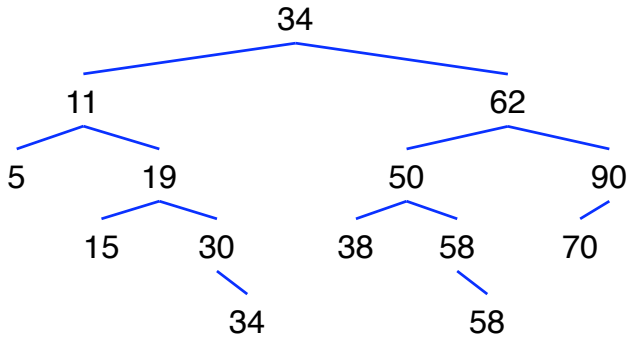
Algorithmique & programmation

Chapitre 5 : Arbres

Arbres binaires ordonnés

Arbre binaire ordonné

- La suite des valeurs dans l'ordre d'un parcours infixé est ordonnée



(nœud↑.droit≠nil, nœud↑.gauche≠nil,
nœud↑.gauche↑.info≤nœud↑.info≤ nœud↑.droit↑.info)

v

(nœud↑.droit=nil, nœud↑.gauche≠nil, nœud↑.gauche↑.info≤nœud↑.info)

v

(nœud↑.droit≠nil, nœud↑.gauche=nil, nœud↑.info≤nœud↑.droit↑.info)

Arbre binaire ordonné

- Par convention, un arbre vide et un arbre réduit à un seul élément sont ordonnés.
- Tous les sous-arbres d'un arbre ordonné sont ordonnés.
- Dans toutes les applications, on choisira de ne laisser l'égalité que d'un seul côté, par exemple à droite, d'où :
  - **(nœud↑.gauche)⁺ < nœud↑.info ≤ (nœud↑.droit)⁺**
- La valeur associée à chaque nœud est
  - **supérieure** à toutes les valeurs du **sous-arbre gauche**, et
  - **inférieure ou égale** à toutes les valeurs du **sous-arbre droit**

Recherche associative dans un ABO

fonction dicho (d racine : pointeur ; d val : t) : booléen ;
spécification *{racine⁺ ordonné}* ➔ *{résultat = (val ∈ racine⁺)}*

- Schéma récursif

➤ racine⁺ = <> ⇨ résultat = faux ✳
➤ racine⁺ ≠ <> ⇨
  ➤➤ racine↑.info = val ⇨ résultat = vrai ✳
  ➤➤ racine↑.info > val ⇨
    résultat = dicho(racine↑.gauche,val) ; ✳
  ➤➤ racine↑.info < val ⇨
    résultat = dicho(racine↑.droit,val) ; ✳

Recherche associative dans un ABO

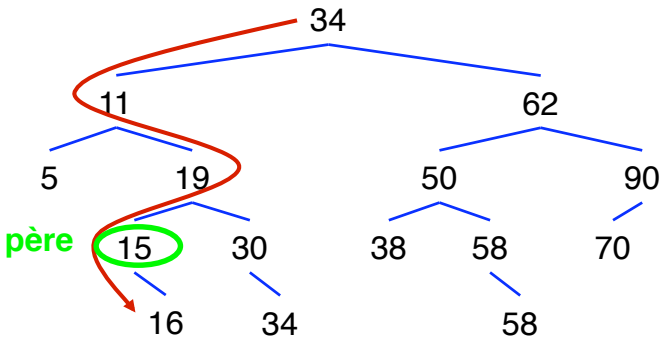
fonction dicho (d racine : pointeur ; d val : t) : booléen ;
spécification *{racine⁺ ordonné}* ➔ *{résultat = (val ∈ racine⁺)}*
debfonc
    si racine = nil alors *{val ∉ racine⁺}*
        retour faux ;
    sinonsi racine↑.info = val alors *{val ∈ racine⁺}*
        retour vrai ;
    sinonsi racine↑.info > val alors *{recherche dans le sous-arbre gauche}*
        retour dicho (racine↑.gauche, val) ;
    sinon *{racine↑.info < val, recherche dans le sous-arbre droit}*
        retour dicho (racine↑.droit, val) ;
    finsi ;
finfonc ;

Insertion dans un ABO

- Insérer **elem** dans un arbre binaire ordonné de telle sorte qu'il demeure ordonné
- Si racine = nil (arbre vide) il suffit de créer un nœud d'adresse racine, contenant **elem**
- Sinon, on parcourt l'arbre, afin de rechercher un nœud qui sera le père de l'élément que nous désirons insérer
  - Ce nœud père est tel que :

