

Plan :

- Hashage
- des optimisations des structures d'arbre
- APg ~~et bouton~~
- Programmation dy namique - Backtracking
- structure de données méthode de conception des apgs
- arbre ABR
- Arbre AVL
- arbre rouge noir

facto
pb < difficile

APg → descendante (APg et bouton)

→ Ascendante (on calcule une fois on exploite plusieurs fois)

Hachage

Motivation:

- * Structure tableau trié (contigüe)
 - Recherche (coût fixe)
 - Inversion (coût plus important)

recherche de @ où on va placer l'élément
 Manque de lien entre l'élément et l'@
 → prévoir une structure qui permet d'avoir l'élément → @ accès direct à l'élément
 réduire le coût de calcul en minimisant les pas de calcul
- Calcul Espace

Recherche dans un fichier (fichiers Etudiant)

→ Parcours de fichier

Select Nom From Etudiant where — optimiser la complexité de recherche

Utilisation des index (fichiers de données → Fichiers d'index (Hachage)) ↓
 Accès direct aux données

Objectif:

- Prévoir des techniques d'indexation qui permettent l'accès rapide et direct aux données Pour :

- Réduire le nb d'op d'E/S

- Réduire l'espace de recherche

Hachage:

→ améliorer la construction des files qui permettent de localiser une clé dans un ensemble pré-déterminé indexé par des entiers

clé $\xrightarrow{\text{fonction}}$ @ . Permet d'établir un lien entre la clé et son adresse

$H(n) \in [0..m-1]$ Déf: On appelle une fonction de hachage une fonction qui détermine la place d'une entité (tuple) uniquement d'après sa clé $H: U \rightarrow [0..m-1]$

Exemple: $H(n) = n \bmod 11$ $|D| \ll |U|$ clé \rightarrow entier

• Des clés à insérer sont :

16 - 18 - 6 - 29 - 50 - 13 - 27

- 31 - 28

$x \neq y$

$H(x) \neq H(y)$
exigence

Medali

0	
1	
2	13
3	
4	
5	16
6	6
7	18
8	31
9	
10	
11	

Table de Hashage

pb de collision

(un châssis pour placer 8 b)

$H \rightarrow$ injective

mais non bijective

Select * from Etudiant where nom = "Med Ali"

Codage ($H(\text{Med Ali})$) = 18 $H(18) = 7$

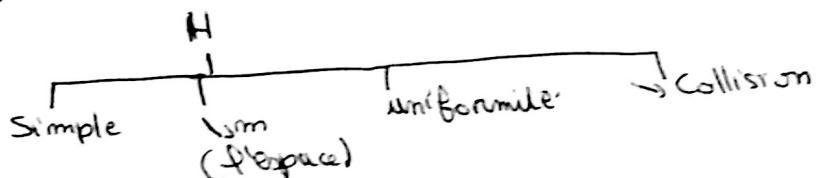
• Contrainte :

la f^{on} de Hashage doit être :

- Simple à calculer
- Permet de réduire la taille des entrées (minimiser m)
- Permet une répartition uniforme des clés dans
- Permet une répartition uniforme des clés dans

la Table de Hashage (avoir plus qu'une clé par case)

- Minimiser le nbre de collisions (avoir plus qu'une clé par case)
- sur la m^e entrée)



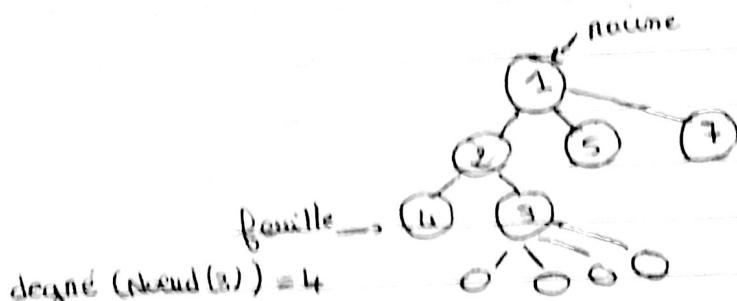
$\rightarrow m \Rightarrow \uparrow \text{collision}$

$\rightarrow m + \text{uniformité} \Rightarrow \downarrow \text{collision}$

Les arbres et leurs optimisations

Introduction : Un arbre est un ensemble :

