Chapitre 6

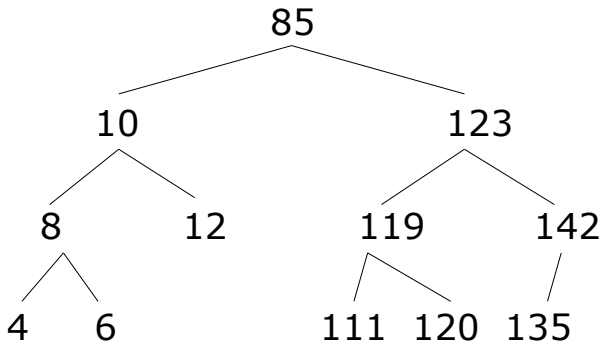
Arbres binaires de recherche AVL / Équilibrage des arbres / Arbres rouges et noirs

2) Arbres AVL

Un arbre binaire de recherche est un **arbre AVL** si pour chaque nœud, la différence de hauteur de chacun de ses fils n'excède pas 1

An algorithm for the organisation of information, G. Adelson-Velsky and E. Landis, 1962

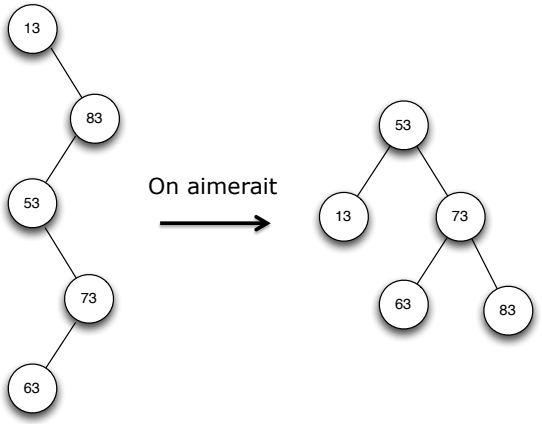
Exemple : un arbre AVL



Arbres AVL

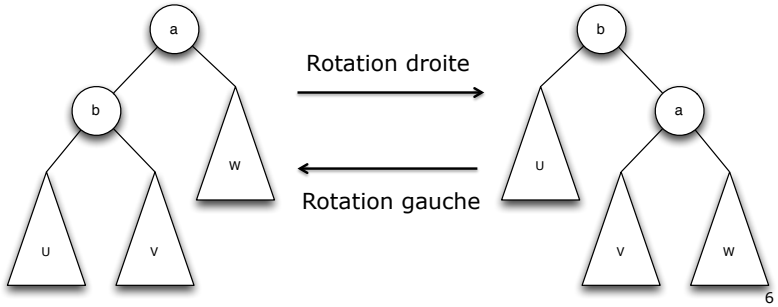
- Nécessité d'équilibrer l'arbre en permanence
- Ex : arbre créé avec les valeurs suivantes :
  - 13, 83, 53, 73 et 63

