Algorithmique
Les arbres

Florent Hivert

Mél : Florent.Hivert@lri.fr
Page personnelle : http://www.lri.fr/~hivert

Algorithmes et structures de données

La plupart des bons algorithmes fonctionnent grâce à une méthode astucieuse pour organiser les données. Nous allons étudier quatre grandes classes de structures de données :

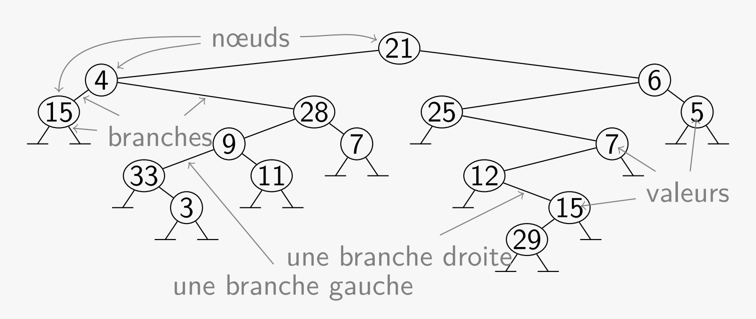
- Les structures de données séquentielles (tableaux) ;
- Les structures de données linéaires (liste chaînées) ;
- **Les arbres** ;
- Les graphes.

Problème de la recherche

On aimerai avoir une structure de donnée où l'insertion et la recherche sont efficace.

- Pour les tableaux : insertion en $O(n)$, recherche en $O(\log(n))$
- Pour les listes : insertion en $O(1)$, recherche en $O(n)$

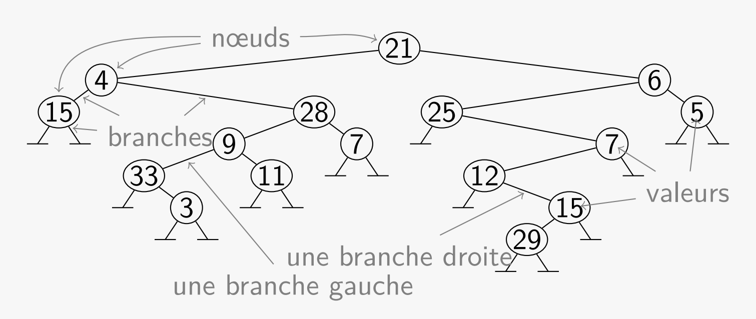
Représentations graphiques d'arbres binaires et vocabulaire



Ici :

- arbre, nœuds, branches ;
- arbre binaire, branches gauches, branches droites ;
- valeurs (ou étiquettes) des nœuds.

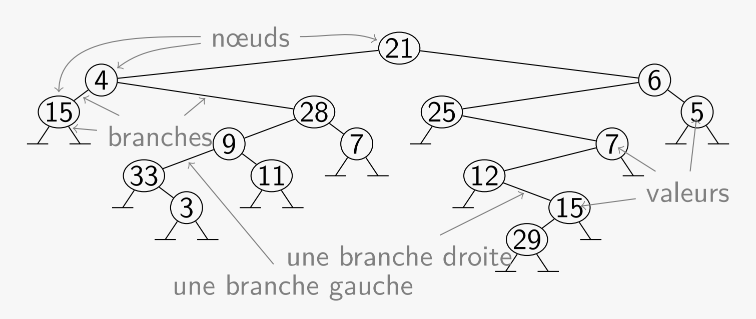
Représentations graphiques d'arbres binaires et vocabulaire



Ici :

- arbre, nœuds, branches ;
- arbre binaire, branches gauches, branches droites ;
- valeurs (ou étiquettes) des nœuds.

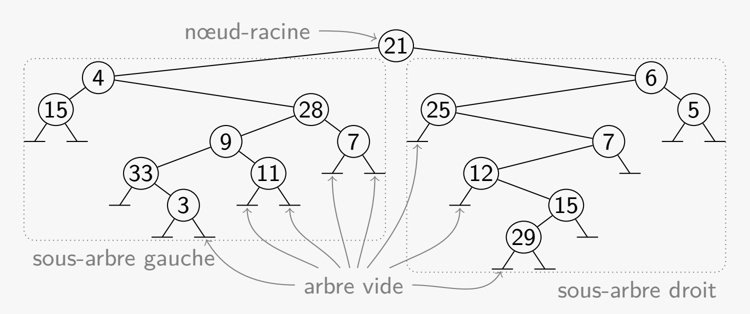
Représentations graphiques d'arbres binaires et vocabulaire



Ici :

- arbre, nœuds, branches ;
- arbre binaire, branches gauches, branches droites ;
- valeurs (ou étiquettes) des nœuds.

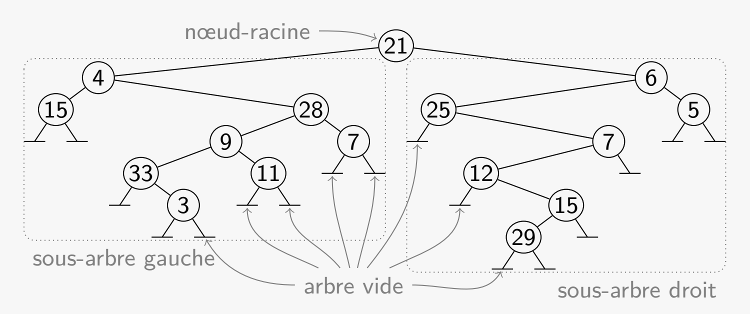
Définition récursive



Ici :

- (nœud-)racine, sous-arbre gauche, sous-arbre droit ;
- l'arbre vide, notion récursive d'arbre binaire valué (ou étiqueté) ;
- notion récursive de sous-arbre.

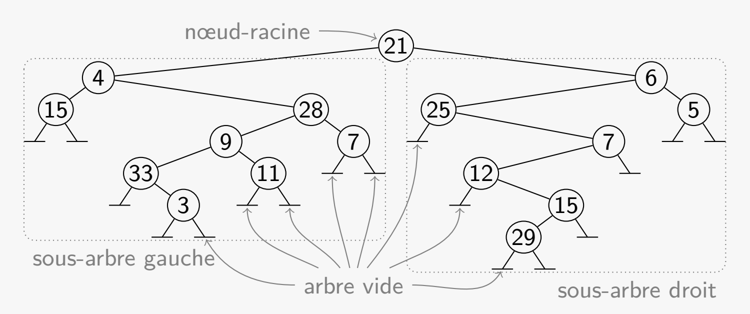
Définition récursive



Ici :

- (nœud-)racine, sous-arbre gauche, sous-arbre droit ;
- l'arbre vide, notion récursive d'arbre binaire valué (ou étiqueté) ;
- notion récursive de sous-arbre.