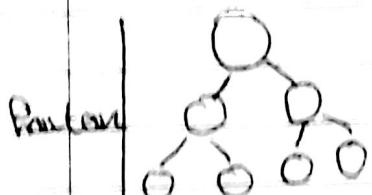
- nœuds
- un nœud racine
- chaque nœud (sauf nœud feuille) possède un ou plusieurs fils
- chaque nœud (sauf la racine) possède un seul nœud père
- Une feuille est un nœud qui ne possède pas de fils
- degré d'un nœud = nombre de nœuds fils



- arbre binnaire :

- Un arbre vide

- Un triplet (sousarbre Gauche, nœud, sousarbre Droit)



arbre binnaire complet : nœud $\left\{ \begin{array}{l} \text{famille} \\ \text{un nœud de degré } k \end{array} \right.$
arbre 2-aire

- arbre k-aire : génération d'un arbre binnaire

arbre complet /

arbre non complet

\forall nœud $\left\{ \begin{array}{l} \text{feuille} \\ \text{ou} \\ \text{degré } \geq k \end{array} \right.$

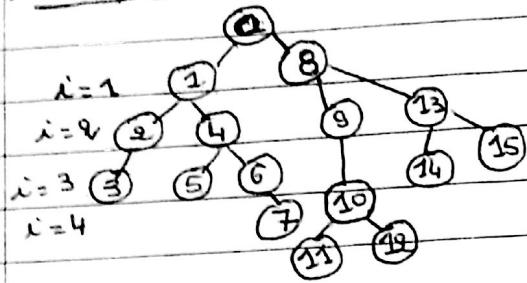
\forall nœud $\left\{ \begin{array}{l} \text{feuille} \\ \text{ou} \\ \text{degré max } = k \end{array} \right.$

- parcours des arbres

parcours en
largeur

en profondeur, hybride (mixte) (Pb spécifiques)

Exemple 1:



Parcours en largeur:

- racine
- puis les fils d'un niveau i
- puis les fils de niveau $i+1$

0, 1, 8, 2, 4, 9, 13, 3, 5, 6, 10, 14, 15, 7, 11, 19

Parcours en profondeur:

- 1) Prefixe: → Traiter Racine 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15
→ Préfixe (S.G)
→ Préfixe (S.N)
- 2) Infixe: → Infixe (S.G) 3, 9, 15, 4, 6, 7, 0, 11, 10, 12, 9, 8
Traiter racine
Infixe (S.D)
- 3) Postfixe: postfixe (S.G) 3, 9, 5, 7, 6, 4, 1, 11, 12, 10, 9, 14, 15, 13, 8, 0
postfixe (S.D)
Traiter racine

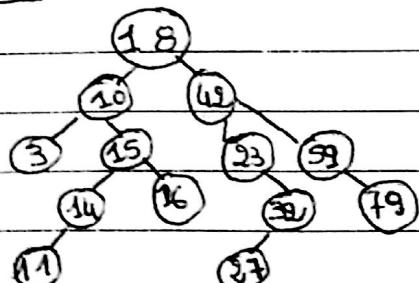
Arbre binaire de recherche : (ABA)

objectif: minimiser les noeuds visités (Recherche)
arbre binaire + répartition d'ordre

noeud { lien vers S.G
contenu : clé
lien vers S.D

répartition d'ordre \Rightarrow contenu (S.G) < clé < contenu (S.D)

Exemple 2:



ABA

- Parcours de en profondeur de type
- * Afficher le contenu de l'ABR → infixé
 - * Recherche d'un élément x
 - (Profondeur)
 - Recherche (racine, x)

→ Comparer le nœud / x

→ si: $x = \text{courant}$ OK

→ si: $x < \text{courant}$ recherche (courant \rightarrow gauche)
 $\quad\quad\quad$ racine recherche (courant \rightarrow droite)

 - * Recherche de min
 se trouve sur le nœud feuille de la branche la plus à gauche
 - * Insertion

racine \downarrow feuille

Problème de déséquilibre de l'ABR.

