

# Arbres

Thierry Lecroq

Université de Rouen  
FRANCE

# Plan

- 1 Les arbres
- 2 Les arbres binaires
- 3 Les arbres binaires de recherche
- 4 Les files de priorité

# Les arbres

- structure non linéaire qui permet de hiérarchiser les données
- exemple typique : organisation de stockage des données des systèmes d'exploitation des ordinateurs en répertoires et fichiers

# Une première définition

- collection d'éléments appelés **nœuds** (ou sommets)
- un nœud particulier : la **racine**
- relation « est le parent de »

# Une définition récursive

- 1 Un nœud seul est un arbre. Dans ce cas le nœud est aussi la racine.
- 2 Étant donné un nœud noté  $n$  et  $k$  arbres  $T_0, T_1, \dots, T_{k-1}$  ayant des racines notées  $n_0, n_1, \dots, n_{k-1}$ , on construit un nouvel arbre en prenant  $n$  comme racine et en posant que  $n$  est le parent de  $n_0, n_1, \dots, n_{k-1}$ .  $T_0, T_1, \dots, T_{k-1}$  sont alors appelés les **sous-arbres** de  $n$ , et  $n_0, n_1, \dots, n_{k-1}$  les **successeurs** (ou **enfants**) de  $n$ .

# L'arbre vide

L'**arbre vide** ne possède aucun nœud.

## Quelques définitions

- Soit  $n_0, n_1, \dots, n_{p-1}$  une suite de nœuds tels que  $n_i$  est le parent de  $n_{i+1}$  pour  $0 \leq i < p - 1$  alors une telle suite est appelée un **chemin**.
- La **longueur** de ce chemin est  $p - 1$  (nombre de nœuds moins un).
- Soient  $a$  et  $b$  deux nœuds. S'il existe un chemin allant de  $a$  à  $b$  alors  $a$  est un **antécédent** (ou **ancêtre**) de  $b$  et  $b$  est un **descendant** de  $a$ .
- $a$  est un antécédent **propre** (resp. descendant **propre**) de  $b$  si  $a$  est un antécédent (resp. descendant) de  $b$  et  $a \neq b$ .
- Un nœud ne possédant pas de descendant propre est appelé une **feuille**.
- La **hauteur** d'un nœud est la longueur maximale d'un chemin allant de ce nœud à une feuille.
- La hauteur de l'arbre est la hauteur de sa racine.
- La **profondeur** (ou niveau) d'un nœud est la longueur de l'unique chemin allant de la racine à ce nœud.

# Plan

- 1 Les arbres
- 2 Les arbres binaires**
- 3 Les arbres binaires de recherche
- 4 Les files de priorité

# Les arbres binaires

## Définition

Un **arbre binaire** est :

- soit un arbre vide ;
- soit un arbre dans lequel chaque nœud possède :
  - ▶ 0 successeur ;
  - ▶ ou 1 successeur ;
  - ▶ ou 2 successeurs.

# Parcours d'un arbre binaire

## Parcours suivant l'ordre **préfixe** (préordre)

- R visiter la racine
- G parcourir le sous-arbre gauche suivant l'ordre préfixe
- D parcourir le sous-arbre droit suivant l'ordre préfixe

## Parcours suivant l'ordre **infixe** (symétrique)

- G parcourir le sous-arbre gauche suivant l'ordre symétrique
- R visiter la racine
- D parcourir le sous-arbre droit suivant l'ordre symétrique

## Parcours suivant l'ordre **suffixe** (postordre)

- G parcourir le sous-arbre gauche suivant l'ordre suffixe
- D parcourir le sous-arbre droit suivant l'ordre suffixe
- R visiter la racine

# Parcours d'un arbre binaire

## Parcours en largeur

Dans l'ordre croissant de profondeur des nœuds

# Le TAD arbre binaire

## Opérations

Étant donné un arbre binaire non vide  $A$  et deux arbres binaires  $B$  et  $C$  :

$ARBREVIDE()$  retourne l'arbre vide ;

$ESTARBREVIDE(B)$  fonction booléenne qui retourne vrai si  $B$  est vide et faux sinon ;

$RACINE(A)$  retourne l'élément situé à la racine de  $A$

$GAUCHE(A)$  retourne le sous-arbre gauche de  $A$

$DROIT(A)$  retourne le sous-arbre droit de  $A$

$CONSTRUIRE(x, B, C)$  retourne un arbre dont la racine contient l'élément  $x$  et dont les sous-arbres gauche et droit sont respectivement  $B$  et  $C$

# Plan

- 1 Les arbres
- 2 Les arbres binaires
- 3 Les arbres binaires de recherche**
- 4 Les files de priorité

# Les arbres binaires de recherche (ABR)

## Définition

Un arbre binaire  $A$  est un arbre binaire de recherche si

- 1 les éléments placés dans les nœuds de  $A$  sont pris dans un ensemble totalement ordonné
- 2 pour tout nœud  $s$  de  $A$ ,  $x$  étant l'élément placé dans  $s$ , tous les éléments du sous-arbre gauche de  $s$  sont strictement inférieur à  $x$  et tous les éléments du sous-arbre droit de  $s$  sont strictement supérieur à  $x$

# Insertion d'un élément dans un ABR

La construction d'un ABR s'effectue par insertions successives.

