Arbres H-équilibrés

Arbre AVL

extension type arbre
opérations
déséquilibre : arbre → entier
sémantique
déséquilibre(a) = **si** a = Ø **alors** 0 **sinon** h(g(a)) - h(d(a)) **fsi**

- a H-équilibré si pour tout b sous-arbre de a, déséquilibre(b) {-1, 0, +1}
- arbre H-équilibré => $\log_2(n+1) \leq h + 1 \leq 1.45 \log_2(n+1)$
- AVL: arbre binaire de recherche H-équilibré

=> *la recherche dans un arbre AVL est au pire en* $\Theta(\log_2 n)$

Remarques

- Rappel:

  h(a) = **si** a = Ø **alors** -1 **sinon** 1 + max(h(g(a)), h(d(a))**) fsi**
  déséquilibre(a) = **si** a = Ø **alors** 0 **sinon** h(g(a)) - h(d(a)) **fsi**

- Il s'en suit:

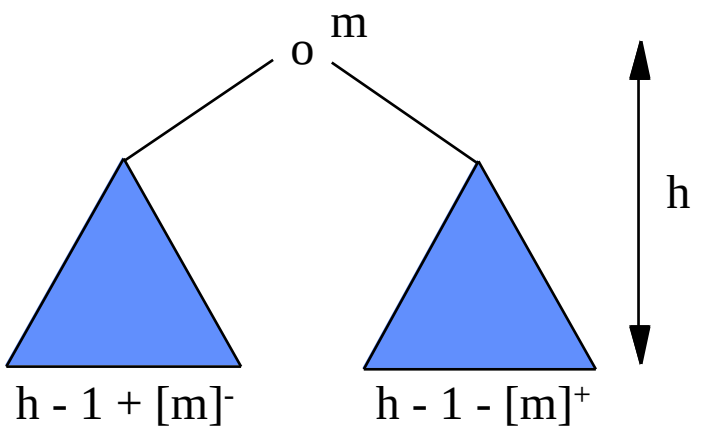
  $h(g(a)) = h(a) - 1 + inf(0, déséquilibre\ (a)) = h(a) - 1 + [déséquilibre(a)]^-$

  $h(d(a)) = h(a) - 1 - sup(0, déséquilibre\ (a)) = h(a) - 1 - [déséquilibre(a)]^+$

- en notant:

