Algorithmique et Complexité

3. Structures de données avancées
3.3 Arbres binaires de recherche et variantes

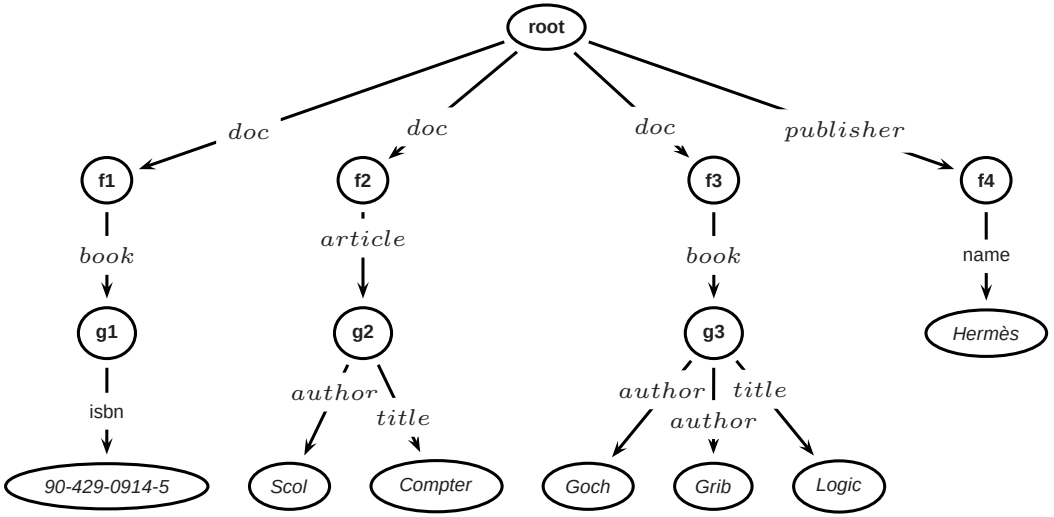
Nicole Bidoit

Université Paris XI, Orsay

Année Universitaire 2008–2009

Arbres : quelques exemples, quelques applications, quelques définitions

- noeuds organisés hiéarchiquement
- noeuds étiquetés (information contenus dans les noeuds)
- liens orientés étiquetés (information marquant les liens)



Representation d'un fichier semi-structuré (XLM) "bibliothèque"

Arbres : quelques exemples, quelques applications, quelques définitions

Exemples

arbre généalogique, arbre phylogénétique,

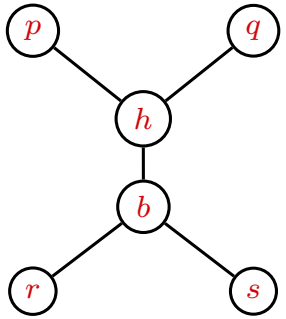
résultat d'un tournoi, contenu de page web, répertoire de fichiers,

expression arithmétique, arbre des appels récursifs ...

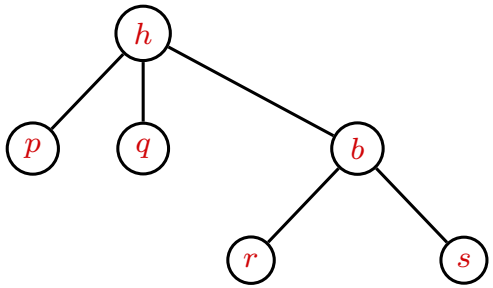
Définition **un arbre**

Soit un graphe G=(S,A) un graphe non orienté. **G est un arbre** ssi

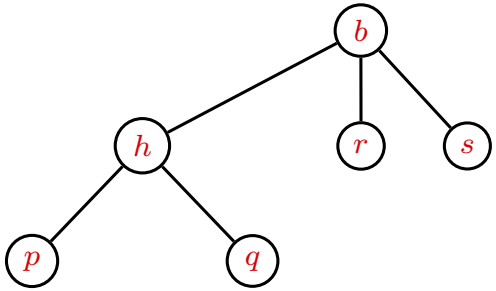
Pour tout couple de sommets de G, il existe un chemin unique reliant ces deux sommets.



arbre général



arbre enraciné en h



arbre enraciné en b

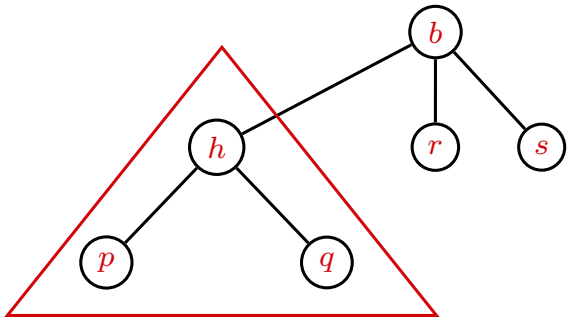
Terminologie **arbre enraciné**

arbre dont un des sommets, appelé **racine** est distingué $\Longrightarrow$ orientation de l'arbre

Exemples (ci-dessus)

arbres qui ne diffèrent que s'ils sont enracinés (ci-dessus)

Terminologie (suite)



la racine de l'arbre : b

le **père** "de" p et q : h

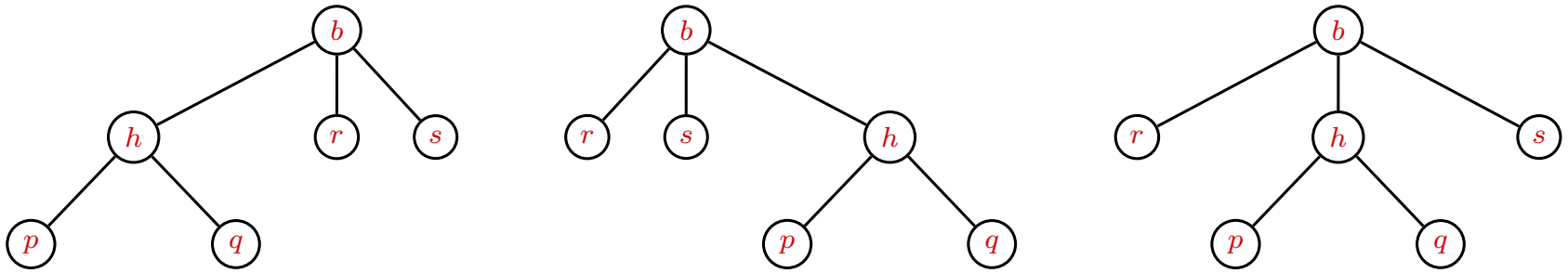
les **fils** "de" b : h , r , s

les **feuilles** de l'arbre : p , q , r , s

un **sous-arbre** : de racine h *encadré par un triangle*

une **branche** : suite de noeuds constituant un chemin de la racine à une feuille

Un **arbre ordonné** est un arbre enraciné dans lequel les fils de chaque noeud sont ordonnés entre eux (liste ordonnée de fils).



Exemple (ci-dessus)

Les trois arbres enracinés ci-dessus sont différents seulement si

ils sont ordonnés i.e l'ordre des fils sur le schéma est pris en compte.

Terminologie (suite)

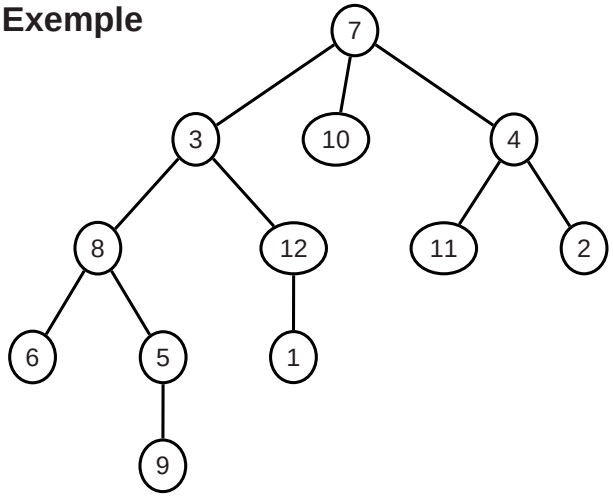
degré d'un noeud n	$deg(n)$	:	nombre de fils de n
degré d'un arbre	$deg(A)$	:	max des degrés de ses noeuds
profondeur d'un noeud n	$prof(n)$	:	longueur du chemin entre la racine et n
hauteur d'un noeud n	$h(n)$	:	1 + longueur du plus long chemin de n aux feuilles
hauteur d'un arbre	$h(A)$	:	hauteur de sa racine
taille d'un arbre	$\mid A \mid$	:	nombre de noeuds

L'arbre vide a pour hauteur 0, l'arbre réduit à un noeud (racine) a pour hauteur 1.

La racine d'un arbre est de profondeur 0, et

si un noeud est de profondeur p alors tous ses fils sont de profondeur $p+1$.

Exemple



prof. 0; $h(7) = 5$

prof. 1; $h(3) = 4, h(10) = 1, h(4) = 2$

prof. 2; $h(8) = 3, h(12) = 2, h(11) = h(2) = 1$

prof. 3; $h(6) = h(1) = 1, h(15) = 2, h(4) = 2$

prof. 4; $h(9) = 1$

Arbres : quelques exemples, ...définitions

Représentation chaînée d'un arbre ordonné

type Noeud = (contenu : elt, liste_fils : liste d' Arbre)

type Arbre = pointeur Noeud

(Illustration au tableau)

Représentation quasi-contiguë d'un arbre ordonné

tableau des noeuds de l'arbre ++

chaque place i du tableau contient les informations relatives à un noeud i.e.

le contenu du noeud et

une liste d'indices sur le tableau correspondant aux fils du noeud "stocké" en i.

(Illustration au tableau)

Codage d'un arbre ordonné quelconque par un arbre binaire

voir après présentation des arbres binaires.

Arbres binaires

Définition (récursive) **Arbre binaire**

Un arbre binaire B est une structure définie sur un ensemble fini de noeuds N

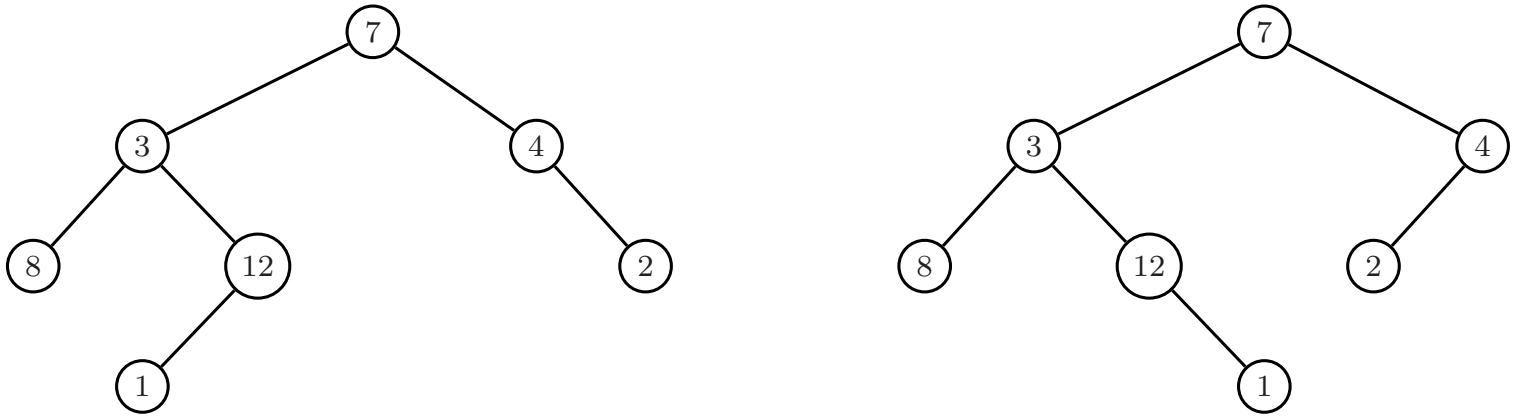
- soit $N = \emptyset$, (cas d'un arbre binaire vide)
- soit B est spécifié par r, G, D tels que $r \in N$ est la **racine** de B, et

G est un arbre binaire sur N_G, appelé le **sous-arbre gauche** de B

D est un arbre binaire sur N_D, appelé le **sous-arbre droit** de B

et $N = \{r\} \cup N_G \cup N_D$

Exemple



Ces deux arbres ordonnés sont différents seulement si ils sont considérés comme des arbres binaires.