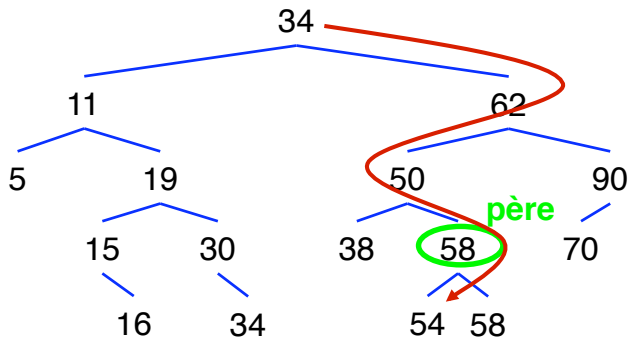
    (père↑.info ≤ elem, père↑.droit = nil)
    v
    (père↑.info > elem, père↑.gauche = nil)
- Père est toujours au bout de la filiation : c'est soit une feuille soit un nœud qui n'a qu'un seul sous-arbre.

Exemple



- Insérer 16

Exemple



- Insérer 54

Insertion dans un ABO

- L'insertion nécessite de créer une feuille

procédure créfeuille (d elem : t ; dr feuille : pointeur) ;
spécification *{}* ➔ *{création d'un nœud d'adresse feuille contenant l'information elem}*
debproc
    nouveau (feuille) ;
    feuille↑.info := elem ;
    feuille↑.gauche := nil ;
    feuille↑.droite := nil ;
finproc ;

- **Note** : lors de l'appel, **feuille** vaut nil et est le pointeur ↑.gauche ou ↑.droit du père de **feuille**

Insertion dans un ABO

- Schéma récursif

➤ racine⁺ = <> ⇨ créfeuille (elem, racine) ; ✳
➤ racine⁺ ≠ <> ⇨

➤➤ racine↑.info ≤ val ⇨
insère(racine↑.droit, elem) ; ✳

➤➤ racine↑.info > val ⇨
insère(racine↑.gauche, elem) ; ✳

Insertion dans un ABO

procédure insère (dr racine : pointeur ; d elem : t) : pointeur ;

spécification *{racine⁺ordonné}* ➔ *{elem a été inséré dans racine⁺, racine⁺ ordonné}*

debproc

si racine = nil alors

créfeuille (elem, racine) ;

sinonsi elem ≥ racine↑.info alors

insère (racine↑.droite, elem) ;

sinon *{elem < racine↑.info}*

insère (racine↑.gauche, elem) ;

finsi ;

finproc ;

Tri d'un vecteur via un ABO

- Rappel
  - la suite des valeurs dans l'ordre d'un parcours infixé est ordonnée
- Donc pour trié un vecteur
  - on peut ranger les valeurs qu'il contient dans un ABO
  - puis faire un parcours infixé de celui-ci en rangeant les valeurs dans le vecteur

Tri d'un vecteur via un ABO

procédure tri (dr V[1..n] : vecteur) ;
spécification *{n > 0}* ➔ *{V[1..n] trié}*
    i : entier ;
    racine : pointeur ;
debproc
    créarbre (V[1..n], racine) ;
    i := 1 ;
    crévect (racine, V[1..n], i) ;
finproc ;

procédure créarbre

- Initialiser l'arbre à **nil**, puis insérer dans l'arbre ordonné tous les éléments du vecteur (dans un ordre quelconque)

procédure créarbre (d V[1..n] : vecteur ; r racine : pointeur) ;
spécification *{n > 0}* ➔ *{racine⁺ ordonné contenant les valeurs de V}*
    i : entier ;
debproc
    racine := nil ;
    pour i := 1 haut n faire
        insère (racine, V[i]) ;
    finfaire ;
finproc ;

procédure crévect

- Forme récursive, parcours infixé

procédure crévect (d racine :pointeur ; r V[1..n] : vecteur ; dr i : entier) ;
spécification *{racine⁺ ordonné}* ➔ *{V[1..n] trié avec les élts de racine⁺}*
debproc

```
si racine ≠ nil alors
    crévect(racine↑.gauche,V[1..n],i) ;
    V[i] := racine↑.info ;
    i := i + 1 ;
    crévect (racine↑.droite,V[1..n],i) ;
finsi ;
```

finproc ;

Traitement de la racine :
- **écriture à V[i]**
- **préparer l'écriture à la place i+1**

- Le paramètre i doit être passé en "**dr**"
- S'il était passé en "**d**", on rangerait un nœud et son fils gauche au même emplacement dans V