  $[x]^+ = sup\ (0, x)$

  $[x]^- = inf\ (0, x)$



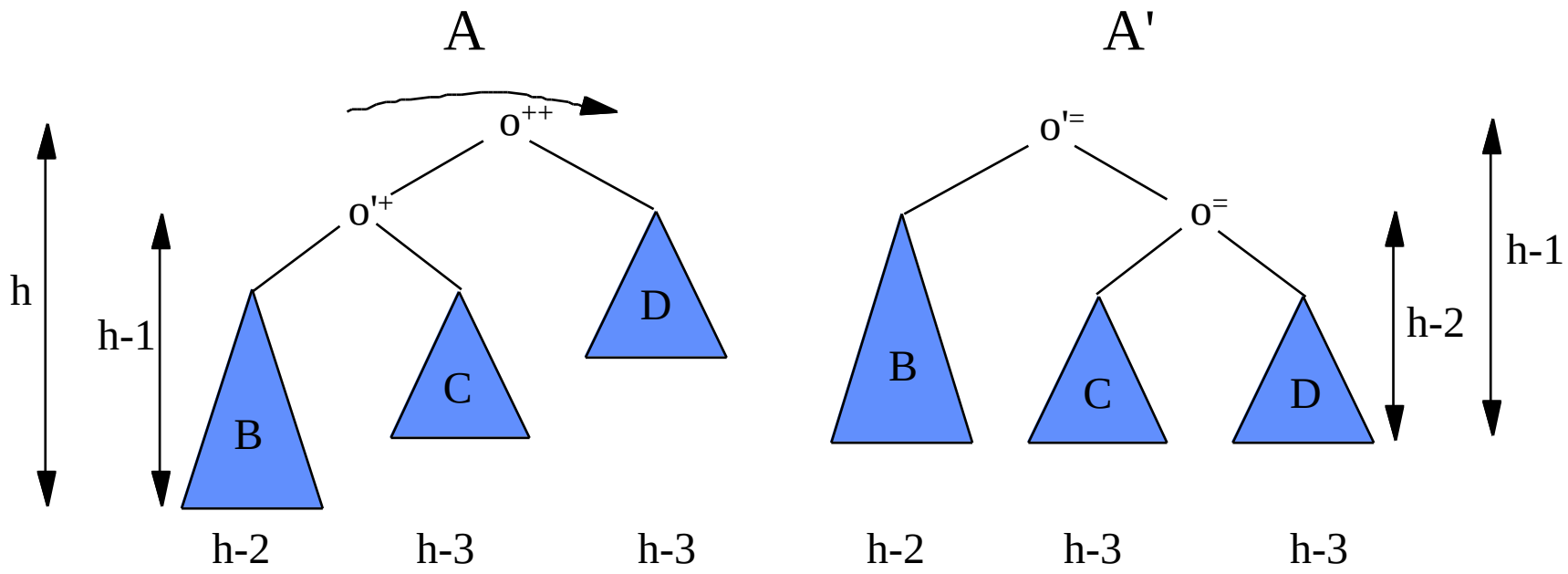
Implantation Ada


```
generic
package Arbres_Binaires.De_Recherche.AVL is
    type Arbre_AVL is new Arbre_Rech with null record;
    procedure Ajouter (Dans : in out Arbre_AVL; Val : in Element);
    procedure Supprimer_Courant (Dans : in out Arbre_AVL);
private
    type Desequilibre is range -2 .. +2;
    type Noeud_AVL is new Noeud_Rech with record
        Deseq : Desequilibre := 0;
    end record;
    procedure Reequilibrage (Le_Noeud : in out Pt_Noeud);
end Arbres_Binaires.De_Recherche.AVL ;
```

Implantation Java


```
package bibSDJava.Les_Arbres_Binaires;
class Noeud_AVL extends Noeud_Recherche { int deseq = 0; }
public class Arbre_AVL extends Arbre_De_Recherche {
   public Arbre_AVL (Avec_Cle Specimen)
                throws Erreur_Specification, Erreur_Typage {
      super (34, Specimen);
   }
   public void Ajouter (Avec_Cle Val)
                throws Cle_Existante, Limite_Implnatation, Erreur_Typage {}
   public void Supprimer_Courant ()
                throws Erreur_Specification {}
}
```

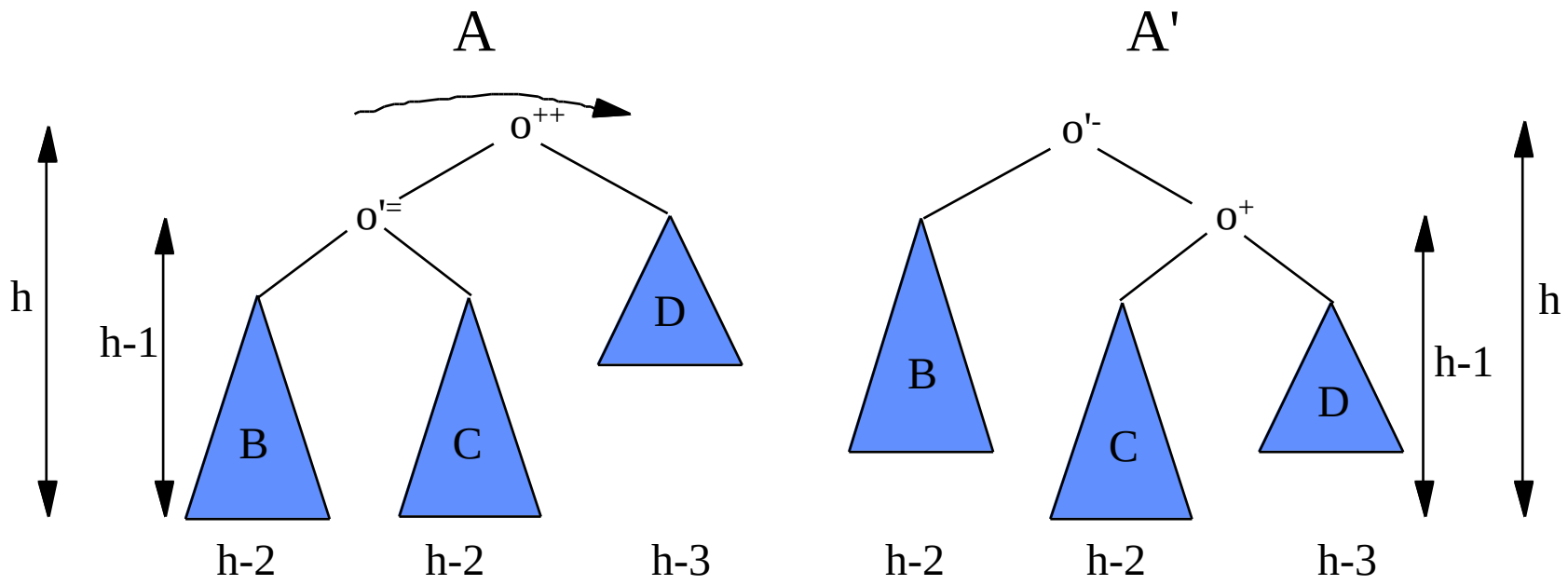
Rotation à droite (1)