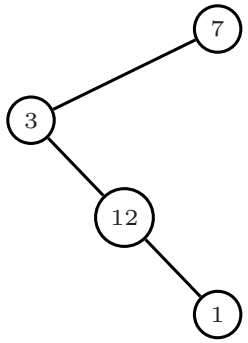
Arbres binaires particuliers

dégénéré ou **filiforme** : réduit à une branche

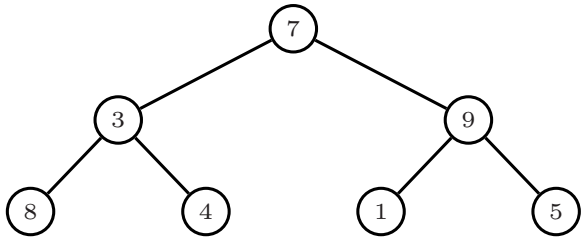
localement complet : $\forall n$ non feuille, $deg(n) = 2$

complet : localement complet et $\forall f$ feuille, $prof(f) = prof(B)$

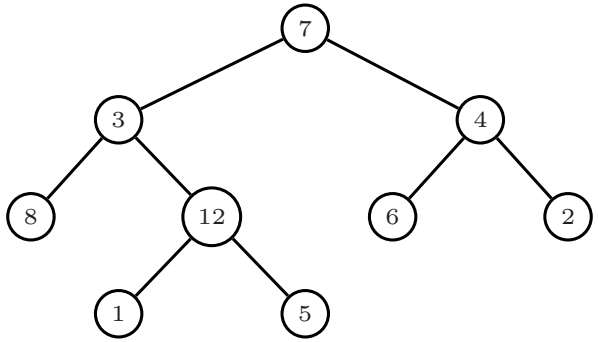
parfait : tous les niveaux ”remplis” sauf le dernier dont les feuilles sont ”tassées” à gauche



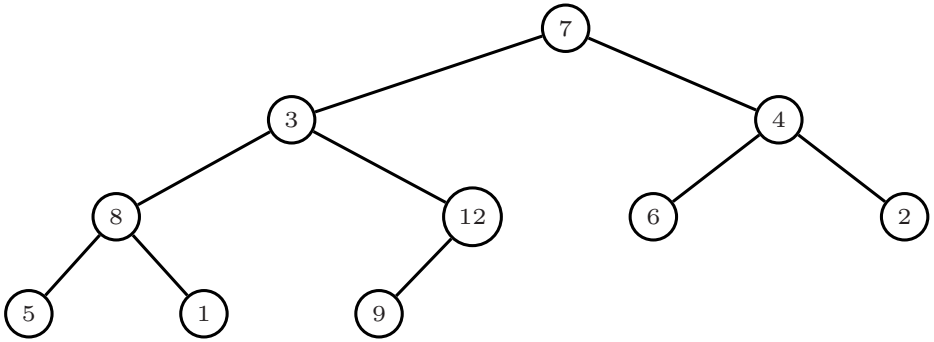
arbre dégénéré



arbre complet



arbre localement complet



arbre parfait

Arbres binaires

Taille versus hauteur, ...

Soit un arbre binaire B tel que $h(B) = h$ et $\mid B \mid = n$

Cas 1: B est un arbre dégénéré

$$h(B) = h = \mid B \mid = n$$

Cas 2: B est un arbre binaire complet

$$\mid B \mid = n = \sum_{p=0...h-1} 2^p = 2^h - 1$$

Si l'arbre binaire B est non vide alors: $\boxed{\lfloor \log_2(n+1) \rfloor \leq h \leq n}$

Représentation chaînée

type Noeud = (contenu :elt, gauche, droit : Arbin) type Arbin = pointeur Noeud

Représentation contiguë voir tri par tas

Notations : soit $B = (r, G, D)$,

Racine(B)=r Fils_Gauche(B)=G Fils_Droit(B)=D

Contenu(r) : élément contenu dans le noeud r

Les algorithmes seront donnés soit en utilisant la définition "formelle" d'un arbre binaire

soit en utilisant l'une des deux représentations ci-dessus.

Parcours en profondeur d'un arbre binaire

Parcours Préfixe

`Algo ParcoursPré`

Entrée : B un arbre binaire;

Sortie : ??

if B non vide **then**

Traite(Racine(B));

ParcoursPré(Fils_Gauche(B));

ParcoursPré(Fils_Droit(B));

$+ \times 8 / 1 5 + 6 2$

Parcours Infixe

`Algo ParcoursInf`

Entrée : B un arbre binaire;

Sortie : ??

if B non vide **then**

ParcoursInf(Fils_Gauche(B));

Traite(Racine(B));

ParcoursInf(Fils_Droit(B));

$8 \times 1 / 5 + 6 + 2$

Parcours Postfixe

`Algo ParcoursPost`

Entrée : B un arbre binaire;

Sortie : ??

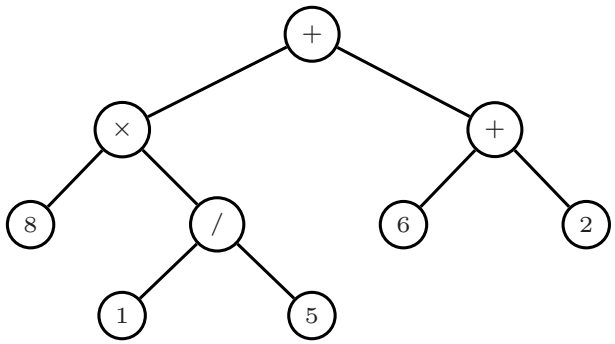
if B non vide **then**

ParcoursPost(Fils_Gauche(B));

ParcoursPost(Fils_Droit(B));

Traite(Racine(B));

$8 1 5 / \times 6 2 + +$

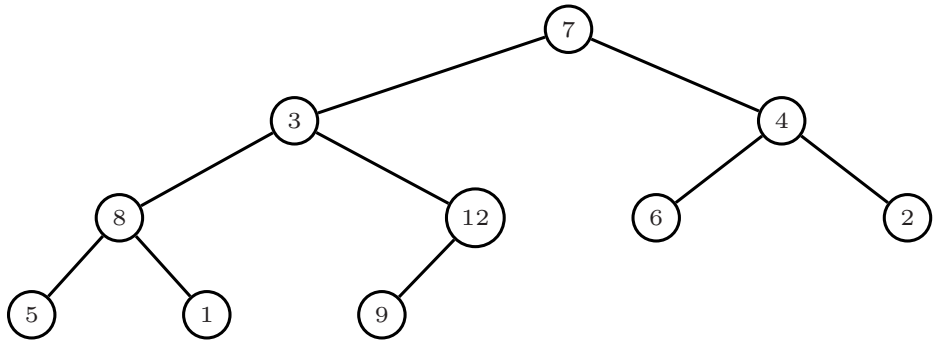


Parcours en largeur d'un arbre binaire

Tous les noeuds de profondeur p sont "traités" avant d'en faire de même avec les noeuds à profondeur $p+1$

$\Longrightarrow$ utilisation d'une file F de noeuds

```
Algo ParcoursLarg
Entrée : B : Arbin;
Auxiliaire : F file de Noeud; N Arbin
F ← nil;
if B ≠ nil then
    F ← Insert_File(B, F);
    while F ≠ nil do
        N ← Supp_File(F);
        Traite(N↑.contenu);
        F ← Insert_File(N↑.gauche, F);
        F ← Insert_File(N↑.droit, F);
    endwhile
endif
```



7 3 4 8 12 6 2 5 1 9

Exercice Version itérative des parcours en profondeur et complexité en espace.

Exercice Tous les parcours présentés pour les arbres binaires peuvent être généralisés sans problème aux arbres ordonnés généraux. Écrire les algorithmes et analyser leur complexité en espace (versions itératives).

Codage d'un arbre ordonné quelconque par un arbre binaire

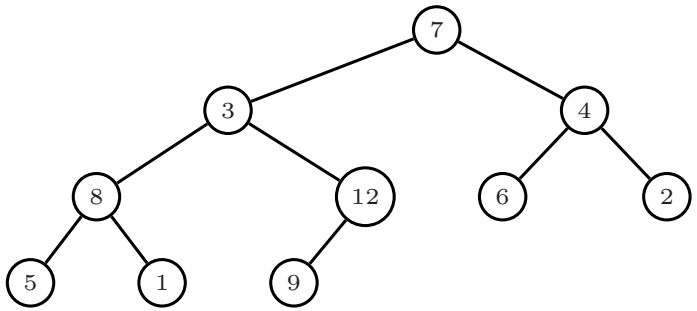
Représentation d'un arbre ordonné A par un arbre binaire B

Le sous-arbre de A enraciné en N_A est représenté par le sous-arbre de B enraciné en N_B tel que

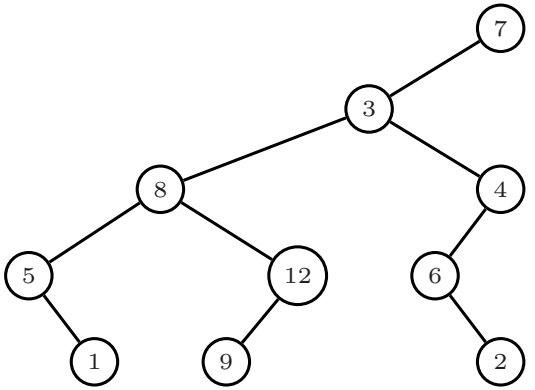
- le contenu de N_B est le contenu de N_A
- le fils gauche de N_B est le premier fils de N_A
- le fils droit de N_B est le frère (immédiatement à droite) de N_A

(Illustration au tableau)

Exemple représentation d' un arbre binaire (parfait) par un arbre binaire ...



Arbre binaire A



et sa représentation par un arbre binaire B

Arbre binaire de recherche

Motivation

Représentation d'un ensemble d'éléments pour offrir des opérations

(insertion,suppression, recherche, ...) efficaces.