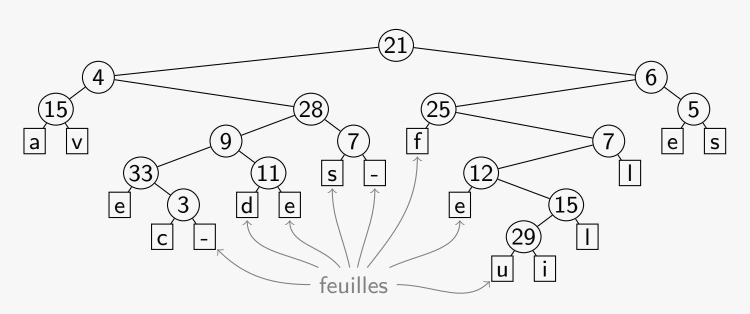
Définition récursive



Ici :

- (nœud-)racine, sous-arbre gauche, sous-arbre droit ;
- l'arbre vide, notion récursive d'arbre binaire valué (ou étiqueté) ;
- notion récursive de sous-arbre.

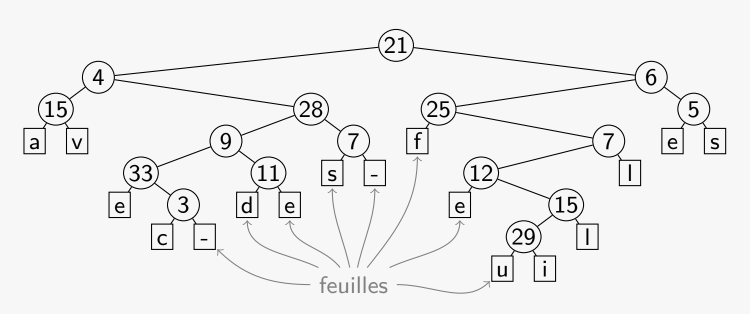
Arbres binaires étendus



Ici :

- feuilles ;
- notion récursive d'arbre binaire étendu.

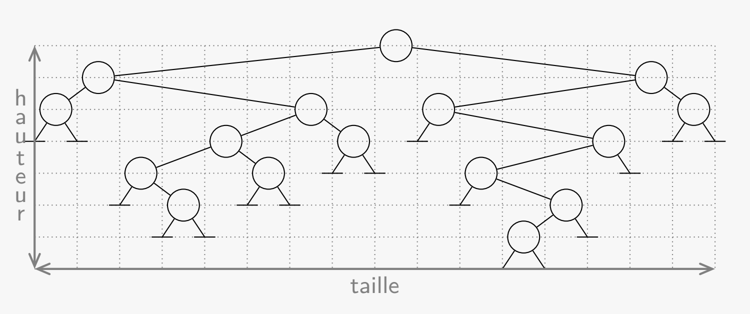
Arbres binaires étendus



Ici :

- feuilles ;
- notion récursive d'arbre binaire étendu.

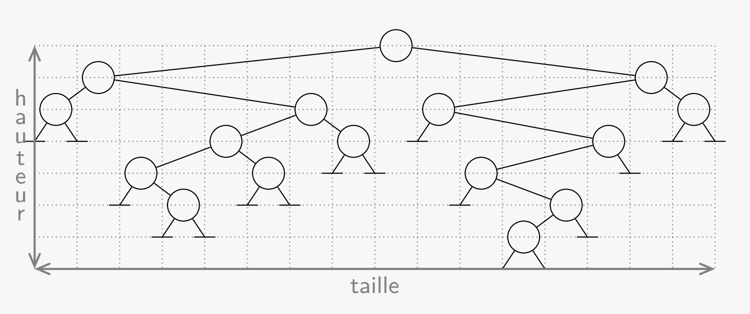
Vocabulaire



Ici :

- structure d'arbre binaire ;
- dimensions : taille, hauteur ;
- équilibre ;
- chemin issu de la racine, longueur d'un chemin.

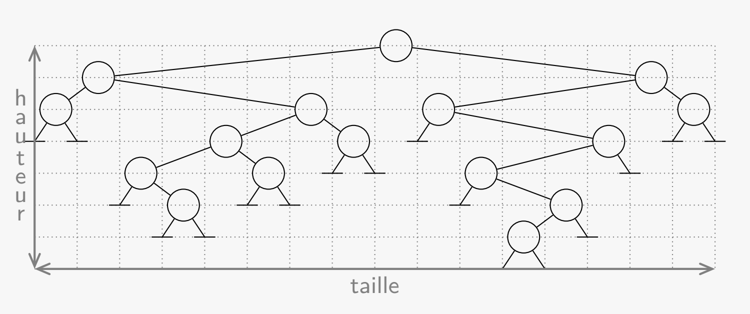
Vocabulaire



Ici :

- structure d'arbre binaire ;
- dimensions : taille, hauteur ;
- équilibre ;
- chemin issu de la racine, longueur d'un chemin.

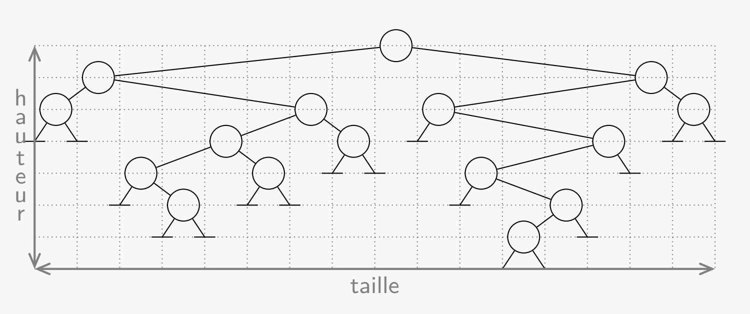
Vocabulaire



Ici :

- structure d'arbre binaire ;
- dimensions : taille, hauteur ;
- équilibre ;
- chemin issu de la racine, longueur d'un chemin.

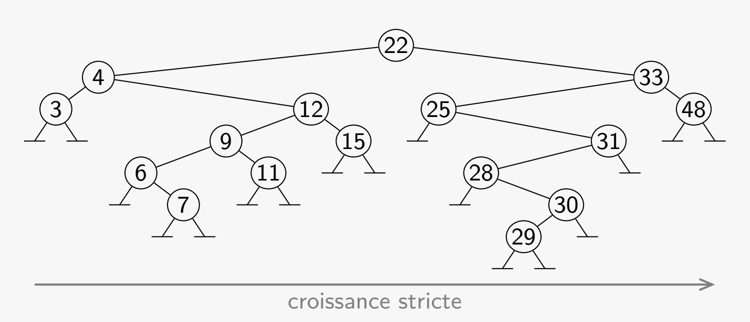
Vocabulaire



Ici :

- structure d'arbre binaire ;
- dimensions : taille, hauteur ;
- équilibre ;
- chemin issu de la racine, longueur d'un chemin.