- Propriétés:
  - A arbre binaire de recherche => A' arbre binaire de recherche
  - si B, C, D sont H-équilibré => A' est H-équilibré

déséquilibre(A)=2, déséquilibre(g(A))=1 => rotation à droite

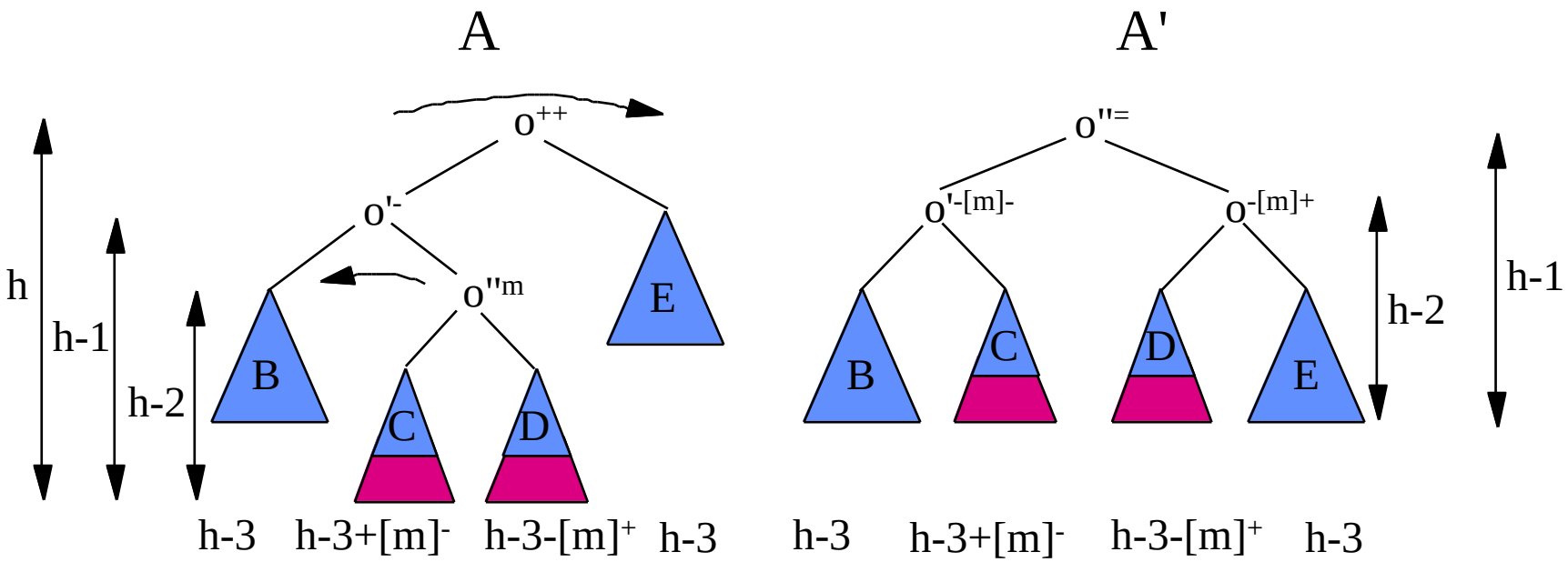
Rotation à droite (2)