Résoudre le pb de déséquilibre

Revoir la Totalité \rightarrow Restructurat° locale

de la structure de restaurat° globale

cost globale (insertion)
 $= \underset{\text{min}}{\text{cost}}(\text{insert}) + \underset{\text{min}}{\text{cost}}(\text{équilibrage})$
- Exemple: Utilisation AVL

Optimisation des ABB : les AVL

ABB: structure peut être non équilibrée

→ recherche sera non efficace → comment remédier à ce pb

→ solution : AVL

Adeffan / Velstui / Landis → les auteurs qui ont proposé cette structure

→ AVL = ABB + équilibre

↳ pour éviter la recherche longue (dans certains cas)

* Définition: AVL ont un arbre binaire

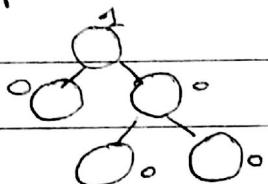
arbre vide est un AVL

$$| H(\text{SAD}) - H(\text{SAG}) | \leq 1$$

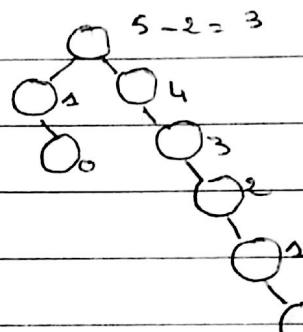
arbre : caractérisé par son SAD, contenu, SAG

$$H(\text{SAD}) - H(\text{SAG}) \in \{-1, 0, 1\}$$

Exemple:



→ non AVL



→ n'est pas un AVL

* Insertion dans un AVL:

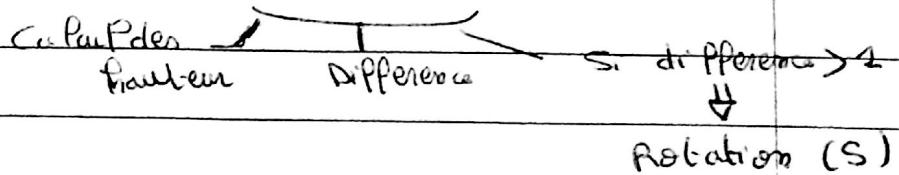
Entrée $\begin{cases} \text{Arbre AVL} \\ (E) \end{cases}$
Élément

Sortie(s) $\begin{cases} \text{arbre AVL} \\ (\text{modifié}) \end{cases}$

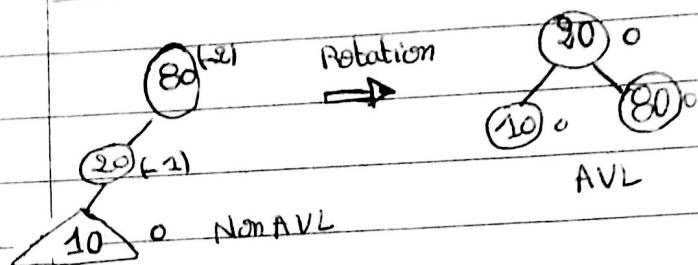
$E \rightarrow \boxed{\text{Insertion}} \rightarrow S$

insertion ABB + Restructuration pour avoir un AVL → basé sur l'opérat° d'équilibrage

$$\text{Coût (insert°)} = \text{Coût (insert ABB)} + \text{coût (Restructurat°)}$$

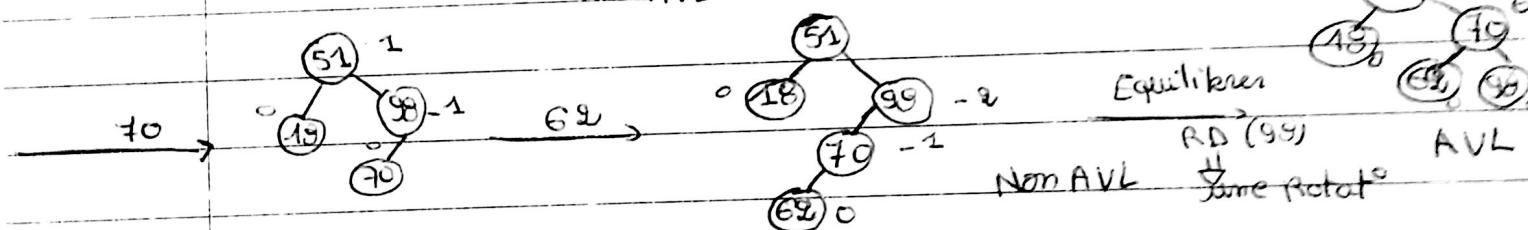
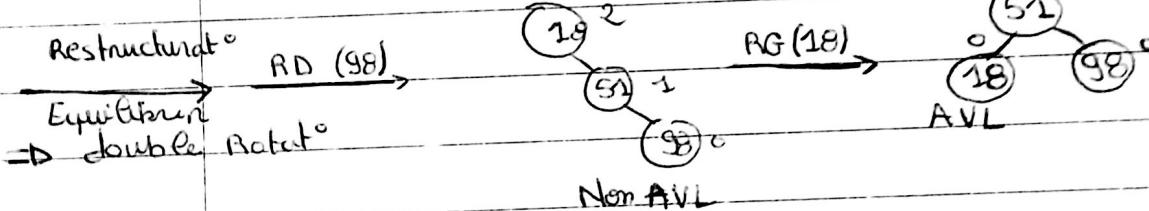
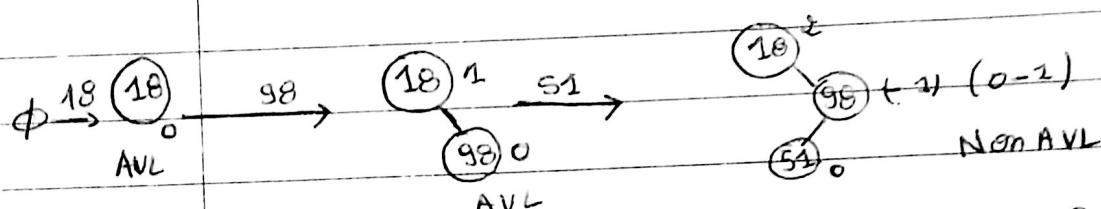