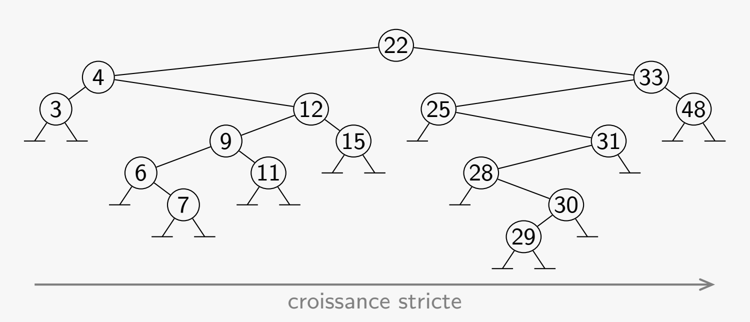
Arbre binaire de recherche



Ici :

- arbre binaire de recherche (ou ordonné) ;
- parcours infixe (ou symétrique) ;
- recherche, insertion, suppression.

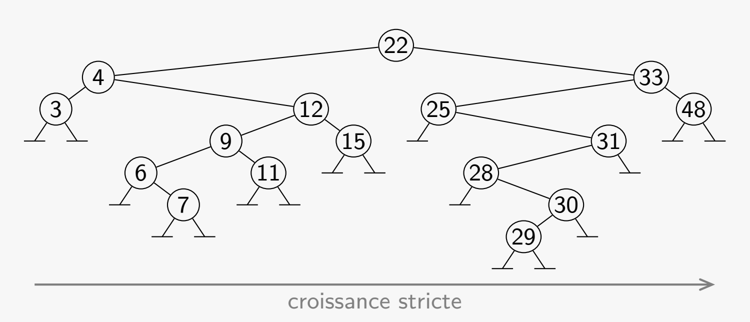
Arbre binaire de recherche



Ici :

- arbre binaire de recherche (ou ordonné) ;
- parcours infixe (ou symétrique) ;
- recherche, insertion, suppression.

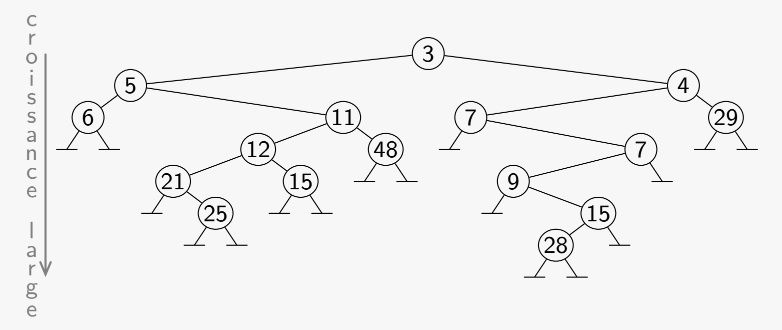
Arbre binaire de recherche



Ici :

- arbre binaire de recherche (ou ordonné) ;
- parcours infixe (ou symétrique) ;
- recherche, insertion, suppression.

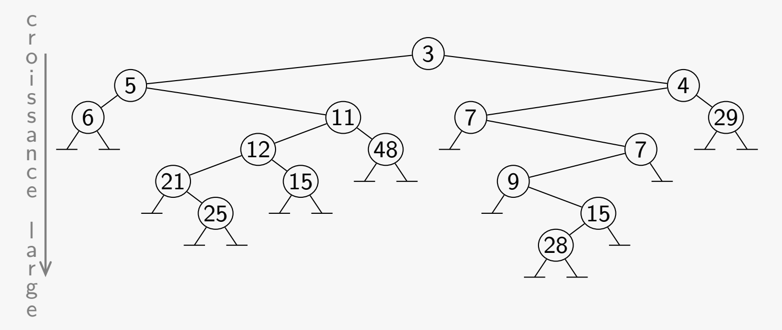
Arbre Tournoi



Ici :

- arbre tournoi ;
- minimum, insertion, suppression du minimum.

Arbre Tournoi



Ici :

- arbre tournoi ;
- minimum, insertion, suppression du minimum.

Termes anglo-saxons

- *binary tree* ;
- *node*, *branch*, *value*, *label*, *root*, *subtree*, *leaf* ;
- *size*, *height*, *distance* ;
- *balanced tree* ;
- *path from the root*, *length of a path* ;
- *infix traversal* ;
- *valued binary tree*, *label(l)ed binary tree*, *extended binary tree*, *binary search tree*, *ordered binary tree*, *tournament tree*.

Applications des arbres

- **Classifications :** par questionnaire binaire :
  - nœud = question, feuille = réponse ;
  - branche gauche étiquetée par FAUX, branche droite par VRAI.
- **Recherche :** par arbres binaires de recherche.
- **Files de priorité :** par arbres-tournoi : gestion des tampons avec priorité.

Spécification formelle

Définition (**Type abstrait** ABin)

Opérations :

- Vide : $\{\} \to$ ABin
- Noeud : ABin $\times$ ABin $\to$ ABin
- EstVide : ABin $\to$ Booleen
- $\mathrm{SAG}, \mathrm{SAD}$: ABin $\to$ ABin

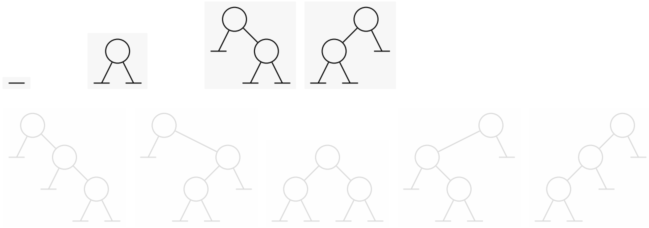
Préconditions :

- $\mathrm{SAD}(t), \mathrm{SAG}(t)$ *défini seulement si non* $\mathrm{EstVide}(t)$

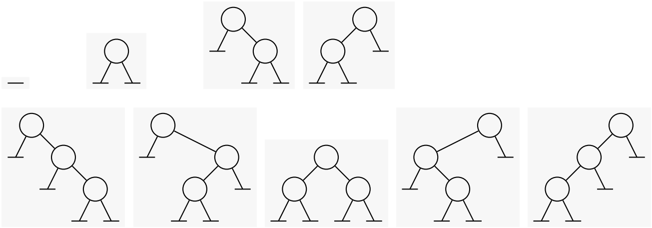
Axiomes :

- $\mathrm{EstVide}(\mathrm{Vide}()) = \mathrm{VRAI}$
- $\mathrm{EstVide}(\mathrm{Noeud}(g, d)) = \mathrm{FAUX}$
- $\mathrm{SAG}(\mathrm{Noeud}(g, d)) = g$
- $\mathrm{SAD}(\mathrm{Noeud}(g, d)) = d$
- $\mathrm{Noeud}(\mathrm{SAG}(t), \mathrm{SAD}(t)) = t$ *si non* $\mathrm{EstVide}(t)$.