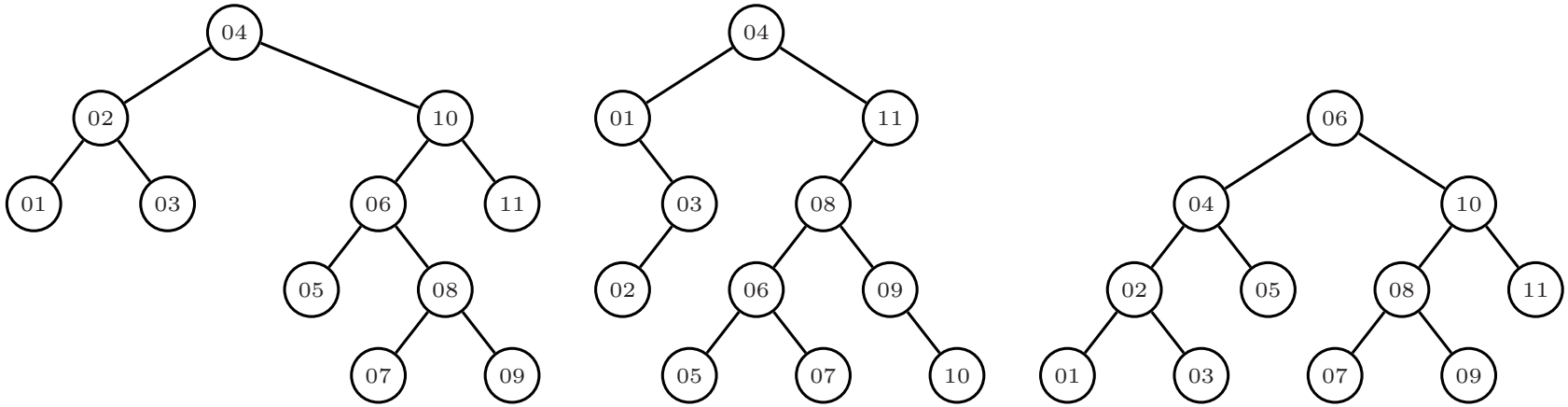
Quelle est la structure la plus efficace pour la recherche ? **tableau trié**

Grosso modo Arbre binaire = Représentation arborescente d'un tableau trié

Exemple

01	02	03	04	05	06	07	08	09	10	11

Choix de n comme racine ; les éléments plus petits que n dans le sous-arbre gauche, et ceux plus grands dans le sous arbre droit ... et ainsi de suite ... de choix ... en choix ...



Trois arbres binaires de recherche contenant les mêmes éléments

Arbre binaire de Recherche

Définition

Soit $B{=}(r, G, D)$ un arbre binaire tel que $<$ est une relation d'ordre sur les éléments contenus dans les noeuds de B.

Soit e_r l'élément contenu dans r.

B est un **arbre binaire de recherche** ssi

1. pour tout noeud N de G dont le contenu est e on a $e \leq e_r$,
2. pour tout noeud N de D dont le contenu est e on a $e_r < e$,
3. G et D sont des arbres binaires de recherche

Parcours d'un arbre binaire de recherche

Le parcours **infixe** ou **symétrique** traite les éléments de l'arbre binaire de recherche dans l'ordre croissant.

Exemple

les 3 arbres binaires de recherche du transparent précédent.

Arbre binaire de Recherche

Recherche d'un élément dans un arbre binaire de recherche

`Algo Recherche_Bin`

Entrée : B un arbre binaire de recherche;
e elt;
Sortie : Trouvé booléen;

---

if B non vide **then**
if $e = Contenu(Racine(B))$ **then** trouvé=vrai
else if $e < Contenu(Racine(B))$ **then** rouvé $\leftarrow$ Recherche_Bin(Fils_Gauche(B),e)
else trouvé $\leftarrow$ Recherche_Bin(Fils_Droit(B),e);
else trouvé $\leftarrow$ $false$

Exemple (en utilisant les 3 arbres du transparent 13.)

Complexité on compte le nombre de noeuds visités
$\Theta(h)$ où h est la hauteur de l'ABR.

Exercice : Modifier l'algorithme précédent pour proposer un algorithme qui compte le nombre d'occurrences d'un élément e (dans le cas où un arbre binaire peut contenir plusieurs fois le même élément).

Opérations fréquentes sur les arbres binaires de recherche

Recherche du plus grand élément dans un arbre binaire de recherche

`Algo Rech_Max_Bin`

Entrée : B un arbre binaire de recherche;

Sortie : max elt;

---

if B non vide **then**

  while $Fils\_Droit(B)$ non vide **do** $B \leftarrow$ Fils_Droit(B) ;

  $max \leftarrow Contenu(Racine(B))$

else $max \leftarrow special.$

Exemple (en utilisant les 3 arbres transparent 13.)

Complexité On compte le nombre de noeuds visités

$\Theta(h)$ où h est la hauteur de l'ABR B.

Exercice Recherche d'un plus petit élément dans un arbre binaire de recherche et complexité.

Opérations fréquentes ...

Recherche du prédécesseur du (contenu du) noeud N de B.

Si N a un fils gauche, alors le prédécesseur "de" N est le plus grand élément du sous arbre gauche de N

Sinon le prédécesseur "de" N est son ancêtre "droit" le plus proche.

`Algo Pred_Bin`

Entrée : B Arbin; N Arbin ;
Sortie : $Pred$ elt;
Auxiliaire : P Arbin

$Pred \leftarrow$ Special;
if $N \neq nil$ **then**
  if $N\uparrow.gauche \neq nil$ **then** $Pred \leftarrow$ Rech_Max_Bin($N\uparrow.gauche$) **else**
    $P \leftarrow N\uparrow.père$;
    while $P \neq nil$ **do if** $N = P\uparrow.gauche$ **then** $\{ N \leftarrow P ; P \leftarrow P\uparrow.père\}$;
    if $P \neq nil$ **then** $Pred \leftarrow P\uparrow.contenu$;
  endelse

Exemple au tableau

Complexité On compte le nombre de noeuds visités

$\Theta(h)$ où h est la hauteur de l'arbre B

Exercice Recherche du successeur du (contenu du) noeud N de B.

Insertion d'un élément dans un arbre binaire de recherche

L'élément est inséré dans une feuille, là où il aurait du être trouvé.

`Algo Insert_AB`

Entrée : B un arbre binaire de recherche; e elt
Sortie : B arbre binaire de recherche;
Auxiliaire : N, $Pere$ arbre binaire de recherche

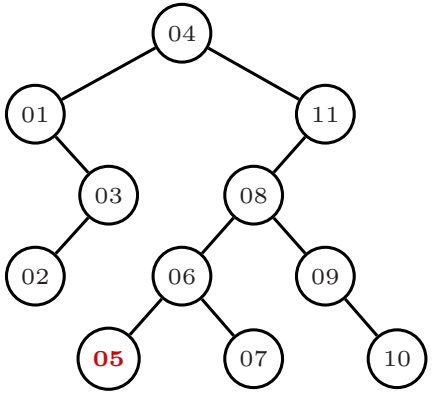
$N \leftarrow B$; $Pere \leftarrow$ vide
while N non vide **do**
  $Pere \leftarrow N$;
  if $e \leq$ Contenu(Racine(N)) **then** $N \leftarrow$ Fils_Gauche(N) **else** $N \leftarrow$ Fils_Droit(N)
endwhile
if $Pere$ vide **then** $B \leftarrow$ Créer_Feuille_Bin(e) **else**
  if $e \leq contenu(Pere)$ **then** Fils_Gauche(Pere) $\leftarrow$ Créer_Feuille_Bin(e)
  else Fils_Droit(Pere) $\leftarrow$ Créer_Feuille_Bin(e)

Exemple au tableau

Complexité On compte le nombre de noeuds visités

$\Theta(h)$ où h est la hauteur de l'arbre.

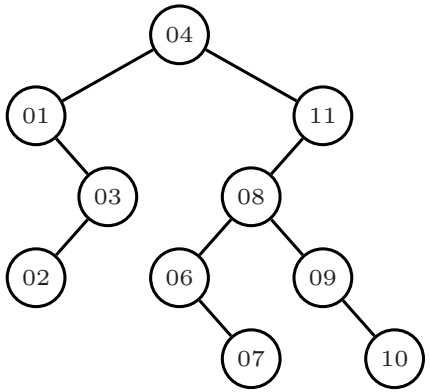
Opérations fréquentes ...Suppression d'un élément dans un arbre binaire de recherche



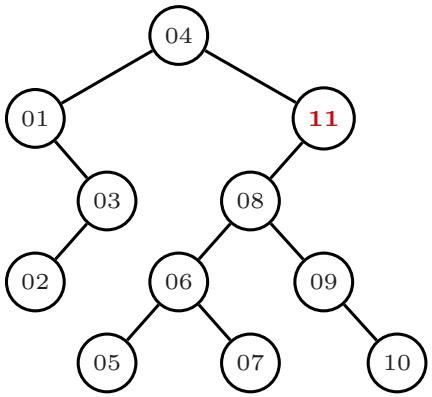
suppression de

l'élément 5

Cas "pas de fils"



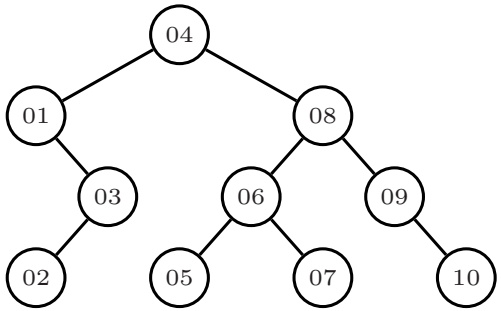
l'opération de suppression est directe, sans effet de bord



suppression de

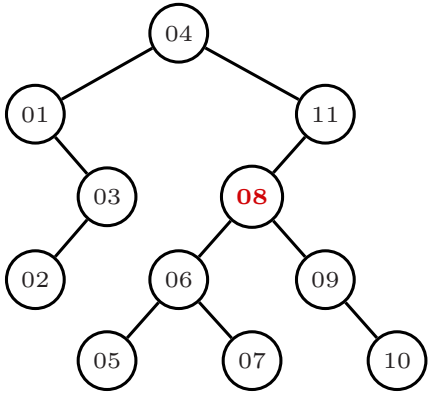
l'élément 11

Cas "un seul fils"



L'opération de suppression est effectuée par remplacement du noeud par son unique fils

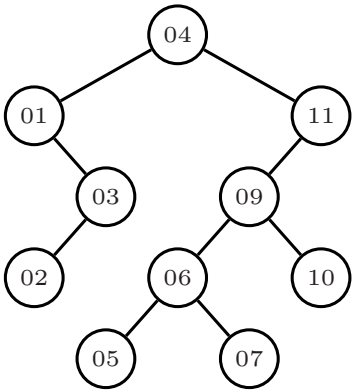
Suppression d’un élément dans un arbre binaire de recherche



suppression de

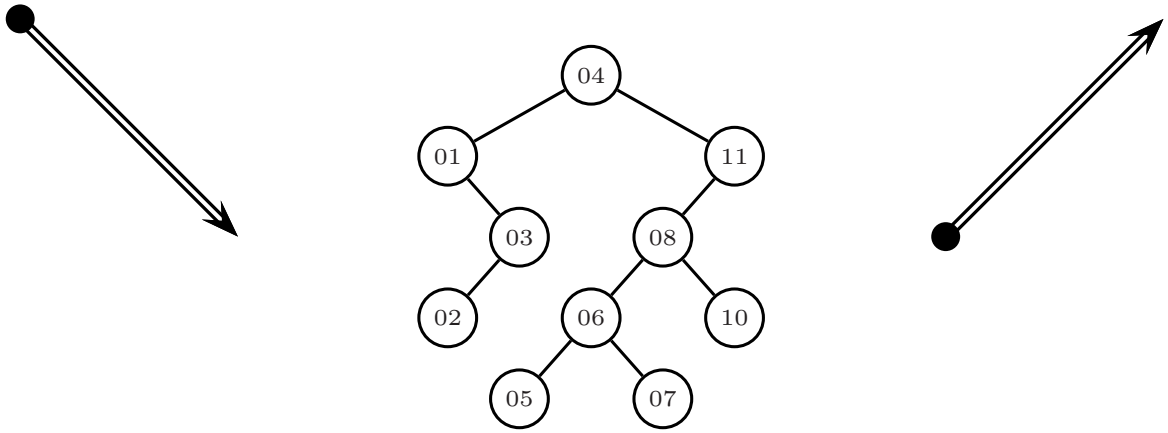
l’élément 8

Cas ”deux fils”



