# Arbres Binaires de Recherche : Introduction

I. Guessarian

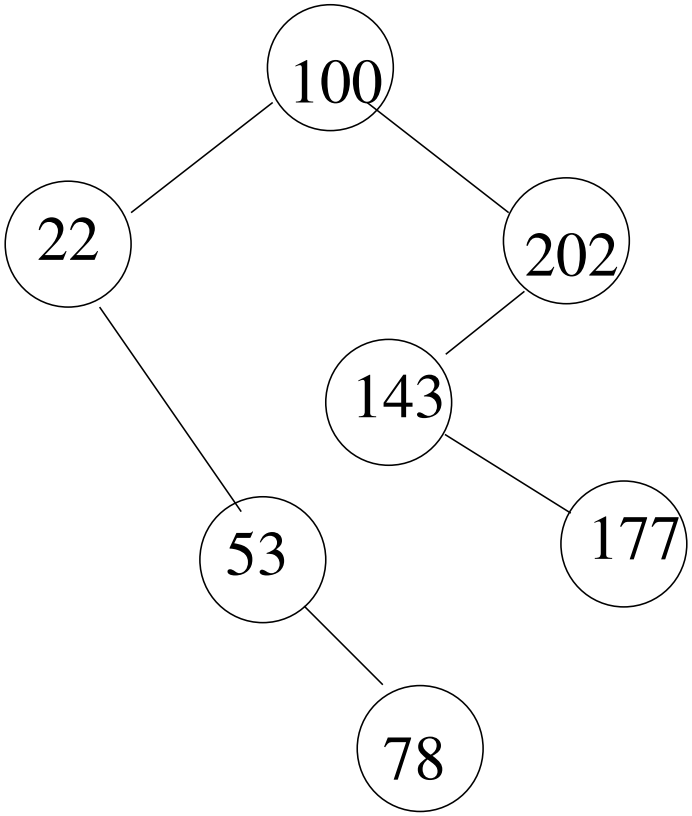
cours ISN 11 janvier 2012

# Arbre Binaire de Recherche

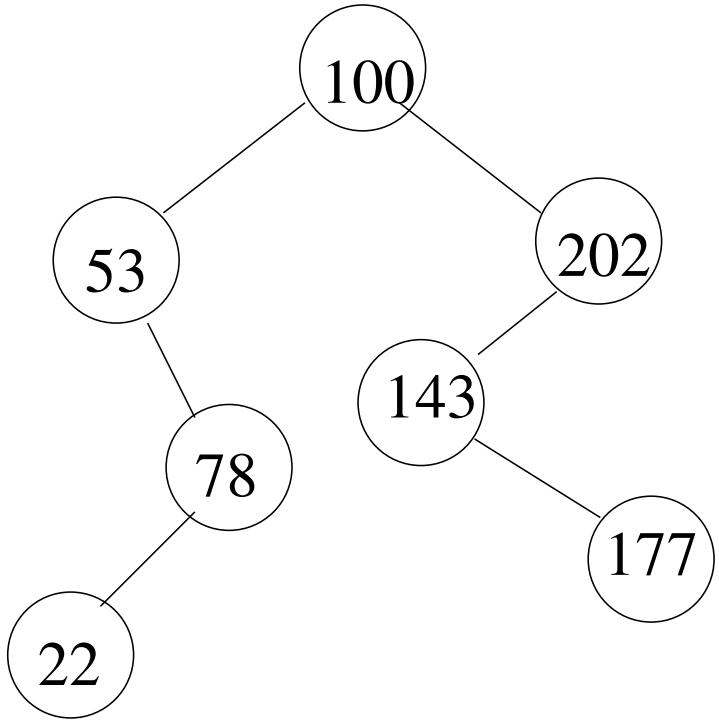
Un Arbre Binaire de Recherche (ABR) est un arbre binaire étiqueté par des éléments d'un ensemble $A$ *totalement ordonné* et tel que si $a$ est l'étiquette d'un nœud $n$,

- tous les nœuds du sous-arbre gauche de $n$ sont étiquetés par des éléments $b$ tels que $b \leq a$, ET
- tous les nœuds du sous-arbre droit de $n$ sont étiquetés par des éléments $b$ tels que $b > a$

# Exemple




a. un arbre binaire de recherche




b. un arbre binaire

# Parcours dans les arbres

- parcours en largeur
- parcours en profondeur
  - parcours infixe
  - parcours préfixe
  - parcours postfixe (ou suffixe)

Le parcours infixe d'un *arbre binaire de recherche* donne la suite des étiquettes en ordre croissant.