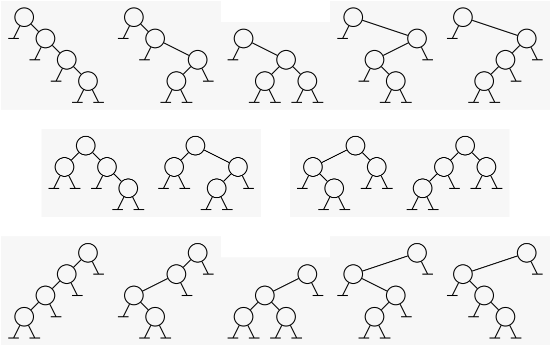
Voici la liste de tous les arbres jusqu'à la taille 3 :



Voici la liste de tous les arbres jusqu'à la taille 3 :



Voici la liste de tous les arbres de taille 4 :



Liste de tous les arbres à *n* Nœuds

Algorithme

- ***Entrée :*** *un entier positif ou nul* n
- ***Sortie :*** *une liste d'arbres*

```
res <- listeVide()
si n = 0 alors
    ajoute(res, arbreVide())
    retourner res
pour i de 0 à n-1 faire
    lg <- ALGO(i); ld <- ALGO(n-1-i)
    pour g dans lg faire
        pour d dans ld faire
            ajoute(res, Noeud(g,d))
retourner res
```

Nombre de Catalan

Proposition

Le nombre d'arbres binaires à n nœuds est appelé n-ième nombre de Catalan noté C_n. Les nombre de Catalan vérifient la récurrence :

$$C_0 = 1 \qquad C_n = \sum_{i=0}^{n-1} C_i C_{n-1-i} \,.$$

On en déduit

$$C_n = \frac{(2n)!}{n!(n+1)!} \,.$$

Voici les premières valeurs :

$$C_0 = 1,\ C_1 = 1,\ C_2 = 2,\ C_3 = 5,\ C_4 = 14,\ C_5 = 42,\ c_6 = 132 \,.$$

taille et hauteur

Définition

On définit deux fonctions sur les arbres binaires :
Le nombre de noeuds appelé Taille *:*

- $\text{Taille}(\text{Vide}) = 0$
- $\text{Taille}(\text{Noeud}(a_0, a_1)) = 1 + \text{Taille}(a_0) + \text{Taille}(a_1)$

Le nombre de noeuds du plus long chemin appelé Hauteur *:*

- $\text{Hauteur}(\text{Vide}) = 0$
- $\text{Hauteur}(\text{Noeud}(a_0, a_1)) = 1 + \max\{\text{Hauteur}(a_0), \text{Hauteur}(a_1)\}$

Comparaison taille/hauteur

Proposition

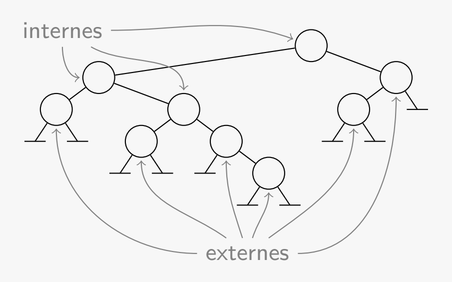
Pour tout arbre binaire de taille n et de hauteur h :

$$h \leqslant n \leqslant 2^h - 1 \,.$$

Noeuds

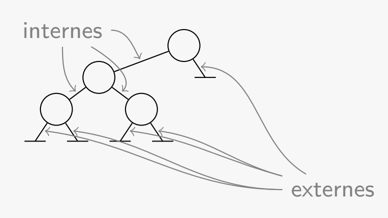
Retenir

*Un noeud est dit **interne** s'il a deux fils non vide. Sinon il est dit **externe**.*



Retenir

*Une **branche** relie un noeuds à l'un des deux sous-arbres. Une **branche** est soit la branche gauche soit la branche droite d'un nœud.*
*Une branche est **interne** lorsqu'elle relie deux nœuds ; elle est **externe** dans le cas contraire.*



En conséquence de quoi :

- un nœud interne possède deux branches internes ;
- un nœud externe possède au moins une branche externe.

Nombre de branches

Proposition

Tout arbre binaire de n nœuds possède $2n$ branches.

Plus précisément, lorsque $n \geqslant 1$, il possède $n-1$ branches internes et $n+1$ branches externes.

Retenir

*Un **chemin** de longueur k issu de a est un couple de la forme :*

$$(a, \langle b_1, b_2, \ldots, b_k \rangle)$$

pour lequel il existe $t_1, t_2, \ldots, t_k$ tels que :

> *en posant $t_0 = a$, t_j est le sous-arbre gauche ou droit de a'_{j-1} selon que le **bit de direction** b_j vaut 0 ou 1.*

On dit d'un tel chemin qu'il mène de a à t_k.

Le chemin de longueur nulle $(a, \langle\rangle)$ mène de a à lui-même.

chemin $(a, \langle 0, 1, 0, 0 \rangle)$

Un chemin est **interne** lorsqu'il mène à un nœud ; il est **externe** sinon.

Retenir

*Un **chemin** de longueur k issu de a est un couple de la forme :*

$$(a, \langle b_1, b_2, \ldots, b_k \rangle)$$

pour lequel il existe $t_1, t_2, \ldots, t_k$ tels que :

*en posant $t_0 = a$, t_j est le sous-arbre gauche ou droit de a'_{j-1} selon que le **bit de direction** b_j vaut 0 ou 1.*

On dit d'un tel chemin qu'il mène de a à t_k.

Le chemin de longueur nulle $(a, \langle\rangle)$ mène de a à lui-même.

chemin $(a, \langle 0, 1, 0, 0 \rangle)$

Un chemin est **interne** lorsqu'il mène à un nœud ; il est **externe** sinon.