L’opération de suppression est effectuée par remplacement du contenu du noeud par son successeur

Le successeur doit par conséquence être supprimé



Suppression d'un élément dans un arbre binaire de recherche

`Algo Supprim_Bin`

Entrée : B Arbin ; e elt; Sortie : B Arbin;

Auxiliaire : N, Del, M noeud;

---

1 $N \leftarrow$ Recherche_Bin$^*(B,e)$;

2 **if** $N \neq nil$ **then**

3 **if** $N\uparrow.gauche = nil$ ou $N\uparrow.droit = nil$ **then** $Del \leftarrow N$ **else** $Del \leftarrow Succ\_Bin^*(B, N)$;

4 **if** $Del\uparrow.gauche \neq nil$ **then** $M \leftarrow Del\uparrow.gauche$ **else** $M \leftarrow Del\uparrow.droit$;

5 **if** $M \neq nil$ **then** $M\uparrow.père \leftarrow Del\uparrow.père$;

6 **if** $Del\uparrow.père = nil$ **then** $B \leftarrow M$

7 **else if** ($Del = Del\uparrow.père\uparrow.gauche$) **then** $Del\uparrow.père\uparrow.gauche \leftarrow M$

8 **else** $Del\uparrow.père\uparrow.droit \leftarrow M$;

9 **if** $Del \neq N$ **then** $N\uparrow.contenu \leftarrow Del\uparrow.contenu$;

endif

Complexité On compte le nombre de neouds vistés

$\Theta(h)$ où h est la hauteur de l'arbre.

arbres binaires de recherche équilibrés

Remarque Tous les algorithmes présentés ont une complexité en $\Theta(h)$ où h est la hauteur de l'arbre.

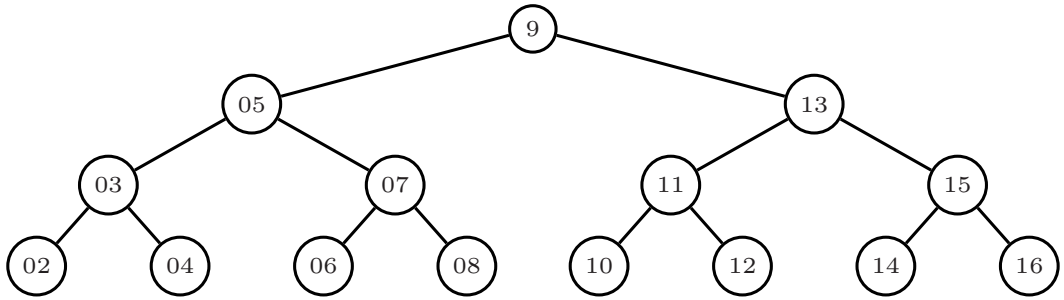
$$log_2 n \leq h \leq n$$

Maintenir un arbre parfaitement équilibré

contrainte : hauteur sous-arbre gauche = hauteur sous arbre droit

très coûteux : peut nécessiter de restructurer l'arbre totalement i.e. de "toucher" à chaque noeud.

Exemple Insérer l'élément 1 dans l'arbre binaire de recherche ci-dessous et le rééquilibrer (pour obtenir un arbre parfaitement équilibré)



Deux types de compromis possible

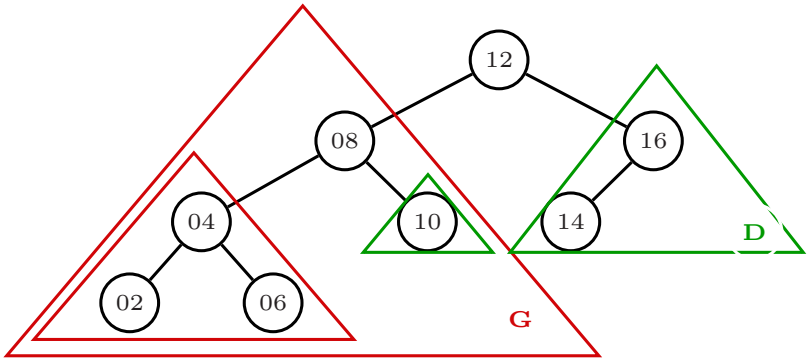
1. relacher (légèrement) la contrainte d'équilibre parfait
2. contrainte d'équilibre parfait mais autoriser plusieurs éléments dans un noeud

Arbres d'Adelson-Velsky et Landis (AVL)

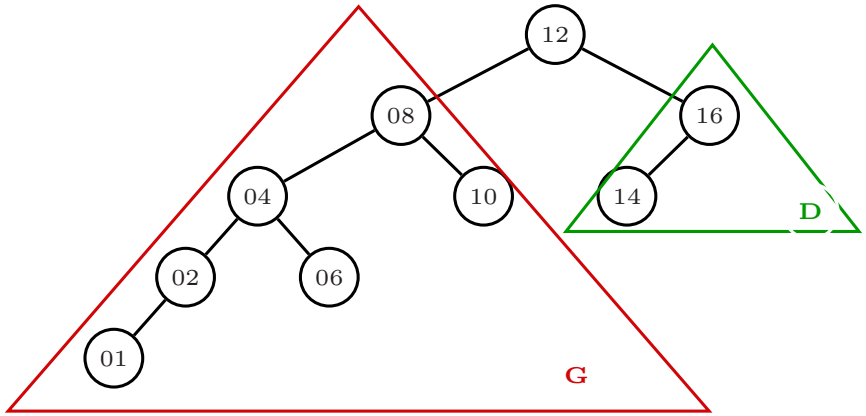
Définition Un AVL est un arbre binaire de recherche B tel que:

soit B est vide

soit $B = (r, G, D)$ et alors : G et D sont des AVLs et $\underline{\mid h(G) - h(D) \mid \leq 1}$

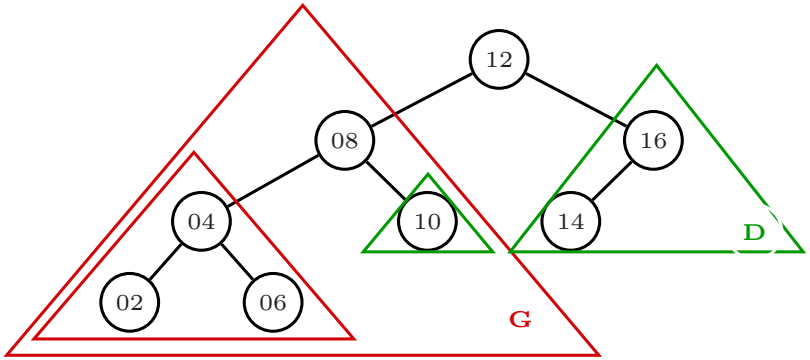


Exemple d' arbre AVL

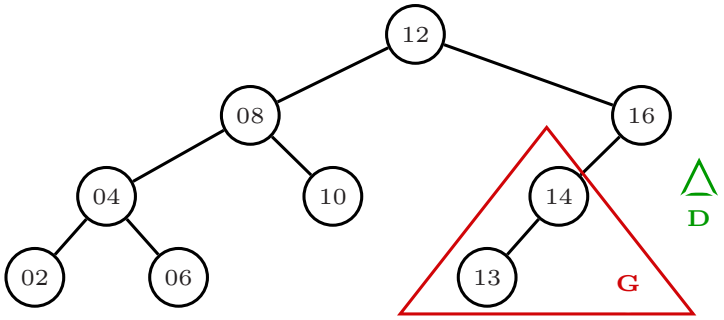


Contre-exemple d'arbre AVL (après insertion de 1)

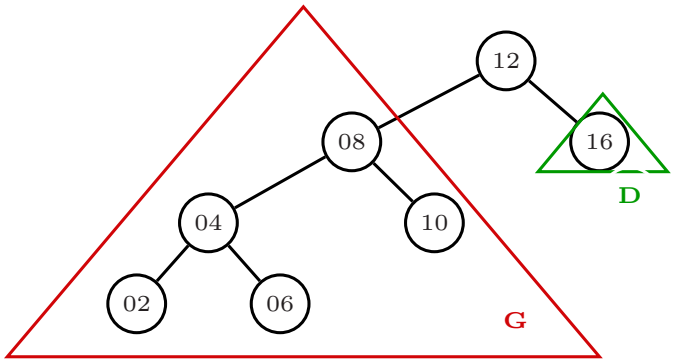
Arbre AVL



Exemple d' arbre AVL



Contre-exemple d'arbre AVL (après insertion de 13)



Contre-exemple d'arbre AVL (après suppression de 14)

insertion

suppression $\Longrightarrow$ **rééquilibrage**

Taille versus hauteur d'un AVL

Soit un AVL B de hauteur $h(b) = h$ et de taille $\mid B \mid = n$

On a $\boxed{log_2(n+1) \leq h < 1,44\, log_2(n+2)}$

Représentation chaînée d'un arbre AVL

type NAVL = (contenu : elt; **bal: [-1,1]**; gauche, droit : AVL);

type AVL = pointeur NAVL;

Le coefficient d'équilibrage (**la balance**) d'un sous-arbre B enraciné en N d'un arbre AVL est défini comme la différence

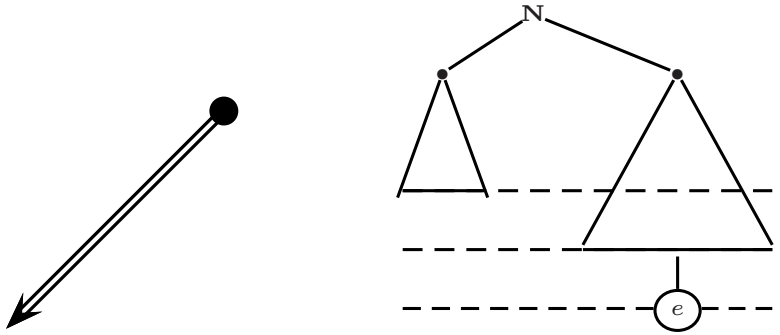
$h(Fils\_Gauche(B)) - h(Fils\_Droit(B))$, noté **bal(**$B$**)**, ou encore

