout(A, V)

D : V : $\langle T \rangle$;

D/R : A : ArbreBinaire de $\langle T \rangle$

Si vide(A) Alors

A \leftarrow cons(V, \emptyset , \emptyset)

Sinon

Si V \leq valeur(A) Alors

ajout (**gauche**(A), V)

Sinon

ajout (**droite**(A), V)

Fsi

Fsi

Faction