Retenir

*Un **chemin** de longueur k issu de a est un couple de la forme :*

$$(a, \langle b_1, b_2, \ldots, b_k \rangle)$$

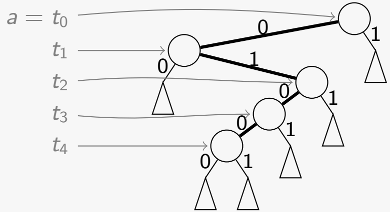
pour lequel il existe $t_1, t_2, \ldots, t_k$ tels que :

*en posant $t_0 = a$, t_j est le sous-arbre gauche ou droit de a'_{j-1} selon que le **bit de direction** b_j vaut 0 ou 1.*

On dit d'un tel chemin qu'il mène de a à t_k.

Le chemin de longueur nulle $(a, \langle\rangle)$ mène de a à lui-même.

chemin $(a, \langle 0, 1, 0, 0 \rangle)$



Un chemin est **interne** lorsqu'il mène à un nœud ; il est **externe** sinon.

Retenir

*Un **chemin** de longueur k issu de a est un couple de la forme :*

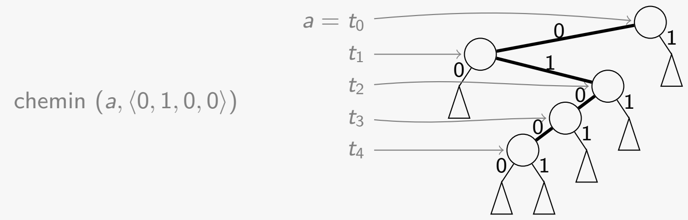
$$(a, \langle b_1, b_2, \ldots, b_k \rangle)$$

pour lequel il existe $t_1, t_2, \ldots, t_k$ tels que :

*en posant $t_0 = a$, t_j est le sous-arbre gauche ou droit de a'_{j-1} selon que le **bit de direction** b_j vaut 0 ou 1.*

On dit d'un tel chemin qu'il mène de a à t_k.

Le chemin de longueur nulle $(a, \langle\rangle)$ mène de a à lui-même.



Un chemin est **interne** lorsqu'il mène à un nœud ; il est **externe** sinon.

Proposition

Pour tout nœud a' d'un arbre binaire non vide a, il existe un unique chemin menant de la racine a de l'arbre au nœud a'.

Proposition

La hauteur d'un arbre binaire a est la longueur du plus long chemin issu de la racine a.

Proposition

Tout arbre binaire de n nœuds possède $2n + 1$ chemins distincts issus de sa racine. Parmi ceux-là, n sont internes et $n + 1$ sont externes.

Définition

Soit $(a, \langle b_1, b_2, \ldots, b_k \rangle)$ le chemin menant de a à a'.

*Le **numéro** de a' relativement à a, noté $Num_a(a')$, est $[1 b_1 b_2 \ldots b_k]_2$, l'entier dont l'écriture en base 2 est $1\, b_1\, b_2\, \ldots\, b_k$.*

Autrement dit :

- racine : 1 ;
- vers la gauche : $\times 2$, $+0$;
- vers la droite : $\times 2$, $+1$.

Définition

Soit $(a, \langle b_1, b_2, \dots, b_k\rangle)$ le chemin menant de a à a'.

*Le **numéro** de a' relativement à a, noté $Num_a(a')$, est $[1 b_1 b_2 \dots b_k]_2$, l'entier dont l'écriture en base 2 est $1\, b_1\, b_2\, \dots\, b_k$.*

Autrement dit :

- racine : 1 ;
- vers la gauche : $\times 2$, $+0$;
- vers la droite : $\times 2$, $+1$.

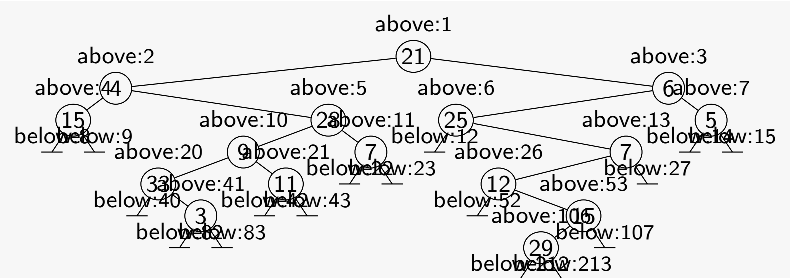
Définition

Soit $(a, \langle b_1, b_2, \ldots, b_k \rangle)$ le chemin menant de a à a'.

*Le **numéro** de a' relativement à a, noté $Num_a(a')$, est $[1 b_1 b_2 \ldots b_k]_2$, l'entier dont l'écriture en base 2 est $1\ b_1\ b_2\ \ldots\ b_k$.*

Autrement dit :

- racine : 1 ;
- vers la gauche : $\times 2$, $+0$;
- vers la droite : $\times 2$, $+1$.



Notion de parcours

Retenir

- *Un **parcours** est un algorithme qui appelle une fonction, méthode où procédure sur tous les noeuds (ou les sous arbres) d'un arbre.*
- *L'**ordre** sur les nœuds dans lequel la procédure est appelée doit être fixé. Il y a de nombreux choix possibles.*

Exemple de fonctions : affichage, liste des valeurs, accumulation. . .

Retenir

*Un **parcours** est dit **en profondeur** lorsque, systématiquement, si l'arbre n'est pas vide, le parcours de l'un des deux sous-arbres est terminé avant que ne commence celui de l'autre.*

Notion de parcours

Retenir

- *Un **parcours** est un algorithme qui appelle une fonction, méthode où procédure sur tous les noeuds (ou les sous arbres) d'un arbre.*
- *L'**ordre** sur les nœuds dans lequel la procédure est appelée doit être fixé. Il y a de nombreux choix possibles.*

Exemple de fonctions : affichage, liste des valeurs, accumulation. . .

Retenir

*Un **parcours** est dit **en profondeur** lorsque, systématiquement, si l'arbre n'est pas vide, le parcours de l'un des deux sous-arbres est terminé avant que ne commence celui de l'autre.*

Notion de parcours

Retenir

- *Un **parcours** est un algorithme qui appelle une fonction, méthode où procédure sur tous les noeuds (ou les sous arbres) d'un arbre.*
- *L'**ordre** sur les nœuds dans lequel la procédure est appelée doit être fixé. Il y a de nombreux choix possibles.*

Exemple de fonctions : affichage, liste des valeurs, accumulation. . .