$h(N.gauche) - h(N.droit)$ aussi noté **bal(**N**)**

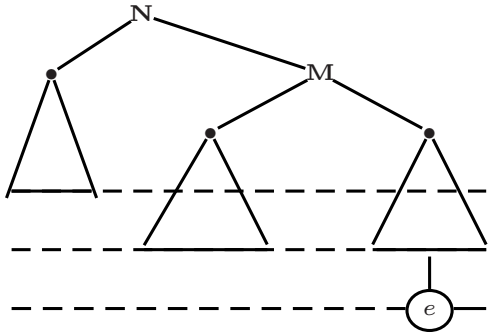
À partir de maintenant , on travaille avec la représentation chaînée d'un arbre AVL

Rééquilibrage après insertion dans un arbre AVL

Hypothèse Insertion $\Longrightarrow$ **déséquilibre**

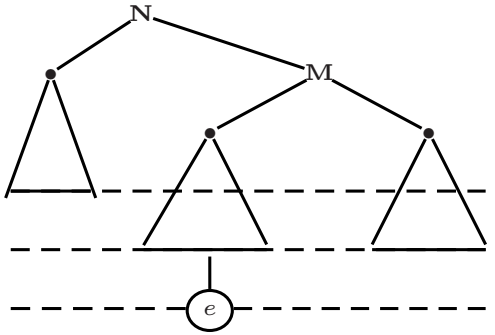


bal(N)=-2



– CAS 1 –

bal(N)=-2 avec bal(M)=-1

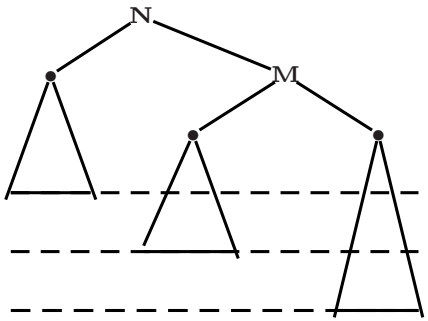


– CAS 2 –

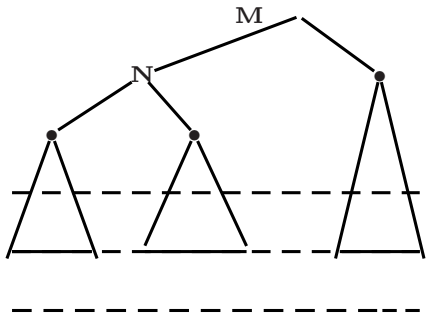
bal(N)=-2 avec bal(M)=1

Rééquilibrage après insertion dans un arbre AVL

Rotation gauche – CAS 1 –



bal(N)=-2 avec bal(M)=-1



bal(N)=0 et bal(M)=0

$$Rotation\_Gauche(N, G, (M, G_D, D_D)) = (M, (N, G, G_D), D_D)$$

Propriété de la rotation gauche

décrémente la hauteur de l'arbre (dans les conditions ci-dessus)

préserve les propriétés des arbres binaires (toujours)

préserve le parcours infixe/symétrique (toujours)

Rééquilibrage après insertion dans un arbre AVL

Rotation gauche (simple)

`Algo Rotation_Gauche_Avl`

Entrée : N AVL Sortie : N AVL ;

Auxiliaire : b_n, b_m : integer; M : AVL ;

$M \leftarrow N.droit$;

$b_n \leftarrow N.bal$; $b_m \leftarrow M.bal$;

$N.droit \leftarrow M.gauche$;

$M.gauche \leftarrow N$;

$N.bal \leftarrow b_n - min(b_m, 0) + 1$;

$M.bal \leftarrow max(b_n + 2, b_n + b_m + 2, b_m + 1)$;

$N \leftarrow M$;

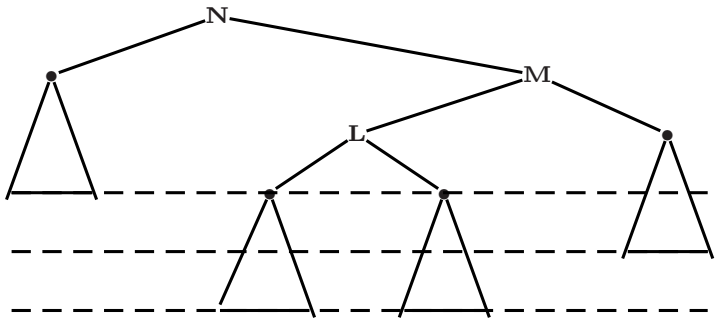
Cet algorithme prend en entrée des arbres binaires qui ne sont pas des AVLs

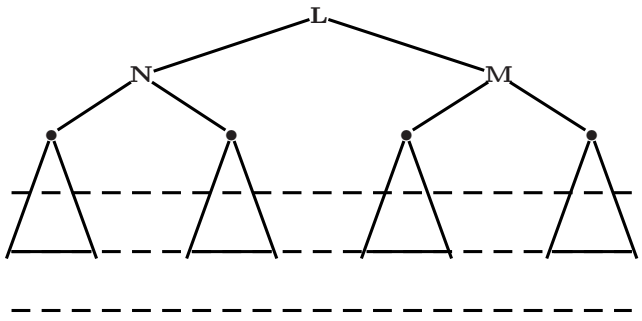
Exercice Rotation droite simple

Effectuer l'analyse pour le cas de l'insertion d'un nouvel élément dans le fils gauche du fils gauche de la racine

Rééquilibrage après insertion dans un arbre AVL

Double Rotation droite-gauche – CAS 2 –

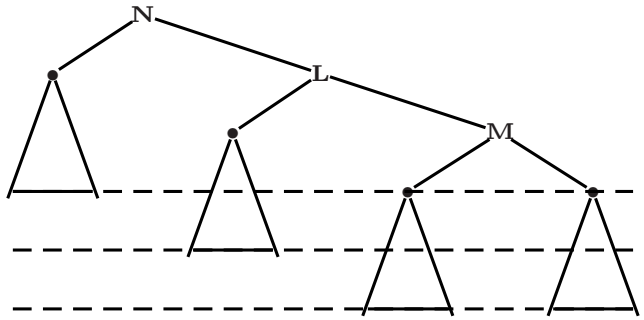




bal(N)=-2 avec bal(M)=1

bal(L)=bal(N)=bal(M)=0

Rotation droite (en M)



Rotation gauche (en N)

bal(N)=-2 bal(L)=-1 bal(M)=0

Rééquilibrage après insertion dans un arbre AVL

Double Rotation droite-gauche (suite)

cas traité : $bal(L) = 0$

les cas $bal(L) = 1$ et $bal(L) = -1$ sont similaires

Exercice Double rotation gauche-droite

Effectuer l'analyse duale pour traiter l'insertion du nouvel élément dans le sous-arbre gauche du sous-arbre gauche.

`Algo Equilibre_AVL`

Entrée/Sortie : N AVL ;

---

if (N.bal = -2) **then**
  if ($N.droit.bal \leq 0$) **then** $N \leftarrow$ Rotation_Gauche(N)
  else $\{$ $N.droit \leftarrow$ Rotation_Droit($N.droit$); $N \leftarrow$ Rotation_Gauche(N) $\}$
else if (N.bal = 2) **then**
  if ($N.gauche.bal \geq 0$) **then** $N \leftarrow$ Rotation_Droit(N)
  else $\{$ $N.gauche \leftarrow$ Rotation_Gauche($N.gauche$); $N \leftarrow$ Rotation_Droit(N) $\}$;
endelse

Cet algorithme prend en entrée un arbre binaire de recherche qui n'est pas toujours un AVL

Insertion dans un arbre AVL

Variation de la hauteur d'un AVL après insertion

(en supposant que l'inseertion est faite dans le sous-abre droit)

balance		variation
avant	après	de hauteur
1	0	0
0	-1	+1
-1	0	0

L'insertion est décomposée en trois étapes:

1. insertion "classique" dans un arbre binaire de recherche
2. mise à jour des balances des noeuds
3. rééquilibrage

Lors d'une insertion dans un AVL, un rééquilibrage, au plus, est nécéssaire.

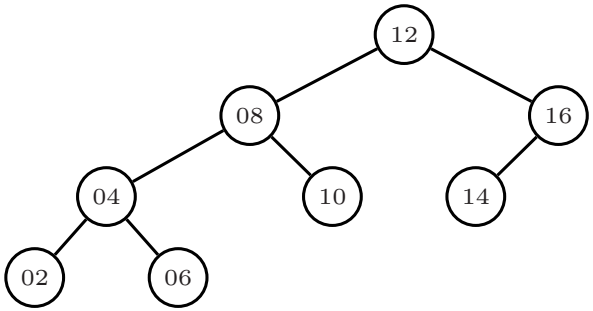
Exemple : Traiter la construction d'un AVL par insertion successive de 2 10 12 4 16 8 14

Exercice : Écrire l'algorithme d'insertion dans un AVL.

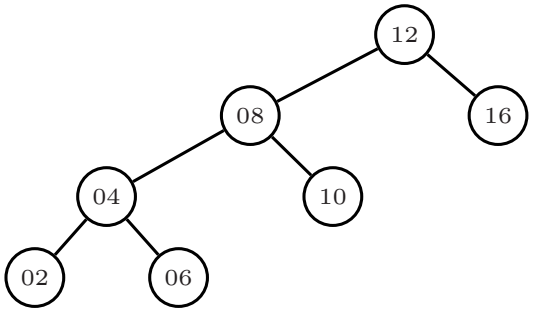
Suppression d'un élément dans un arbre AVL

Même principe que pour l'insertion ...

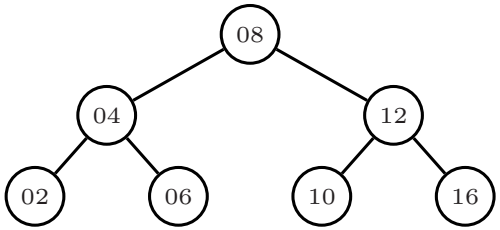
Exemple



Arbre AVL



Après suppression de 14



Après rotation droite (à la racine)

Suppression d'un élément dans un arbre AVL

Attention ... Variation de la hauteur d'un AVL après suppression

(en supposant que la suppression est faite dans le sous-abre gauche)

balance		variation
avant	après	de hauteur
1	0	-1
0	-1	0
-1	0	-1
-1	-1	0

La suppression est décomposée en deux étapes:

1. la suppression "classique" dans l'arbre binaire de recherche
   - avec mémorisation du chemin de la racine au noeud supprimé
2. le rééquilibrage est ensuite effectué du père du noeud supprimé jusqu'à la racine (si nécessaire)

Lors de la suppression, il peut être nécessaire d'effectuer un nombre de rééquilibrages égal à la hauteur de l'arbre

Exercice : Écrire l'algorithme de suppression d'un élément dans un arbre AVL.

Conclusion sur les arbres AVL

Complexité des opérations sur un arbre AVL

Opération	coût†	rotation
Recherche	$\Theta(log_2 n)$	0
Insertion	$\Theta(log_2 n)$	$\Theta(1)$
Suppression	$\Theta(log_2 n)$	$\Theta(1)$

† : nombre de noeuds visités

L'objectif est atteint mais **la mise en oeuvre est complexe**

il y a d'autres types d'arbres équilibrés plus facile à implémenter et plus efficaces ?!?!?!

Arbres 2-3-4

L'équilibre parfait mais en augmentant le degré de l'arbre

mettre plusieurs éléments dans un noeud pour retarder le moment

où le rééquilibrage est nécessaire.