Exemple



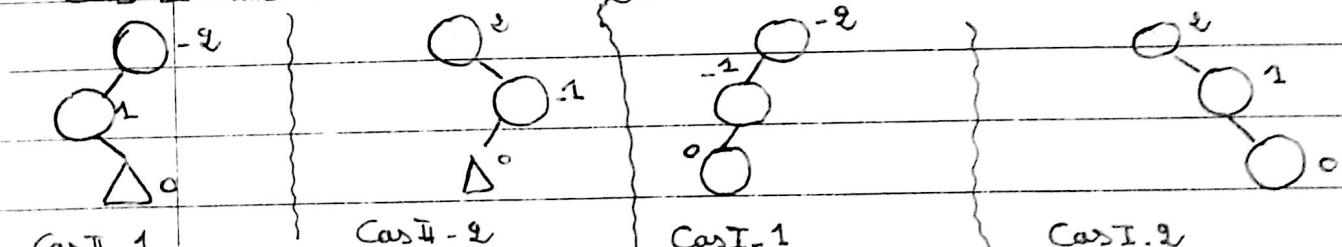
* Exemple : Insérer dans un AVL les éléments suivants

18 - 98 - 51 - 70 - 69



Entrée AVL + l'élément Δ

Cas II : insert° intérieur \Rightarrow TAVL } Cas I : insert° extérieur \Rightarrow TAVL