Construction classique



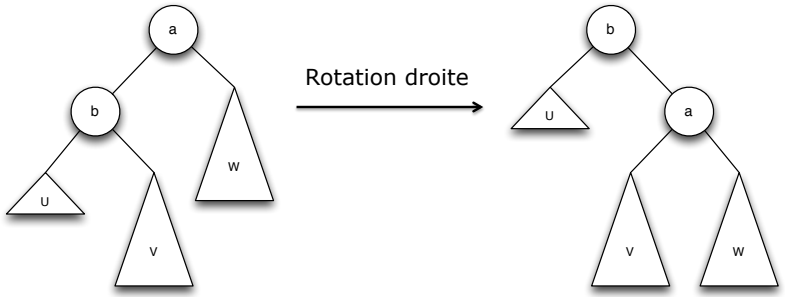
Équilibrage des arbres

- Est effectué par **rotations**
- Deux types de rotations



Problème

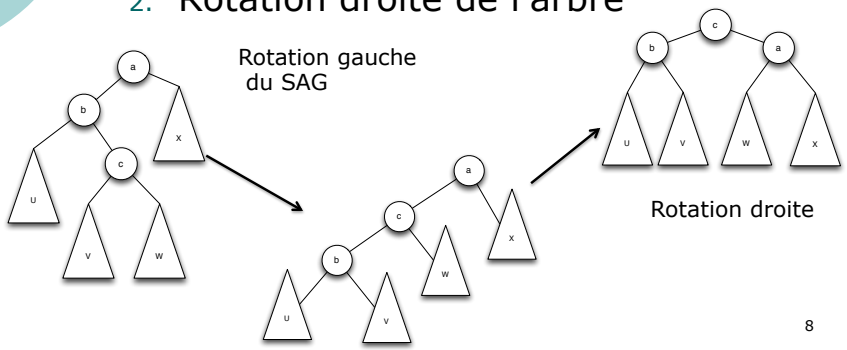
- Les **rotations** ne préservent pas la propriété AVL



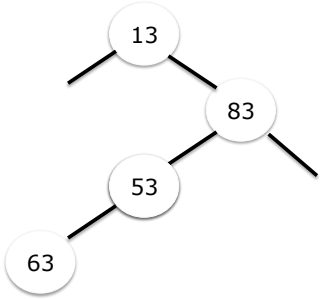
Solution : double rotation !

Ex:

1. Rotation gauche du SA gauche
2. Rotation droite de l'arbre



Construction avec équilibrage



|H(fg) – H(fd)| > 1

RÉÉQUILIBRAGE !

Rééquilibrage arbre AVL

- après une insertion
  - Une rotation ou double rotation suffit.
- après une suppression
  - jusqu'à h rotations ou double rotations
  - h est la hauteur de l'arbre

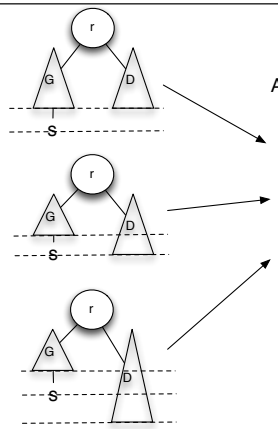
Rééquilibrage – Algorithme 1/1

- Soit A un arbre, G et D ses sous arbres gauche et droit tel que
  - |h(G) – h(D)| = 2
- Si h(G) – h(D) = 2
  - rotation droite de A

Rééquilibrage – Algorithme 2/2

- **g** et **d** : les sous-arbres gauche et droit de **G**.
- Si h(g) < h(d)
  - rotation gauche de G.
  - Rotation droite de A
- Si h(G) – h(D) = -2
  - opérations symétriques.

Suppression / variation de la hauteur



Équilibre Avant	Équilibre Après	Variation de hauteur
-1	0	-1
0	1	0
1	0 ou -1	-1 ou 0

- Variation = -1 ssi équilibre = 0 après retrait

Suppression d'un nœud – algo 1/3

```
(arbre,int) ENLEVER(valeur x, arbre A) {
        si (x > A->val)
                (A->fd,h) = ENLEVER(x,A->fd);
        sinon si (x < A->val)
                (A->fg,h) = ENLEVER(x,A->fg); h = -h;
        sinon si (A->fg == NULL) {
                /* free */ renvoie (A->fd,-1);
        } sinon si (A->fd == NULL) {
                /* free */ renvoie (A->fg,-1);
        } sinon {
                A->val = min(A->fd);
                (A->fd,h) = OTERMIN(A->fd);
        }                                                    …/…
```


Suppression d'un nœud – algo 2/3

```
        si (h == 0) return (A,0); // différence de
hauteur nulle : pas d'équilibrage
        sinon {
                A->bal = A->bal + h; // MAJ equilibre
                A = EQUILIBRER(A);
                si (A->bal == 0)
                        return (A,-1);
                sinon
                        return (A,0);
        }
}
```


Suppression d'un nœud – algo 3/3

```
(arbre,int) OTERMIN(arbre A) {
        si (A->fg == NULL) {
                min = A->val;          /* free */
                return (A->fd,-1);
        } sinon
                (A->fg,h) = OTERMIN(A->fg); h = -h;
        si (h == 0)
                return (A,0);
        sinon {
                A->bal = A->bal + h;
                A = EQUILIBRER(A);
                si (A->bal == 0) return (A,-1);
                sinon  return (A,0);
        }
```