# Algorithmes simples sur les arbres binaires (de recherche)

- parcours infixe
- recherche d'un élément
- recherche du maximum (minimum)
- construction d'un ABR (insérer un élément dans un ABR)

# parcours infixe

PARCOURS-IN (arbre $T$ )
Var liste d'entiers $L, L_1, L_2$
**if** $(T == \emptyset)$ **then**
  $L = \emptyset$
**else**
  $L_1$ =PARCOURS-IN(*sous-arbre-gauche*$(T)$)
  $L_2$ =PARCOURS-IN(*sous-arbre-droit*$(T)$)
  $L = L_1 \cdot label_T(racine(T)) \cdot L_2$
**retourner** $L$

# Maximum

MAX (arbre $T$)
 Var nœud $n$
 $n = racine(T)$
 **while** $(T(filsdroit(n)) \neq NIL)$ **do**
  $n = filsdroit(n)$
 **retourner** $T(n)$

**Notation**

On note $T(n)$ l'étiquette du nœud $n$ ( $label_T(n)$)

# Recherche d'un élément

RECHERCHER (arbre $T$, entier $k$)

Var nœud $n$

$n = racine(T)$

**while** $((T(n) \neq NIL) \wedge (T(n) \neq k))$ **do**

  **if** $(k < T(n))$ **then**

    $n = filsgauche(n)$

  **else**

    $n = filsdroit(n)$

**if** $(T(n) \neq NIL)$ **then**

  afficher $k$ est dans $T$

**else**

  afficher $k$ n'est pas dans $T$

INSERER (arbre $T$, entier $v$)

// Algorithme récursif

Var nœud $x$

$x = racine(T)$

**if** $(T == \emptyset)$ **then**

  $T = (v, \emptyset, \emptyset)$

**else**

  **if** $(v < T(x))$ **then**

    INSERER(*sous-arbre-gauche*(T),v)

  **else**

    **if** $(v > T(x))$ **then**

      INSERER(*sous-arbre-droit*(T),v)

INSERER (arbre $T$, entier $v$)

// Algorithme itératif

Var nœud $x, y$
Var arbre $T'$
$T' = T$
$y = NIL$
$x = racine(T)$
**while** $(x \neq NIL)$ **do**
    $y = x$
    **if** $(v < T(x))$ **then**
        $T' = sous\text{-}arbre\text{-}gauche(T')$
    **if** $(v > T(x))$ **then**
        $T' = sous\text{-}arbre\text{-}droit(T')$
    $x = racine(T')$

```
/* suite de INSERER                                          */
parent(x) = y;
if (T == ∅) then
    T = (v, ∅, ∅)
else
    if (v < T(y)) then
        sous-arbre-gauche(y) = (v, ∅, ∅)
    /* le sous-arbre gauche vide de y est
       remplacé par (v, ∅, ∅)                                 */
    if (v > T(y)) then
        sous-arbre-droit(y) = (v, ∅, ∅)
    /* le sous-arbre droit vide de y est
       remplacé par (v, ∅, ∅)                                 */
```

# Exercices

## Exercice 1

Ecrire un algorithme **récursif** qui cherche le maximum d'un arbre binaire de recherche, puis le programme *Javascool* correspondant (à ajouter comme méthode dans la classe abr.jvs).

## Exercice 2

Ecrire un algorithme **récursif** qui cherche un élément dans un arbre binaire de recherche, puis le programme *Javascool* correspondant (à ajouter comme méthode dans la classe abr.jvs).

# Exercices (suite)

## Exercice 3

Ecrire le programme *Javascool* correspondant à l'agorithme TRI-FUSION vu en cours. Pour implémenter la version avec sentinelle du cours, on pourrait au préalable définir une méthode JAVA qui calcule le maximum d'un tableau.