Equilibrer \Rightarrow deux notations

Equilibrer \Rightarrow une notation

10-28-09

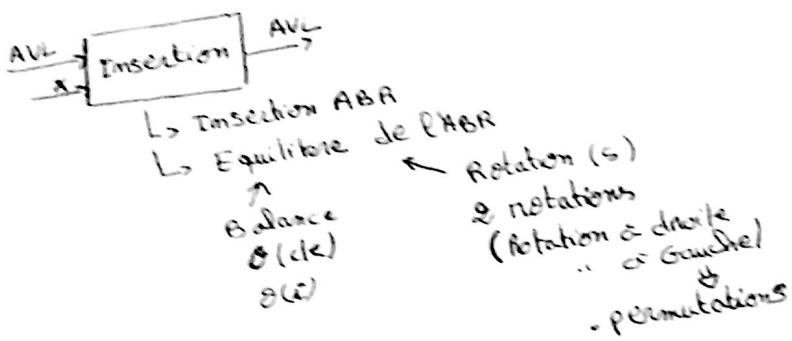
10-28-09 - 10-29-09

10-28-09
10-28-09

10-28-09

10-28-09

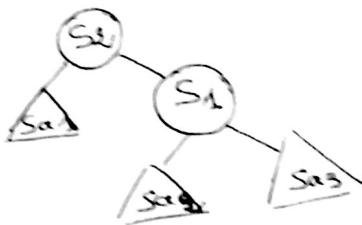
10-28-09
10-28-09
10-28-09



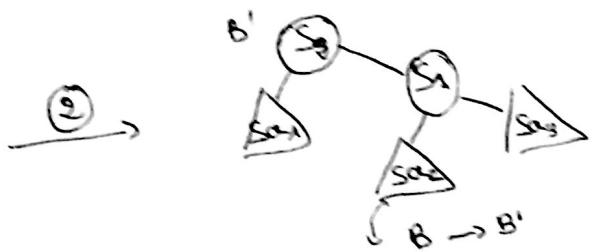
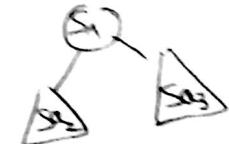
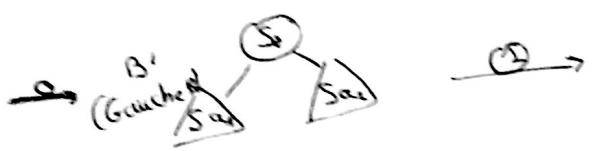
Rotation à droite:



$RD(S_1)$



Trace d'exécution



Afgo

ABA Rotation D(ABR B)

$B' \leftarrow$ gauche (B)

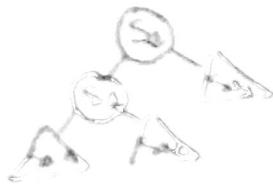
gauche (B) \leftarrow Droite (B')

Droite (B') $\leftarrow B$

$B \leftarrow B'$

retourner B

• Rotation à gauche.



→

non rotatif (cas 0)

$\theta = \text{chute}(0)$

$\text{chute}(\theta) \leftarrow \text{chute}(\theta)$

$\text{chute}(\theta) \leftarrow \theta$

$\theta \leftarrow \theta'$

rotatif

position \leftarrow rotation (double rotation)
position \leftarrow rotation (single rotation)

• Rotation (cas 1)

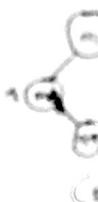


LD simple rotation

Cas 1.1 \leftrightarrow Rotation A(n)

Cas 1.2 \leftrightarrow Rotation B(n)

• Rotation (cas 2)



LD Double rotation

Cas 2.1 \leftrightarrow Rotation D(m)
+ Rotation G(m)

Cas 2.2 \leftrightarrow Rotation B(m)
+ Rotation D(m)

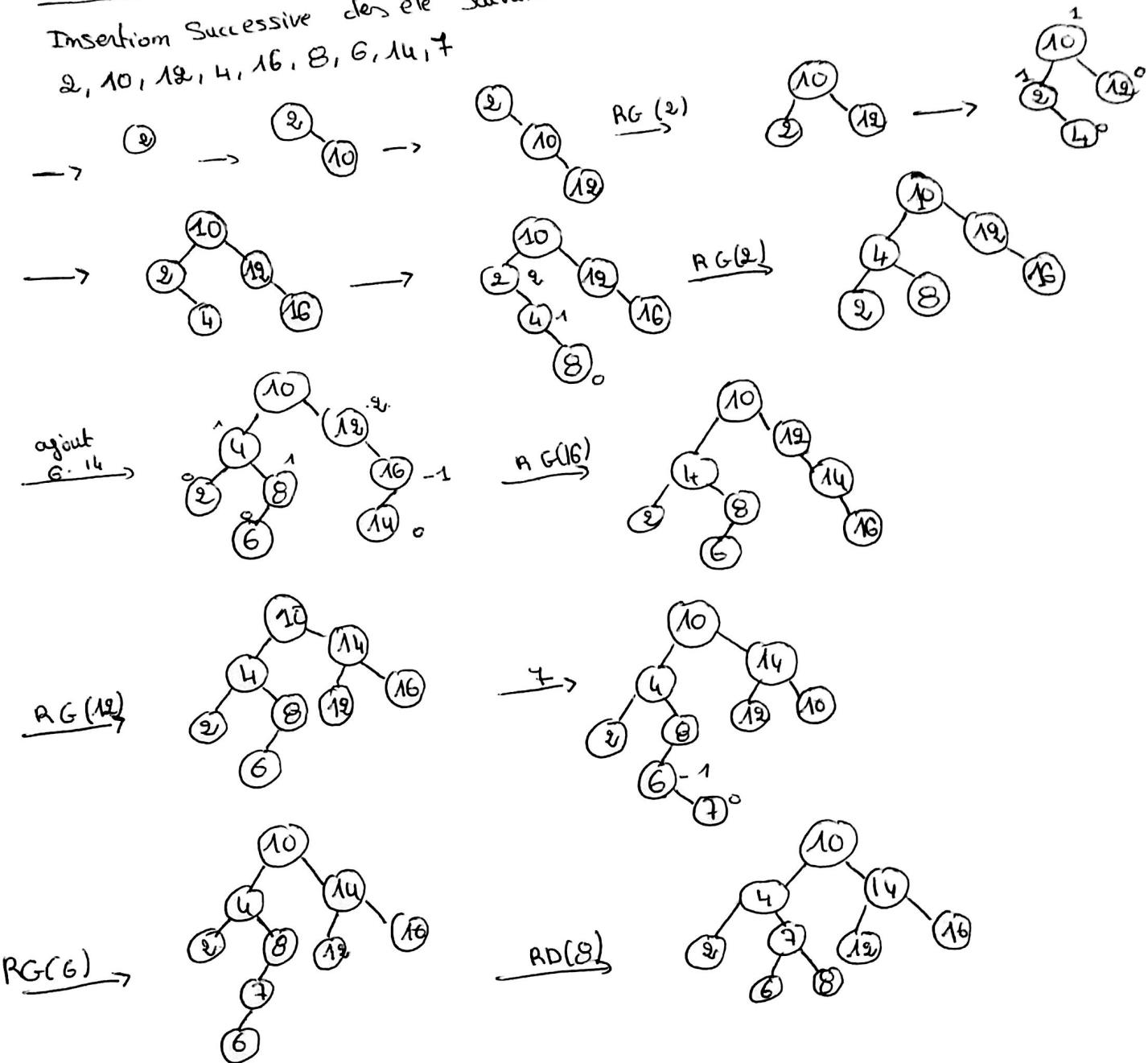
rot (cas 3)