Remarque

- ENLEVER et OTERMIN peuvent exécuter une rotation sur chaque ancêtre du nœud supprimé
  - Maximum **h** rotations

Question

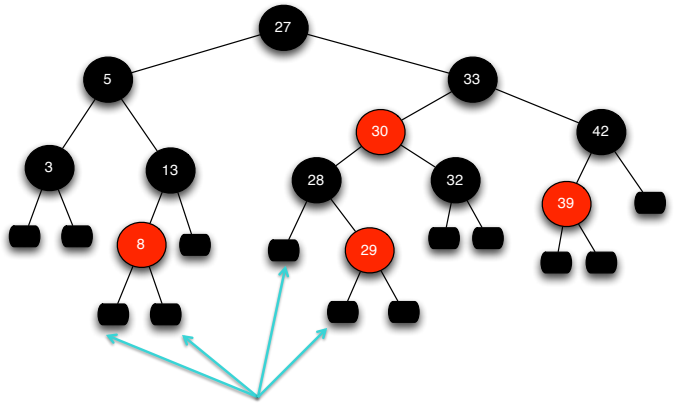
- Pourrait-on s'affranchir des multiples rotations à effectuer après suppression dans un arbre AVL ?

➢ Arbres rouges et noirs

Arbre rouge et noir / ARN

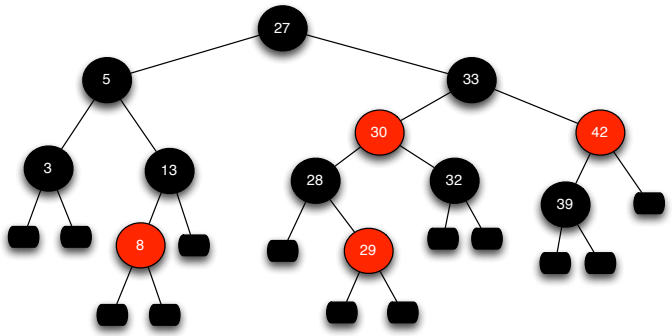
- C'est un arbre binaire de recherche
- Chaque nœud est soit rouge, soit noir
- Avec les règles suivantes:
  1. les nœuds externes (= pointeurs NULL des feuilles) sont noirs
  2. les fils d'un nœud rouge sont noirs
  3. Le nombre de nœuds noirs traversés par un chemin allant de la racine à une feuille est indépendant de ce chemin.
  - Autrement dit : pour tout nœud de l'arbre, les chemins de ce nœud vers les feuilles possèdent le même nombre de nœuds noirs.

Arbre rouge et noir - exemple



Nœuds externes : soit NULL, ou alors nœud sans valeur

Arbre rouge et noir – contre exemple



Arbre rouge et noir - règles

- Règle 1 : justification technique
- Règle 2 : les nœuds rouges seront peu nombreux
- Règle 3 : si on enlève les nœuds rouges, on obtient un arbre binaire de recherche parfaitement équilibré

Hauteur et hauteur noire

- On appelle **hauteur noire *hn(a)*** d'un arbre ***a*** le nombre de nœuds internes noirs le long d'une branche de la racine de ***a*** à une feuille
- ***a*** possède au moins $\mathbf{2^{hn(a)}-1}$ nœuds internes
  - Si **hn(*a*) == 0** alors **a** est une feuille
  - Si **hn(*a*) > 0**, alors la **hn** de ses fils vaut :
    - **hn(*a*)** si le fils est rouge
    - **hn(*a*)-1** si le fils est noir
  - Donc : **a** contient au moins $\mathbf{2(2^{hn(a)-1}-1)+1 = 2^{hn(a)}-1}$ nœuds internes

Hauteur noire et nœuds internes

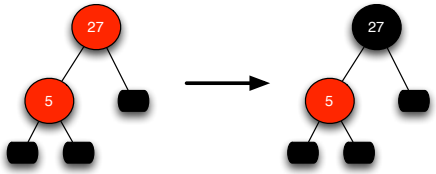
- Soit ***h*** la hauteur d'un ARN ***a***, la moitié des nœuds sur le chemin vers une feuille sont noirs.
- Donc ***hn(a)*** vaut au moins **h/2**
- Donc
  - $n >= 2^{hn(a)}-1$
  - $\log_2(n+1) >= hn(a) >= h/2$
  - $h <= 2.\log_2(n+1)$