Retenir

*Un **parcours** est dit **en profondeur** lorsque, systématiquement, si l'arbre n'est pas vide, le parcours de l'un des deux sous-arbres est terminé avant que ne commence celui de l'autre.*

Parcours préfixe, infixe, postfixe

Parcours en profondeur de gauche à droite (on applique F sur tous les sous-arbres) :

- **Préfixe** :
  1. application de F à la racine,
  2. parcours préfixe du sous-arbre gauche,
  3. parcours préfixe du sous-arbre droit.
- **Infixe** (ou **symétrique**) :
  1. parcours infixe du sous-arbre gauche,
  2. application de F à la racine,
  3. parcours infixe du sous-arbre droit.
- **Postfixe** :
  1. parcours postfixe du sous-arbre gauche,
  2. parcours postfixe du sous-arbre droit,
  3. application de F à la racine.

Les parcours droite-gauche se déduisent par symétrie.

Parcours préfixe, infixe, postfixe

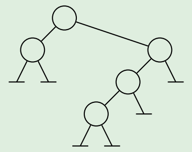
Parcours en profondeur de gauche à droite (on applique F sur tous les sous-arbres) :

- **Préfixe** :
  1. application de F à la racine,
  2. parcours préfixe du sous-arbre gauche,
  3. parcours préfixe du sous-arbre droit.
- **Infixe** (ou **symétrique**) :
  1. parcours infixe du sous-arbre gauche,
  2. application de F à la racine,
  3. parcours infixe du sous-arbre droit.
- **Postfixe** :
  1. parcours postfixe du sous-arbre gauche,
  2. parcours postfixe du sous-arbre droit,
  3. application de F à la racine.

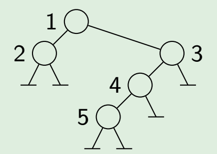
Les parcours droite-gauche se déduisent par symétrie.

Exemple

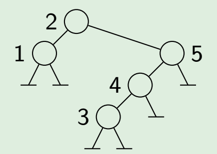
Pour l'arbre :



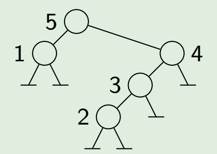
les ordres de traitement des nœuds sont, selon les parcours :



préfixe



infixe



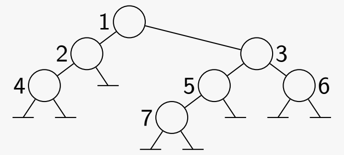
postfixe

Parcours en largeur

Retenir

Un ***parcours*** *est dit* ***en largeur*** *lorsqu'il procède en croissant selon les niveaux.*

Voici un parcourt en largeur de gauche à droite :



Algorithme de parcours en largeur

Idée : on remplace la pile d'appels par une file d'attente dans l'algorithme de parcours préfixe.

Algorithme

- ***Entrée :*** *un arbre binaire a, une procédure f*
- ***Effet :*** *appelle f sur tous les sous arbres*

```
p <- FileVide()
p <- Enfile(p, a)
tant que non EstVideFile(p) faire
    ssa, p <- Defile(p)
    si non EstVide(ssa)
       f(ssa)
       p <- Enfile(p, SAG(ssa))
       p <- Enfile(p, SAD(ssa))
```

Définition (**Type abstrait arbre binaire valué** ABinV(T))

Opérations :

- $\mathrm{Vide}_{val} : \{\} \to \mathrm{ABinV}(T)$
- $\mathrm{EstVide}_{val} : \mathrm{ABinV}(T) \to \mathrm{Booleen}$
- $\mathrm{Noeud}_{val} : T \times \mathrm{ABinV}(T) \times \mathrm{ABinV}(T) \to \mathrm{ABinV}(T)$
- $\mathrm{SAG}_{val},\ \mathrm{SAD}_{val} : \mathrm{ABinV}(T) \to \mathrm{ABinV}(T)$
- $\mathrm{Val} : \mathrm{ABinV}(T) \to T$

Préconditions :

- $\mathrm{SAD}(t), \mathrm{SAG}(t), \mathrm{Val}(t)$ *défini seulement si non* $\mathrm{EstVide}(t)$

Axiomes :

- $\mathrm{EstVide}(\mathrm{Vide}()) = \mathrm{VRAI}$
- $\mathrm{EstVide}(\mathrm{Noeud}(v, g, d)) = \mathrm{FAUX}$
- $\mathrm{SAG}(\mathrm{Noeud}(v, g, d)) = g$
- $\mathrm{SAD}(\mathrm{Noeud}(v, g, d)) = d$
- $\mathrm{Val}(\mathrm{Noeud}(v, g, d)) = v$
- $\mathrm{Noeud}(\mathrm{Val}(t), \mathrm{SAG}(t), \mathrm{SAD}(t)) = t$ *si non* $\mathrm{EstVide}(t)$.

Arbres valués et non valués

Retenir

Les définitions de taille, hauteur, chemin, interne, externe et numéro s'applique également pour les arbres binaires valués.

Définition (Forme d'une arbre binaire valués)

On défini récursivement la forme d'un binaire valués par

- Forme : $\mathrm{ABinV}(T) \to \mathrm{ABin}$;
- $\mathrm{Forme}(\mathrm{Vide}_{val}()) = \mathrm{Vide}()$;
- $\mathrm{Forme}(\mathrm{Noeud}_{val}(v, g, d)) = \mathrm{Noeud}(\mathrm{Forme}(g), \mathrm{Forme}(d))$.

Arbres binaires de recherche

Définition

*Un **arbre binaire de recherche** (ABR ; ou **arbre binaire ordonné**, ABO) est un ABV qui, s'il n'est pas vide, est tel que :*

- *ses sous-arbres gauche et droit sont des ABR ;*
- *les valeurs des nœuds du sous-arbre **gauche** sont **strictement inférieures** à la valeur du nœud-racine de l'arbre ;*
- *les valeurs des nœuds du sous-arbre **droit** sont **strictement supérieures** à la valeur du nœud-racine de l'arbre.*

Les valeurs des nœuds dans un ABR sont donc deux à deux distinctes. Autrement dit, la qualitificatif « ordonné » est à prendre au sens strict.