AG (m)

Exemple:

Insertion successive des éléments suivants:
2, 10, 12, 4, 16, 8, 6, 14, 7



6 notations.

des arbres B (B-tree)

Introduction :

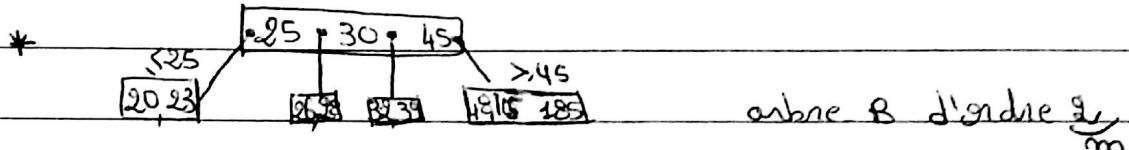
$$AVL = ABB + \text{équilibré}$$

Pb : équilibrage (rotation)

nb insertion \Rightarrow coût d'équilibrage \gg

Solution possible : retarder les opérations de redistr.

\Rightarrow Un nœud peut avoir un nbre de clés > 1



généralisat° de l'AVL

Un nœud contient pls clés

Reyer B-tree et H
McAllister 1972

Définit°.

• Arbre B est un arbre équilibré

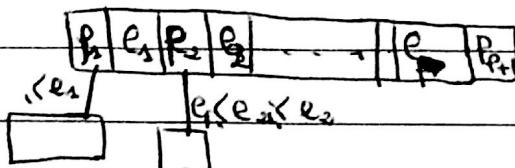
• Arbre B est un arbre d'ordre m est un arbre où chaque nœud sauf la racine contient un nbre d'éléments $|k|$

tq $m \leq |k| \leq 2m$ (chaque nœud est à moitié rempli)
et un nbre de pointeurs : $|p|$

$$m+1 \leq |p| \leq 2m+1$$

• La racine contient $1 \leq |k| \leq m$

• Chaque chemin de la racine à une feuille est de la m^e hauteur H



* L'utilisat° du B-tree est assez fréquente dans les SGBD et SG-f
(BT-tree, BT-tree, B-tree)

Exemple : *

Recherche dans un B-tree

↳ Généralisat° de la recherche sur la structure ABB

• Recherche dans un nœud (à partir de la racine)

• Si Trouvé → sortir

• Sinon recherche dans le sous arbre déterminé par l'opération

Comparaison

Insertion dans un B-tree

Suite à une insert° \Rightarrow B-tree.

Il est possible que la contrainte $|k| \leq 2m$ ne soit pas respectée

→ insert° de l'élément au niveau feuille

↳ La recherche de la feuille

↳ Possibilité que la contrainte ne soit pas respectée

\Rightarrow B-tree \Rightarrow il faut redessiner pour avoir un B-tree

Exemple 1:

Insérer dans un B-tree 10, 5, 40, 15, 30