ARN – Insertion

- Étape 1 : insertion classique dans un ABR
  - Le nouveau nœud est rouge -> règle 3 vérifiée
- Étape 2 : règle 2 peut ne plus être vérifiée
  - Si le père du nouveau nœud est rouge
  - -> faire des rotations
  - 3 cas possibles :
    - le père est la racine
    - le frère du père est rouge
    - le frère du père est noir

ARN – insertion – cas 1

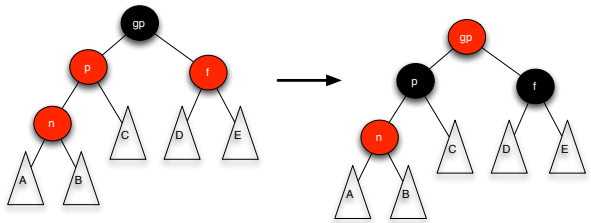
- Cas 1 : le père est la racine
  - La père devient noir



  - Seul cas où la hauteur noire augmente
    - pour tous les chemins !

ARN – insertion – cas 2

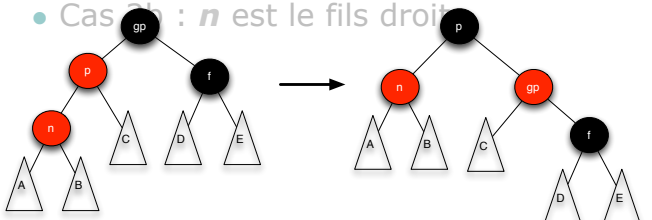
- Cas 2 : le frère ***f*** du père ***p*** est rouge
  - Le père et son frère deviennent noir
  - Le grand père ***gp*** devient rouge



- **gp** et son père peuvent être rouges

ARN – insertion – cas 3

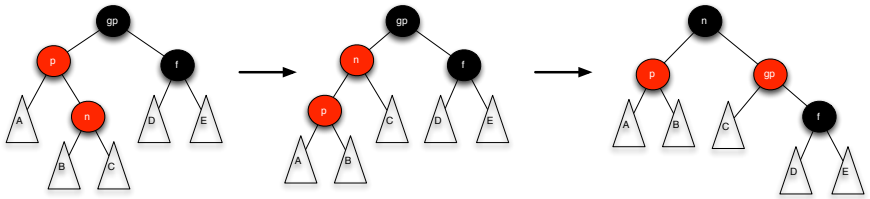
- Cas 3 : le frère ***f*** du père ***p*** est noir
  - Cas 3a : ***n*** est le fils gauche
    - rotation droite de **gp**, **gp** devient rouge
    - **p** devient noir
  - Cas 3b : ***n*** est le fils droit



- Symétrie si **p** est le fils droit

ARN – insertion – cas 3

- Cas 3 : le frère ***f*** du père ***p*** est noir
  - Cas 3a : ***n*** est le fils gauche
  - Cas 3b : ***n*** est le fils droit
    - double rotation gauche droite de ***p***
    - **gp** devient rouge, **n** devient noir



ARN – insertion - recap

- Cas 1 : le père est la racine
  - le père devient noir
- Cas 2 : le frère ***f*** du père ***p*** est rouge
  - le père et son frère deviennent noir
  - le grand père ***gp*** devient rouge
- Cas 3 : le frère ***f*** du père ***p*** est noir
  - Cas 3a : ***n*** est le fils gauche
    - rotation droite de **gp**, **gp** devient rouge
    - **p** devient noir
  - Cas 3b : ***n*** est le fils droit
    - double rotation gauche droite de ***p***
    - **gp** devient rouge, **n** devient noir

ARN – insertion - exemple

- Créer un arbre avec les nœuds suivants : A L G O R I T H M E

ARN - suppression

- Idem arbre binaire de recherche
  - Si le nœud possède 0 ou 1 fils
    - Ce nœud est supprimé
    - Son éventuel fils le remplace
  - Si le nœud a 2 fils
    - Le nœud prend la valeur strictement supérieure, c'est le nœud correspondant qui est supprimé
    - Si le nœud supprimé est rouge, les règles sont vérifiées
    - Si le nœud supprimé est noir, la hauteur noire est modifiée
      - Il faut faire les modifications nécessaires