- Propriétés:
  - A arbre binaire de recherche => A' arbre binaire de recherche
  - si B, C, D sont H-équilibré => A' est H-équilibré

déséquilibre(A)=2, déséquilibre(g(A))=0 => rotation à droite

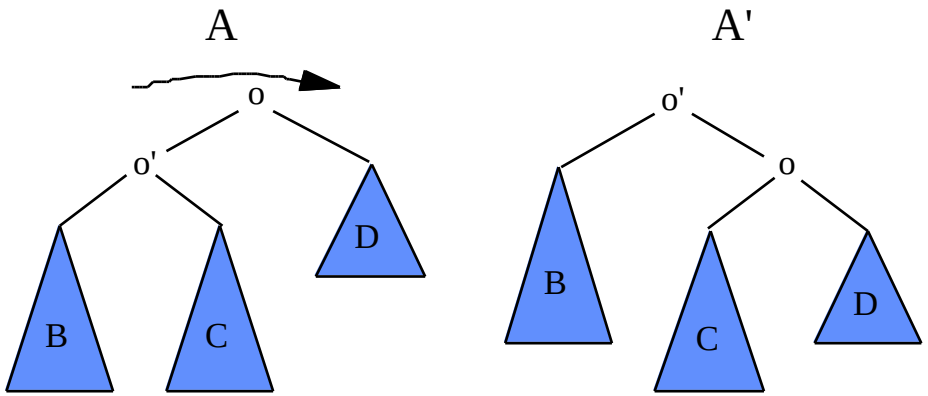
rotation gauche-droite

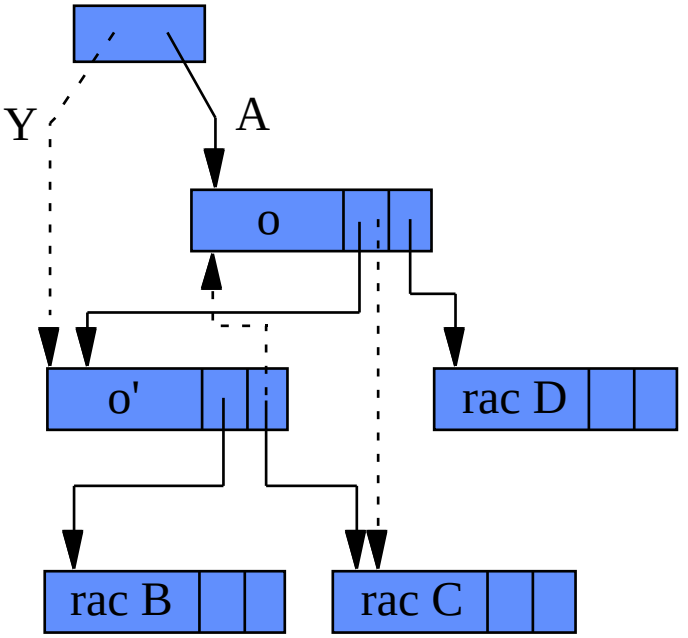


- Propriétés:
  - A arbre binaire de recherche => A' arbre binaire de recherche
  - si B, C, D, E sont H-équilibré => A' est H-équilibré