Arbres binaires de recherche

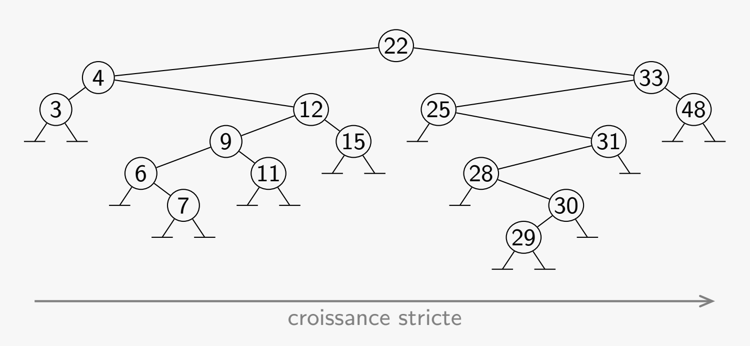
Définition

*Un **arbre binaire de recherche** (ABR ; ou **arbre binaire ordonné**, ABO) est un ABV qui, s'il n'est pas vide, est tel que :*

- *ses sous-arbres gauche et droit sont des ABR ;*
- *les valeurs des nœuds du sous-arbre **gauche** sont **strictement inférieures** à la valeur du nœud-racine de l'arbre ;*
- *les valeurs des nœuds du sous-arbre **droit** sont **strictement supérieures** à la valeur du nœud-racine de l'arbre.*

Les valeurs des nœuds dans un ABR sont donc deux à deux distinctes. Autrement dit, la qualitificatif « ordonné » est à prendre au sens strict.

Exemple d'arbre binaire de recherche



ABR et ordre infixe

Théorème (caractérisation rapide des ABR)

Un ABV est un ABR si seulement si la liste des valeurs des nœuds établie dans l'ordre infixe est strictement croissante.

Exemple

Avec $T =$ Naturel, l'ABV :

est un ABR. La liste des valeurs de ses nœuds, établie dans l'ordre infixe, est strictement croissante : $\langle 2, 4, 7, 12, 20 \rangle$.

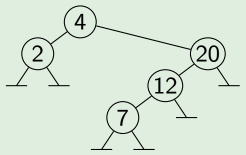
ABR et ordre infixe

Théorème (caractérisation rapide des ABR)

Un ABV est un ABR si seulement si la liste des valeurs des nœuds établie dans l'ordre infixe est strictement croissante.

Exemple

Avec $T =$ Naturel, l'ABV :



est un ABR. La liste des valeurs de ses nœuds, établie dans l'ordre infixe, est strictement croissante : $\langle 2, 4, 7, 12, 20 \rangle$.

Opérations sur les ABR

On veux implanter les opérations suivantes :

- recherche d'un élément : EstDansABR ;
- insertion d'un élément : InsertABR ;
- suppression d'un élément : SupprimeABR ;
- rotation (rééquilibrage).

Note : il y a plusieurs manières d'insérer et de supprimer un éléments. D'autre opérations plus complexes existent (fusion, partition).

Opérations sur les ABR

On veux implanter les opérations suivantes :

- recherche d'un élément : EstDansABR ;
- insertion d'un élément : InsertABR ;
- suppression d'un élément : SupprimeABR ;
- rotation (rééquilibrage).

Note : il y a plusieurs manières d'insérer et de supprimer un éléments. D'autre opérations plus complexes existent (fusion, partition).

Opérations sur les ABR

On veux implanter les opérations suivantes :

- recherche d'un élément : EstDansABR ;
- insertion d'un élément : InsertABR ;
- suppression d'un élément : SupprimeABR ;
- rotation (rééquilibrage).

Note : il y a plusieurs manières d'insérer et de supprimer un éléments. D'autre opérations plus complexes existent (fusion, partition).

Recherche d'un élément dans un ABR

Algorithme (EstDansABR)

- ***Entrée :*** *un ABR a et un élément e*
- ***Sortie :*** VRAI *si e apparaît dans a,* FAUX *sinon*

```
si EstVide(a) alors
    retourner FAUX
sinon si e = Val(a) alors
    retourner VRAI
sinon si e < Val(a) alors
    retourner EstDansABR(SAG(A))
sinon
    retourner EstDansABR(SAD(A))
```

$\Rightarrow$ *Complexité :* $O(\text{Hauteur}(a)) \subseteq O(\text{Taille}(a))$.

Insersion aux feuilles

Algorithme (InsertABR)

- ***Entrée :*** *un ABR a et un élément e*
- ***Sortie :*** *un ABR a'*

```
si EstVide(a) alors
    retourner Noeud(a, Vide(), Vide())
sinon si e = Val(a) alors
    retourner a
sinon si e < Val(a) alors
    retourner Noeud(Val(a), InsertABR(SAG(a)), SAD(a))
sinon
    retourner Noeud(Val(a), SAG(a), InsertABR(SAD(a)))
```

$\Rightarrow$ *Complexité :* $O(\text{Hauteur}(a)) \subseteq O(\text{Taille}(a))$.

Correction de InsertABR

Proposition

Soit $a \in \mathrm{ABR}(T)$ *un ABR et* $e \in T$ *un élément. Soit* $a' = \mathrm{InsertABR}(a, e)$. *Alors, pour tout* $x \in T$ *on a*

$$\mathrm{EstDansABR}(a', x) = \mathrm{EstDansABR}(a, x) \text{ ou } (e = x)\,.$$

Autrement dit,

$$\mathrm{Valeurs}(a') = \mathrm{Valeurs}(a) \cup \{e\}$$

où $\mathrm{Valeurs}(a)$ *désigne l'ensemble des valeurs qui apparaissent dans l'arbre* a.

Exemples d'insertions

Exemple

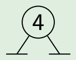
Insertions successives de 4, 20, 12, 2, 7, 3, 6, 0, 15, 1, 13, 14 dans l'ABR vide :

—

Exemples d'insertions

Exemple

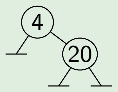
Insertions successives de 4, 20, 12, 2, 7, 3, 6, 0, 15, 1, 13, 14 dans l'ABR vide :



Exemples d'insertions

Exemple

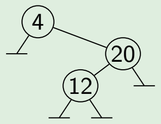
Insertions successives de 4, 20, 12, 2, 7, 3, 6, 0, 15, 1, 13, 14 dans l'ABR vide :



Exemples d'insertions

Exemple

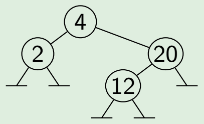
Insertions successives de 4, 20, 12, 2, 7, 3, 6, 0, 15, 1, 13, 14 dans l'ABR vide :



Exemples d'insertions

Exemple

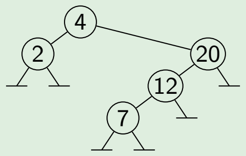
Insertions successives de 4, 20, 12, 2, 7, 3, 6, 0, 15, 1, 13, 14 dans l'ABR vide :