ARN - suppression

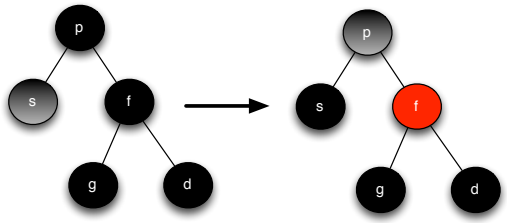
- On considère que le nœud **s** à supprimer porte une couleur noire supplémentaire
  - S'il est **rouge**, il devient **noir**
  - S'il est **noir**, il devient **doublement noir**
  - Il faut supprimer le nœud doublement noir
- Plusieurs cas
  - Cas 1 : **s** est la racine de l'arbre
  - Cas 2 : le frère **f** de **s** est noir
  - Cas 3 : le frère **f** de **s** est rouge

ARN - suppression

- Cas 1 : **s** est la racine de l'arbre
  - **s** devient simplement noir
  - Toutes les contraintes sont alors respectées

ARN - suppression

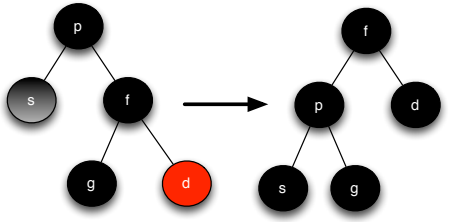
- Cas 2 : le frère **f** de **s** est noir
  - Cas 2a : les fils de **f** sont noirs
    - **s** devient simplement noir, **f** devient rouge
    - **p** devient noir s'il était rouge, doublement noir s'il était noir



  - Symétrie si **s** est le fils droit de **p**

ARN - suppression

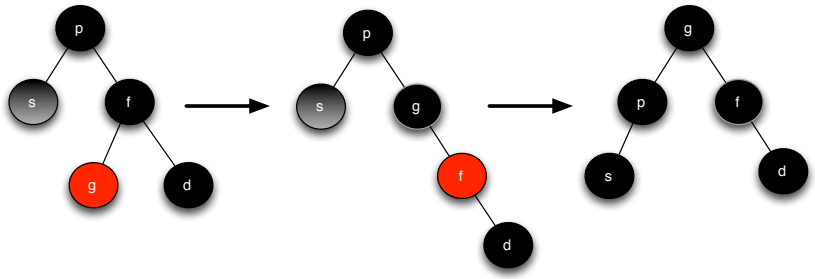
- Cas 2 : le frère **f** de **s** est noir
  - Cas 2b : le fils droit **d** de **f** est rouge
    - rotation gauche sur **p**
    - **f** prend la couleur de **p**
    - **s**, **p** et **d** deviennent noirs



  - Symétrie si **s** est le fils droit de **p**

ARN - suppression

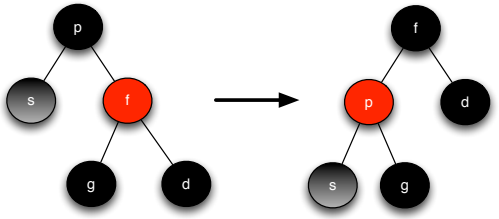
- Cas 2 : le frère **f** de **s** est noir
  - Cas 2c : le fils droit **d** de **f** est noir, **g** est rouge
    - rotation droite sur **f**, et **f** devient rouge
    - On se retrouve dans le cas précédent



  - Symétrie si **s** est le fils droit de **p**

ARN - suppression

- Cas 3 : le frère **f** de **s** est rouge
  - rotation gauche sur **p**
  - **p** devient rouge, **f** devient noir
  - on revient au cas 2