## Exemple

A : 8, 10, 6, 12, 9, 7, 2, 4, 5, 11, 13, 3, 1

B : 6, 12, 8, 10, 2, 13, 1, 4, 5, 9, 3, 11, 7

# Recherche d'un élément dans un ABR

## RECHERCHE( $x, A$ )

- 1 **si** ESTARBREVIDE( $A$ ) **alors**
- 2     **Retourner** faux
- 3 **sinon si**  $x = \text{RACINE}(A)$  **alors**
- 4     **Retourner** vrai
- 5 **sinon si**  $x < \text{RACINE}(A)$  **alors**
- 6     **Retourner** RECHERCHE( $x, \text{GAUCHE}(A)$ )
- 7     **sinon Retourner** RECHERCHE( $x, \text{DROIT}(A)$ )

# Insertion d'un élément dans un ABR

## Notations

Si  $A$  est un arbre binaire alors :

- soit  $A$  est vide ( $A = \langle \rangle$ )
- soit  $A$  est non vide ( $A = \langle r, A_g, A_d \rangle$ )

## Définition récursive

- Si  $A$  est vide alors  $A + x = \langle x, \langle \rangle, \langle \rangle \rangle$
- Si  $A$  est non vide ( $A = \langle r, A_g, A_d \rangle$ ) alors
  - ▶ Si  $x = r$  alors  $A + x = A$
  - ▶ Si  $x < r$  alors  $A + x = \langle r, A_g + x, A_d \rangle$
  - ▶ Si  $x > r$  alors  $A + x = \langle r, A_g, A_d + x \rangle$

## Suppression du plus grand élément dans un ABR non vide

$$A = \langle r, A_g, A_d \rangle$$

$$A - \max(A) = \begin{cases} A_g & \text{si } A_d \text{ est vide} \\ \langle r, A_g, A_d - \max(A_d) \rangle & \text{sinon} \end{cases}$$

# Suppression de l'élément contenu dans la racine d'un ABR non vide

$$A = \langle r, A_g, A_d \rangle$$

$$A - \text{rac}(A) = \begin{cases} A_d & \text{si } A_g \text{ est vide} \\ A_g & \text{si } A_d \text{ est vide} \\ \langle \max(A_g), A_g - \max(A_g), A_d \rangle & \text{sinon} \end{cases}$$

# Suppression d'un élément quelconque dans un ABR

$$A = \langle r, A_g, A_d \rangle$$

- Si  $A$  est vide alors  $A - x = A$
- Sinon  $A = \langle r, A_g, A_d \rangle$  et

$$A - x = \begin{cases} A - \text{rac}(A) & \text{si } x = r \\ \langle r, A_g - x, A_d \rangle & \text{si } x < r \\ \langle r, A_g, A_d - x \rangle & \text{si } x > r \end{cases}$$

# Complexité

dépend de la forme de l'arbre

$n$  nombre d'éléments

pire des cas éléments déjà triés  $O(n)$

meilleur des cas arbre binaire complet  $O(\log n)$

en moyenne  $O(\log n)$

Insertion des éléments dans un ABR puis parcours infixe :  
 $O(n \log n)$  en moyenne

# Plan

- 1 Les arbres
- 2 Les arbres binaires
- 3 Les arbres binaires de recherche
- 4 Les files de priorité**

# File de priorité (ou tas)

- Arbre binaire
- Tout élément placé à un nœud est supérieur à ses deux enfants
- Arbre binaire complet partiellement ordonné (ABCPO)

# File de priorité

- représentation d'un ABCPO à l'aide d'un tableau  $T$
- on place la racine en  $T[1]$
- si un nœud est placé en  $T[i]$ 
  - ▶ son enfant gauche est placé en  $T[2i]$  ;
  - ▶ son enfant droit est placé en  $T[2i + 1]$  ;

## File de priorité - insertion

- insertion d'un élément  $x$  dans une file de priorité  $T$  à  $n$  éléments
- placer  $x$  en  $T[n + 1]$
- faire monter  $x$  (par échanges) jusqu'à sa place
- la file contient maintenant  $n + 1$  éléments

## File de priorité - suppression

- suppression de la racine d'une file de priorité  $T$  à  $n$  éléments
- placer  $x = T[n]$  en  $T[1]$
- faire descendre  $x$  (par échanges) jusqu'à sa place
- la file contient maintenant  $n - 1$  éléments

# Tri par tas (Heap Sort)

- tri de  $n$  éléments à l'aide d'une file de priorité  $T$
- insertion des  $n$  éléments dans  $T$
- pour  $i$  de 1 à  $n$  faire
  - ▶ supprimer la racine et la placer en  $T[n - i + 1]$