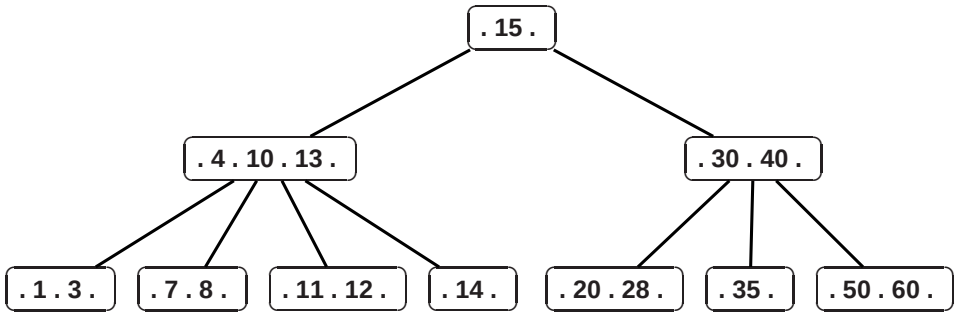
Définition

Un **arbre de recherche** est un arbre général ordonné dont chaque noeud contient k éléments distincts ordonnés.

Un noeud contenant les éléments $e_1 < ... < e_k$ a $k+1$ sous-arbres R_1, ..., R_{k+1} tels que :

1. tous les éléments de R_1 sont inférieurs ou égaux à e_1
2. pour $i = 2...k$, tous les éléments de R_{i-1} sont strictement supérieurs à e_i et inférieurs ou égaux à e_i.
3. tous éléments de R_{k+1} sont strictement supérieurs à e_k

Exemple



Arbres 2-3-4

Définition

Un **arbre 2-3-4** est un arbre de recherche général dont les noeuds contiennent entre 1 et 3 éléments (donc chaque noeud à entre 2 et 4 fils) et dont toutes les feuilles sont au même niveau.

Exemple transparent précédent

Hauteur d'un arbre 2-3-4

La hauteur $h(n)$ d'un arbre 2-3-4 contenant n éléments est en $\Theta(logn)$.

$$\boxed{log_4(n+1) \leq h(n) \leq log_2(n+1)}$$

Recherche dans un arbre 2-3-4

généralisation de la recherche dans un arbre binaire de recherche

à chaque "étape", on compare l'élément recherché e avec les éléments e_i du noeud courant

- si e est parmi les e_i ... la recherche est terminée
- sinon la recherche se poursuit dans le sous-arbre déterminé par la comparaison de e avec les e_i.

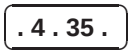
Complexité de la recherche $\Theta(h)$ où h est la hauteur de l'arbre i.e. $\Theta(\log(n))$

Insertion dans un arbre 2-3-4

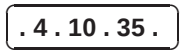
Construction d'un arbre 2-3-4

par insertions successives de 4, 35, 10, 13, 3, 30, 15, 12, 7, 40, 20, 11, 6.

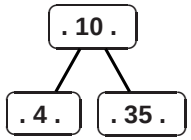
insertion de 4



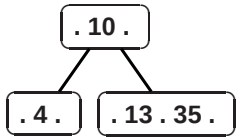
insertion de 35



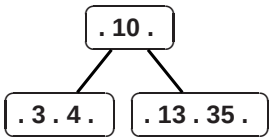
insertion de 10



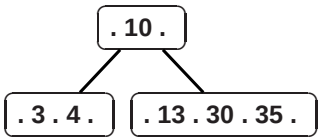
éclatement



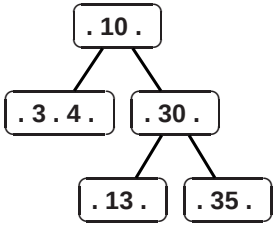
insertion de 13



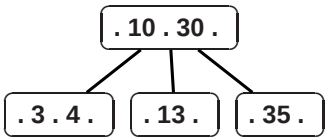
insertion de 3



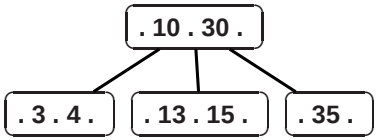
insertion de 30



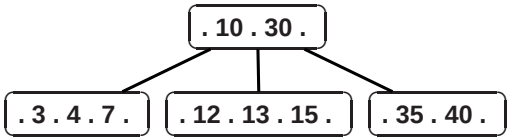
éclatement feuille droite



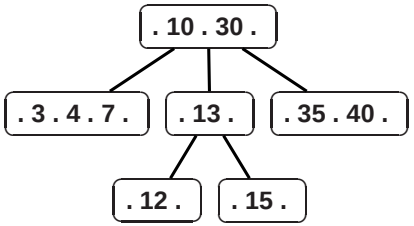
insertion de 30 "au dessus"



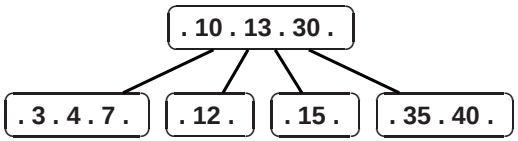
insertion de 15



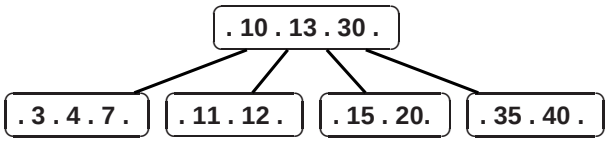
insertion de 12,7 et 40



éclatement feuille centrale



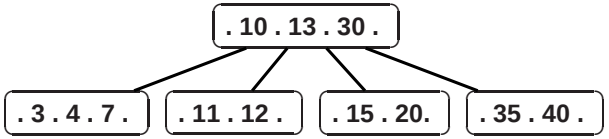
insertion de 13 "au dessus"



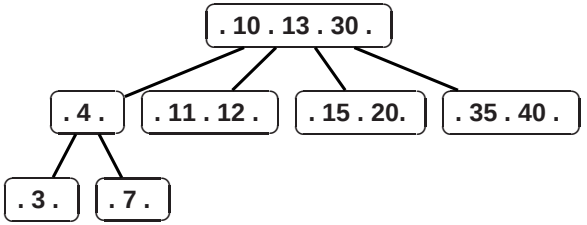
insertion de 20 puis 11

Insertion dans un arbre 2-3-4

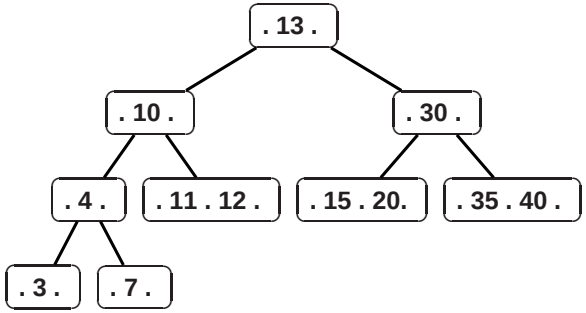
Insertion de 6



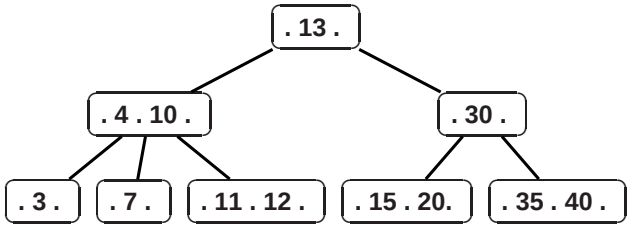
”reprise”



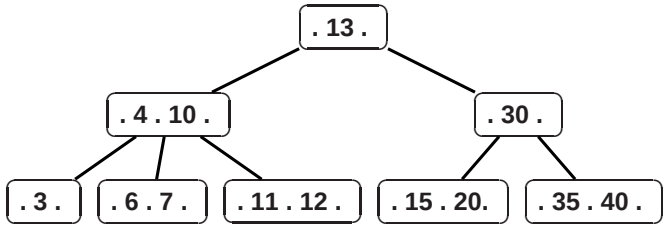
éclatement feuille droite



éclatement racine



insertion de 4 ”au dessus”



insertion de 6

Insertion dans un arbre 2-3-4

Méthode d'insertion avec éclatement en remontée

Deux étapes : **recherche** de la feuille d'insertion puis **restructuration**

- l'insertion dans une feuille peut provoquer l'éclatement de cette feuille et
  l'éclatement de tous les noeuds de cette feuille à la racine
- illustrée par l'exemple précédent
- simple à comprendre
- moyennement compliquée à implémenter

Complexité de l'insertion avec éclatement en remontée (nombre de noeuds modifiés)

$\Theta(h)$ où h est la hauteur de l'arbre i.e. $\Theta(\log(n))$

Méthode d'insertion avec éclatement anticipé

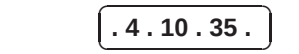
- anticiper les "débordements" pendant la phase de recherche
  pour éviter les remontées et pour favoriser les modifications locales.

"Une étape" : à la descente, lors de la **recherche** de la feuille d'insertion
tout noeud "plein" est éclaté.

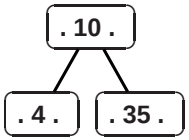
Insertion avec éclatement anticipé dans un arbre 2-3-4

Re-Construction d'un arbre 2-3-4

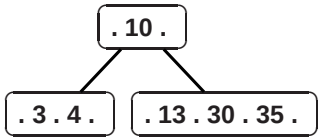
par insertions successives de 4, 35, 10, 13, 3, 30, 15, 12, 7, 40, 20, 11, 6.



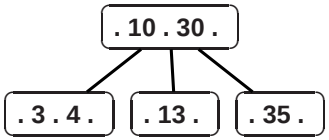
insertion de 4,35 et 10



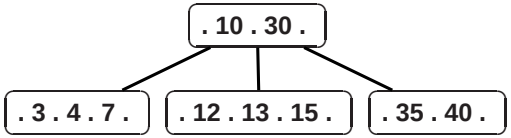
éclatement



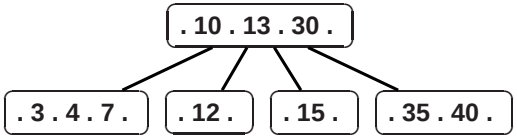
insertion de 13,3,30



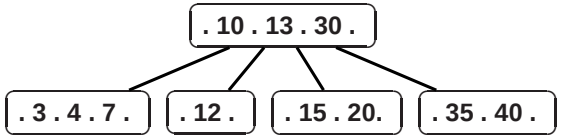
éclatement de la feuille droite



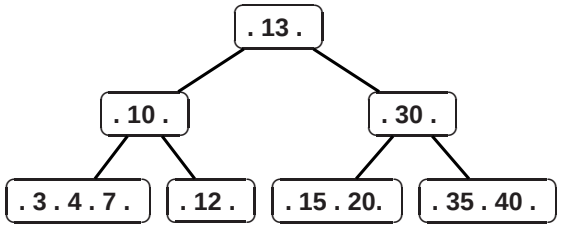
insertion de 15, 12, 7 et 40



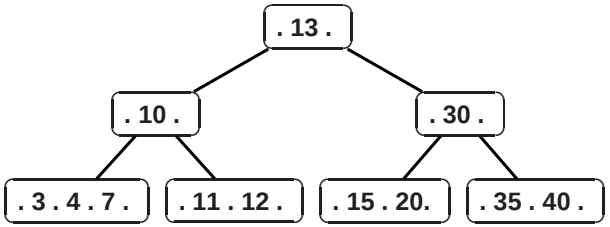
éclatement de la feuille centrale



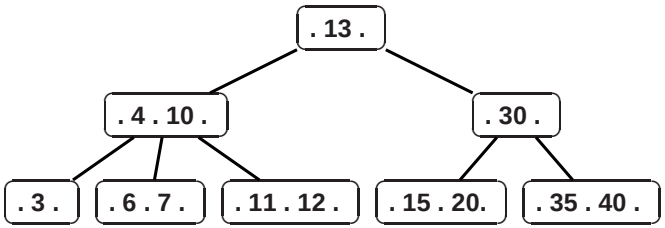
insertion de 20



éclatement racine



insertion de 11

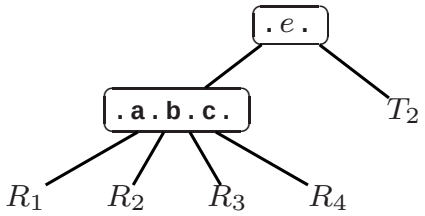


insertion de 6

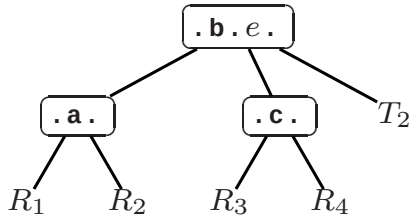
Insertion avec éclatement anticipé dans un arbre 2-3-4