- Symétrie si **s** est le fils droit de **p**

ARN – suppression - recap

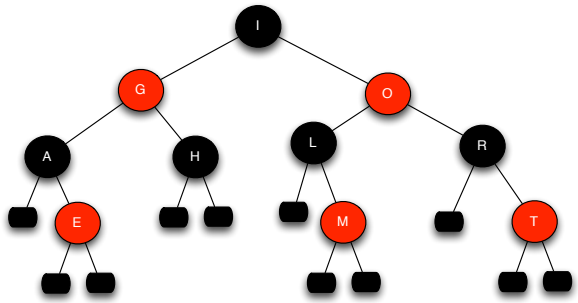
- Cas 1 : **s** est la racine de l'arbre
  - **s** devient simplement noir
- Cas 2 : le frère **f** de **s** est noir
  - Cas 2a : les fils de **f** sont noirs
    - **s** devient noir, **f** devient rouge
    - **p** devient noir si rouge, doublement noir si noir
  - Cas 2b : le fils droit **d** de **f** est rouge
    - rotation gauche sur **p**
    - **f** prend la couleur de **p, s**, **p** et **d** deviennent noirs
  - Cas 2c : le fils droit **d** de **f** est noir, **g** est rouge
    - rotation droite sur **f**, et **f** devient rouge
    - on se retrouve dans le cas précédent (2b)
- Cas 3 : le frère **f** de **s** est rouge
  - rotation gauche sur **p**
  - **p** devient rouge, **f** devient noir
  - on revient au cas 2

ARN – suppression

- Approche «bottom-up»
  - On fait remonter le problème vers la racine jusqu'à résolution
- On peut montrer que la suppression peut se faire en maximum :
  - 1 rotation
  - 2 changements de couleur

ARN – suppression - exercice

- Soir l'ARN suivant :



- Supprimer A L G O R I T H M E

Aujourd'hui : exercices

- AVL d'entiers
- Mise en œuvre par pointeurs

1. Rotation droite
2. Rotation gauche
3. Equilibrage
4. Insertion
5. Bonus : suppression

Arbres construits aléatoirement

Algorithme	Test	h	τ_0 (s)	τ_1 (s)	τ_2 (s)	τ_3 (s)
Naïf	1	41	19,73	0,08	0,04	0,04
AVL	1	18	19,71	0,08	0,05	0,08
Rouge-Noir	1	19	20,09	0,08	0,05	0,06
Naïf	2	45	47,02	0,20	0,13	0,07
AVL	2	19	46,03	0,20	0,14	0,17
Rouge-Noir	2	20	47,05	0,20	0,14	0,13
Naïf	3	42	109,81	0,44	0,36	0,15
AVL	3	20	107,30	0,47	0,26	0,38
Rouge-Noir	3	21	108,79	0,49	0,33	0,27

Test 1 : $m = 1\,000\,000$, $n = 64\,000$ et $p = 4\,000$
Test 2 : $m = 2\,000\,000$, $n = 128\,000$ et $p = 8\,000$
Test 3 : $m = 4\,000\,000$, $n = 256\,000$ et $p = 16\,000$

n : nombre de nœuds de l'arbre
h : hauteur de l'arbre
τ_0 : insertion/recherche de m clés aléatoires
τ_1 : suppression de p clés aléatoires
τ_2 : insertion de p clés aléatoires
τ_3 : suppression des n clés dans l'ordre croissant

Source: http://stephane.glondu.net/projets/tipe/transparents.pdf

Dans un des cas les pires

Algorithme	Test	h	$\frac{h}{\log_2 n}$	τ_0 (s)	$\frac{\tau_0}{n \log_2 n}$ (μs)
Naïf	1	31 999	2 138	614	1 383
AVL	1	14	0,9	0,06	0,13
AVL	2	18	0,9	1,31	0,14
AVL	3	20	1,0	5,51	0,14
Rouge-Noir	1	26	1,7	0,08	0,17
Rouge-Noir	2	34	1,8	1,79	0,19
Rouge-Noir	3	38	1,8	9,76	0,24

Test 1 : $n = 32\,000$
Test 2 : $n = 512\,000$
Test 3 : $n = 2\,000\,000$

n : nombre de nœuds de l'arbre
h : hauteur de l'arbre
τ_0 : insertion des n clés dans l'ordre croissant

Source: http://stephane.glondu.net/projets/tipe/transparents.pdf