Exemples d'insertions

Exemple

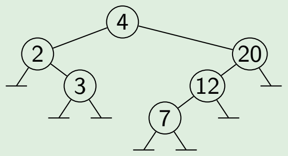
Insertions successives de 4, 20, 12, 2, 7, 3, 6, 0, 15, 1, 13, 14 dans l'ABR vide :



Exemples d'insertions

Exemple

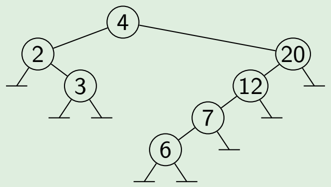
Insertions successives de 4, 20, 12, 2, 7, 3, 6, 0, 15, 1, 13, 14 dans l'ABR vide :



Exemples d'insertions

Exemple

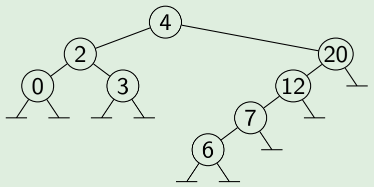
Insertions successives de 4, 20, 12, 2, 7, 3, 6, 0, 15, 1, 13, 14 dans l'ABR vide :



Exemples d'insertions

Exemple

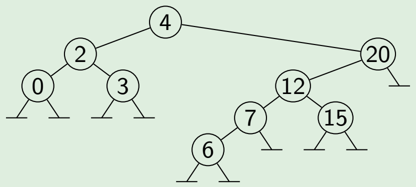
Insertions successives de 4, 20, 12, 2, 7, 3, 6, 0, 15, 1, 13, 14 dans l'ABR vide :



Exemples d'insertions

Exemple

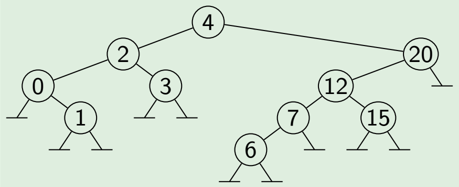
Insertions successives de 4, 20, 12, 2, 7, 3, 6, 0, $\boxed{15}$, 1, 13, 14 dans l'ABR vide :



Exemples d'insertions

Exemple

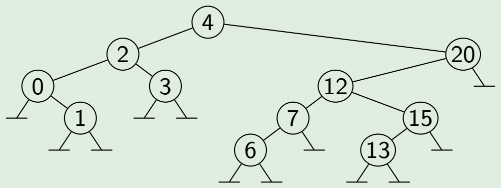
Insertions successives de 4, 20, 12, 2, 7, 3, 6, 0, 15, 1, 13, 14 dans l'ABR vide :



Exemples d'insertions

Exemple

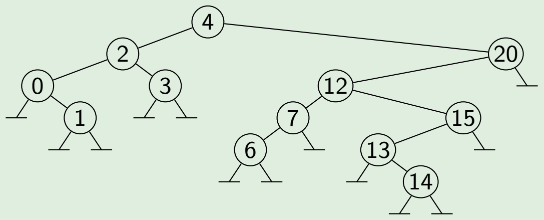
Insertions successives de 4, 20, 12, 2, 7, 3, 6, 0, 15, 1, 13, 14 dans l'ABR vide :



Exemples d'insertions

Exemple

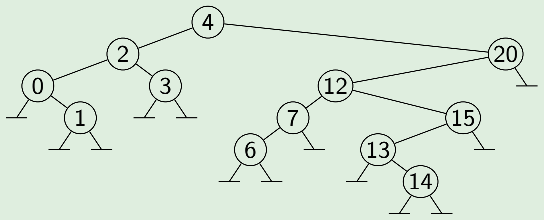
Insertions successives de 4, 20, 12, 2, 7, 3, 6, 0, 15, 1, 13, $\boxed{14}$ dans l'ABR vide :



Exemples d'insertions

Exemple

Insertions successives de 4, 20, 12, 2, 7, 3, 6, 0, 15, 1, 13, 14 dans l'ABR vide :



Bilan

On a donc une structure de donnée pour laquelle les coût de l'insertion et la recherche sont en $O(\text{Hauteur}(a))$:

Retenir

- *Dans le pire des cas (arbre filiforme), le coût est en* $O(\text{Taille}(a))$.
- *En moyenne, le coût est en* $O(\log(\text{Taille}(a)))$.

De plus, en utilisant la rotation (voir G.M. Adelson-Velskii et E.M. Landis 1962, arbre AVL, arbre rouge-noir), on peut s'assurer que Hauteur(a) reste inférieur à

$$\log_\Phi(n+2) - 1 \approx 1.44 \log_2(n+2) - 1$$

où $\Phi = \frac{1+\sqrt{5}}{2}$ est le nombre d'or et $n = \text{Taille}(a)$.

Arbres équilibrés

Définition

*L'**équilibre** d'un arbre binaire est un entier qui vaut 0 si l'arbre est vide et la différence des hauteurs des sous-arbres gauche et droit de l'arbre sinon.*

*Un arbre binaire est **équilibré** lorsque l'équilibre de chacun de ses sous-arbres non vides n'excède pas 1 en valeur absolue.*

Arbres équilibrés

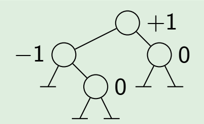
Définition

*L'**équilibre** d'un arbre binaire est un entier qui vaut 0 si l'arbre est vide et la différence des hauteurs des sous-arbres gauche et droit de l'arbre sinon.*

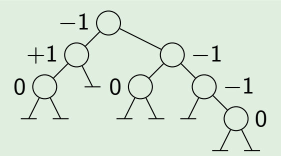
*Un arbre binaire est **équilibré** lorsque l'équilibre de chacun de ses sous-arbres non vides n'excède pas 1 en valeur absolue.*

Exemple

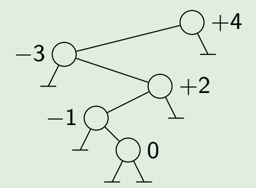
L'équilibre de chacun des sous-arbres non vides est indiqué sur la gauche ou la droite de son nœud-racine :



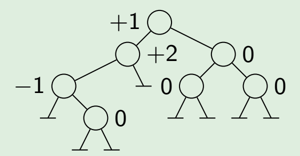
équilibré



équilibré

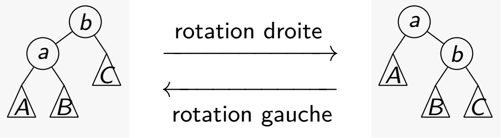


non équilibré



non équilibré

Rotations



Proposition

Après une insertion où une suppression, il suffit de deux rotations pour ré-équilibrer un arbre. Le maintient de l'équilibre est possible en temps constant.