Deux cas à considérer

1. **le père du noeud à éclater contient 1 élément**

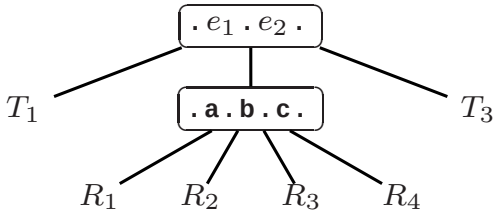


avant éclatement

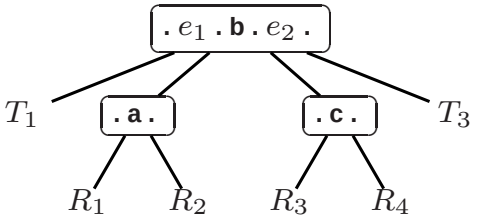


après éclatement

2. **le père du noeud à éclater contient 2 éléments**



avant éclatement



après éclatement

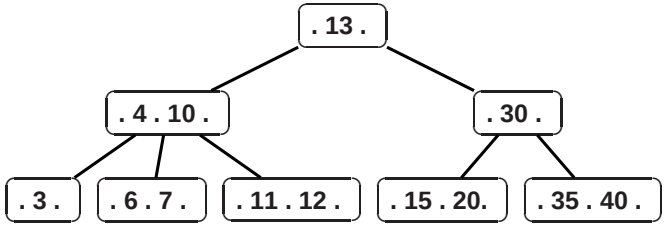
Exercice Écrire l'algorithme d'insertion avec éclatement anticipé.

Suppression dans un arbre 2-3-4

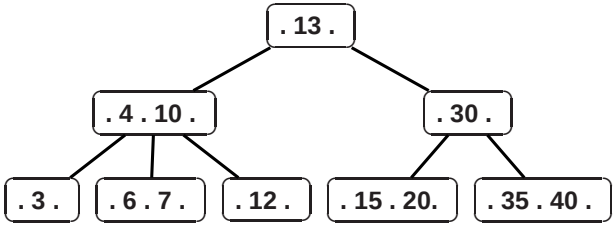
Méthode en ”une” étape Rechercher l’élément à supprimer en restructurant l’arbre.

Hypothèse : le noeud N contenant l’élément e à supprimer contient au moins 2 éléments.

Cas (a): N est une feuille. Suppression directe.



Arbre 2-3-4 initial



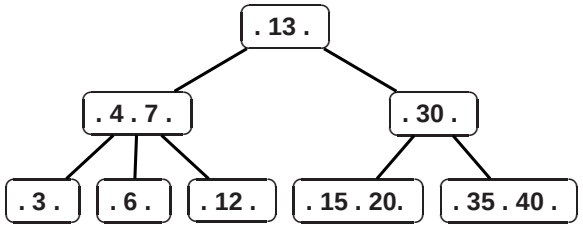
suppression de 11

Cas (b): La racine du fils F_G qui ”précède” l’élément e à supprimer contient au moins 2 éléments.

Rechercher le prédécesseur p de e dans F_G.

Supprimer (récursivement) p

Remplacer e par p dans N

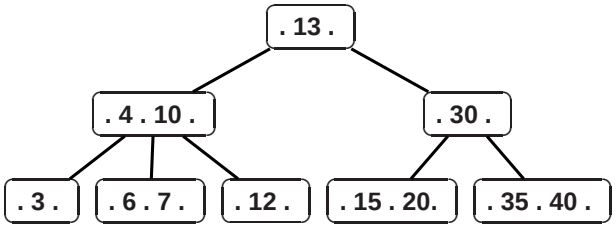


suppression de 10 (à la suite)

Suppression dans un arbre 2-3-4

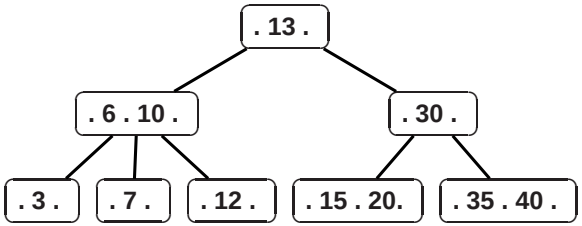
Cas (b) -symètrique-: La racine du fils F_D qui "succède" l'élément e à supprimer contient au moins 2 éléments.

Rechercher le successeur s de e dans F_D. Supprimer (récursivement) s et remplacer e par s dans N



suppression de 11

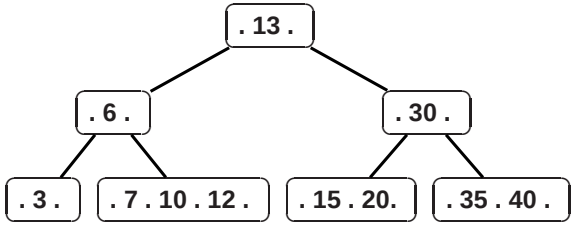
(reprise)



suppression de 4 (à la suite)

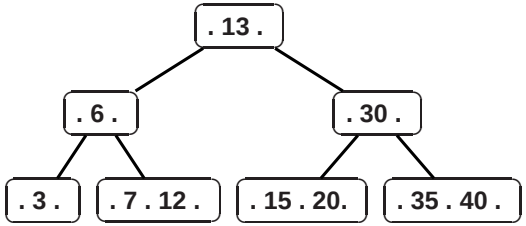
Cas (c) : Les racines M_G de F_G et M_D de F_D contiennent un seul élément chacun.

Fusionner M_G, e et M_D dans M_G. Libérer M_D. Supprimer (récursivement) e dans M_G.



fusion pour suppression de 10

(à la suite)



suppression de 10

Suppression dans un arbre 2-3-4

Ce qu'il reste à faire ?

Forcer à ce que lorsque l'élément e est trouvé dans un noeud N

l'hypothèse $\mathbf{N}$ **a au moins 2 éléments** soit satisfaite.

Traitements lors de la descente dans l'arbre à la recherche de $\mathbf{e}$.

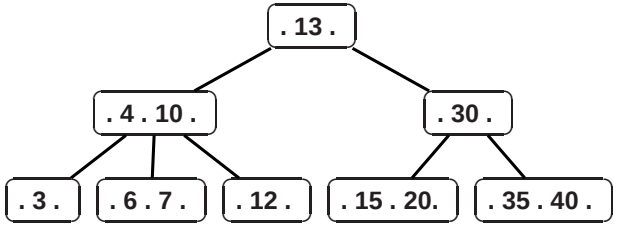
Hypothèse : le noeud N ne contient pas l'élément e à supprimer.

Le sous-arbre F de racine M contient e (si e est dans l'arbre).

Cas (i): M contient au moins deux éléments. Poursuivre la méthode en M.

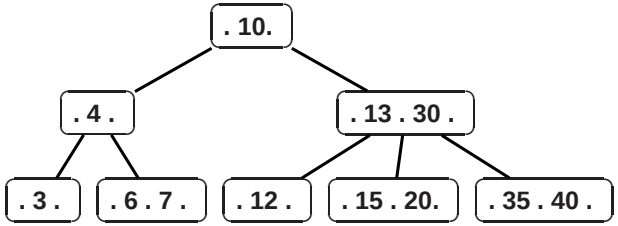
Cas (ii): M contient un seul élément et a un frère F **immédiat** qui contient au moins 2 éléments.

Redistribuer un élément de F vers M en mettant à jour N. Poursuivre en M.



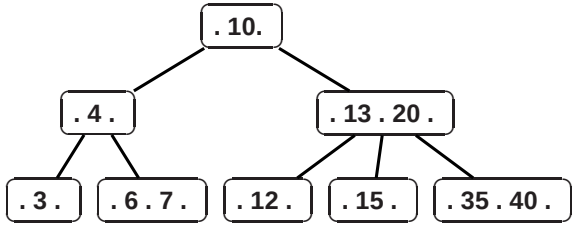
suppression de 11

(reprise)



redistribution pour suppression de 30

(reprise)

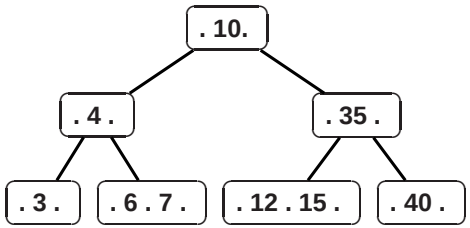


suppression de 30

Cas (a) ou (b)

Suppression dans un arbre 2-3-4

Suppression de 20 (Cas (b)):

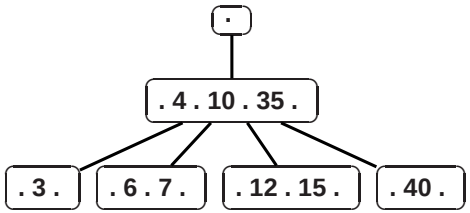


suppression de 13

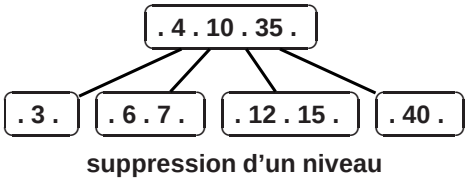
Cas (ii): Aucun frère **immédiat** de M ne contient plus d'un élément.

Soit F un noeud frère de M et f l'élément "entre" F et M. Fusionner F, f et M en M. Poursuivre en M.

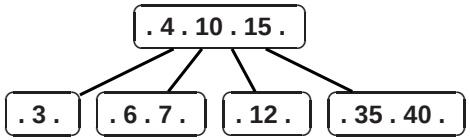
Suppression de 40.



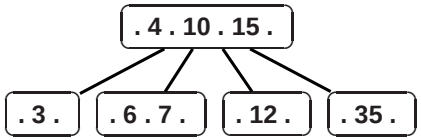
fusion "en" 35



suppression d'un niveau



redistribution "en" 40



suppression de 40

Arbres rouges et noirs

Définition Un arbre rouge et noir (RN) est un arbre binaire de recherche tel que:

1. chaque noeud est soit **rouge** soit **noir**
2. la racine est noire
3. si un noeud est rouge alors ses fils sont noirs
4. soit un noeud N de l'arbre, alors tous les chemins de N à une "demi-feuille" ou à une feuille (du sous-arbre enraciné en N) contiennent tous le même nombre de noeuds noirs

La **hauteur noire** d'un noeud N, notée $hn(N)$, est le nombre de noeuds noirs apparaissant sur un des chemins de N à une feuille plus 1 (voir propriété 4.)

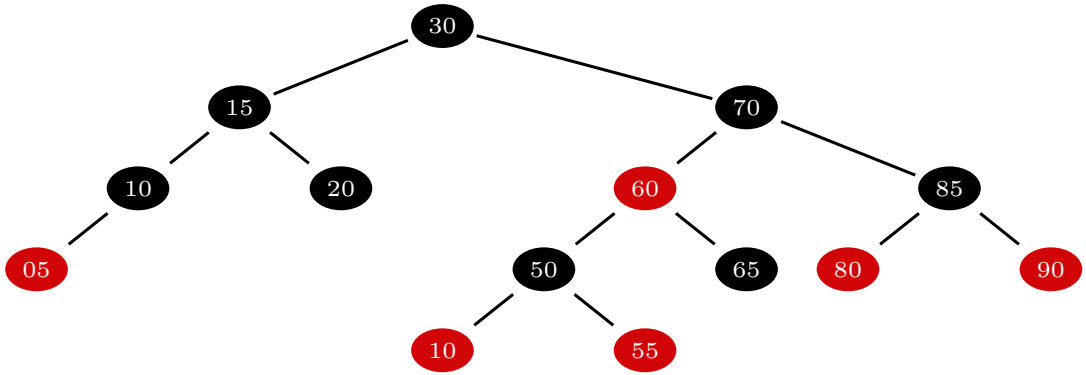
Remarque Un arbre RN peut avoir des noeuds noirs uniquement. Comment est l'arbre dans ce cas ?

Hauteur d'un arbre rouge et noir

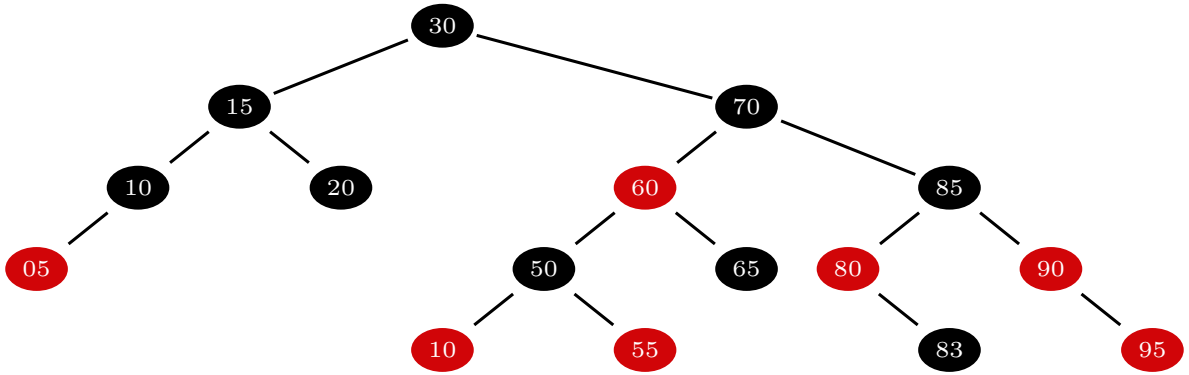
Un arbre RN de taille n (ayant n noeuds) a une hauteur au plus égale à $2log_2(n)$.

La longueur de tout chemin de la racine (noire) à une feuille de l'arbre RN ne peut être supérieure à 2 fois la hauteur noire de la racine (pas 2 noeuds rouges consécutifs).

Arbres rouges et noirs



Un exemple d'arbre rouge et noir



Un contre-exemple d'arbre rouge et noir

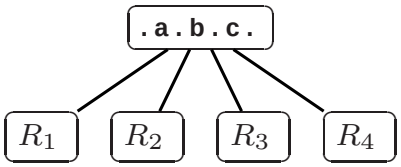
Arbre Rouge et Noir versus 2-3-4

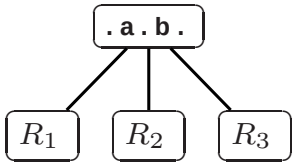
Les arbres bicolores sont une representation des arbres 2-3-4

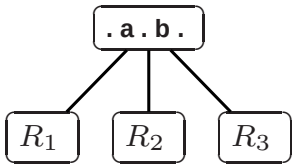
Transformation 2-3-4 vers bicolore

1. un noeud est transformé en un arbre binaire
2. les ”liens” sont colorés en rouge pour indiquer que se sont des noeuds provenant du même noeud 2-3-4.
3. ... on fait couler la couleur rouge sur les noeuds

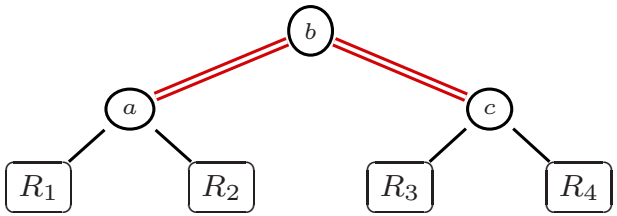
Arbre 2-3-4 initial

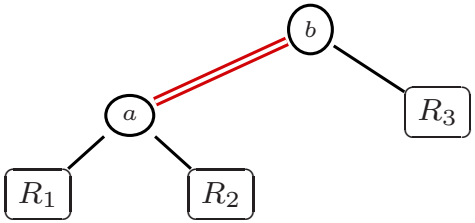


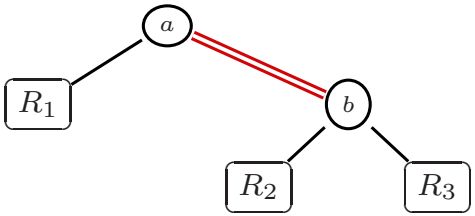




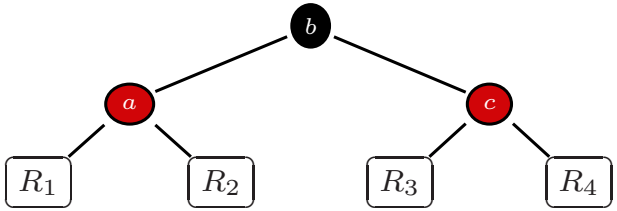
étapes 1 et 2

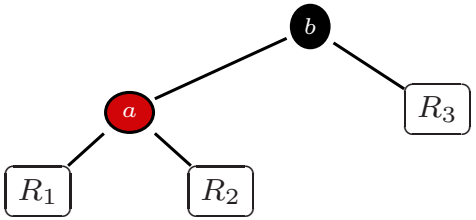


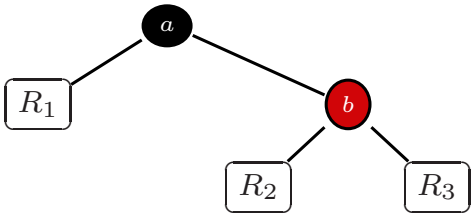




étape 3







Insertion d'un élément dans un arbre rouge et noir

Quelques remarques pour commencer

l' insertion "classique" dans un arbre binaire de recherche crée une nouvelle feuille

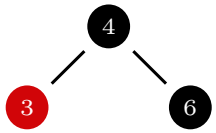
la condition 4 $\Longrightarrow$ **coloriage en rouge** du noeud créé

si le père du noeud créé est noir ... c'est fini

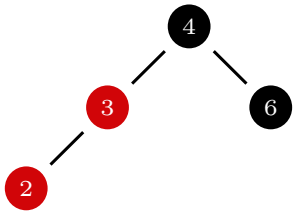
sinon la condition 3 est non satisfaite et la restauration de cette condition

se fait en effectuant des rotations.

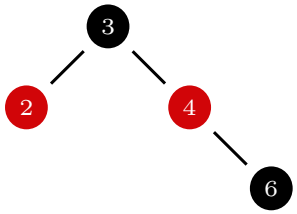
Exemple



arbre RN initial



insertion de 2



arbre RN restauré

Cas illustré :

Le père du noeud 2 dans l'arbre "après insertion" est rouge.

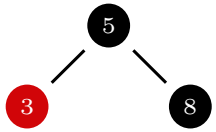
Le frère du père est noir.

Une rotation droite simple suffit à rétablir les conditions.

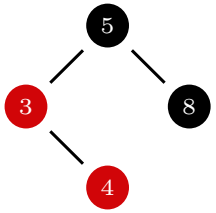
Attention : la racine doit être noire.

Insertion d'un élément dans un arbre rouge et noir

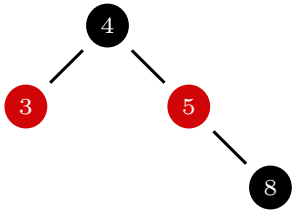
Autre Exemple



arbre RN initial



insertion de 4



arbre RN restauré

Cas illustré :

Le père du noeud 4 dans l'arbre "après insertion" est rouge.

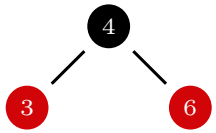
Le frère du père est vide (considéré comme noir).

Une double rotation gauche-droite rétablit les conditions.

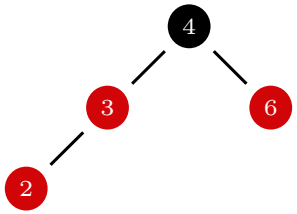
Attention : la racine doit être noire.

Insertion d'un élément dans un arbre rouge et noir

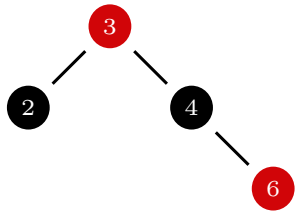
Encore un exemple



arbre RN initial



insertion de 2



arbre après rotation droite

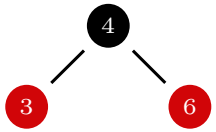
Cas illustré :

Le père du noeud 2 dans l'arbre "après insertion" est rouge.

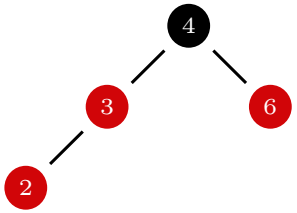
Le frère du père est **rouge**.

Une rotation droite simple **peut ne pas suffire** à rétablir les conditions. Donc ne pas le faire.

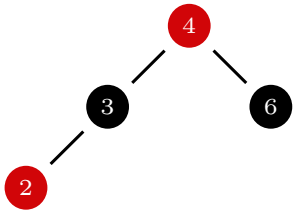
Attention : la racine doit être noire et il faut propager les modifications



arbre RN initial



insertion de 2



arbre après recoloration

Suppression d’un élément dans un arbre rouge et noir

La méthode peut se déduire de la suppression d’un élément dans un arbre 2-3-4

en faisant quelques ”transformations”.