déséq(A)=2, déséq(g(A))=-1 => rotation gauche-droite

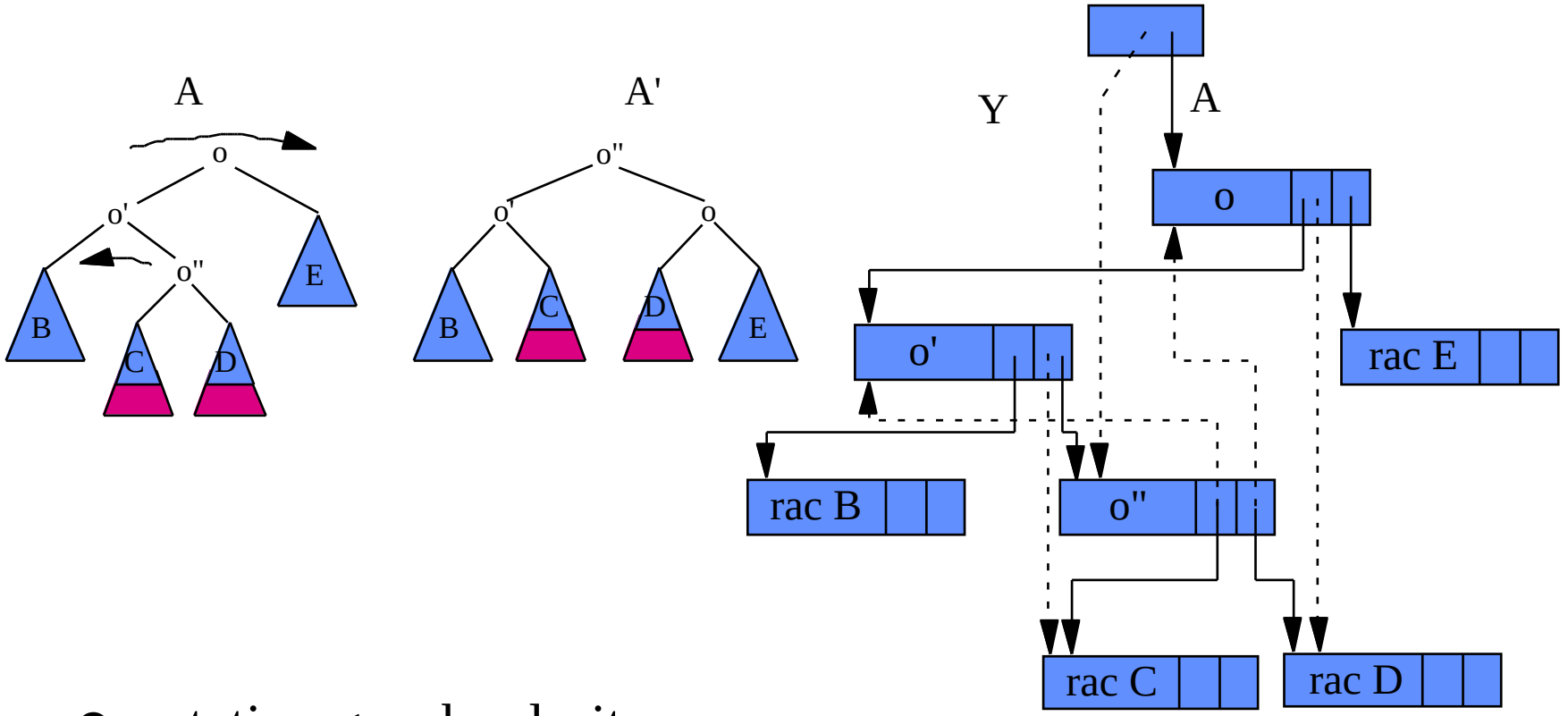
Implantation des rotations (1)





- rotation à droite:
  - Y := A.G; A.G := Y.D; Y.D := A; A := Y;

Implantation des rotations (2)



- rotation gauche-droite
  - Y := A.G.D, A.G.D := Y.G; Y.G := A.G; A.G := Y.D; Y.D := A; A := Y;

Rééquilibrage d'un arbre

- A partir des opérations de rotation, on définit le rééquilibrage:

extension type arbre
opérations
rééquilibrage : arbre → arbre
sémantique
rééquilibrage (a) = **si** déséquilibre(a)=2 déséquilibre(g(a))=1 **alors** rd(a)
sinsi déséquilibre(a)=2 déséquilibre(g(a))=0 **alors** rd(a)
sinsi déséquilibre(a)=2 déséquilibre(g(a))=-1 **alors** rgd(a)
sinsi déséquilibre(a)=-2 déséquilibre(d(a))=-1 **alors** rg(a)
sinsi déséquilibre(a)=-2 déséquilibre(d(a))=0 **alors** rg(a)
sinsi déséquilibre(a)=-2 déséquilibre(d(a))=1 **alors** rdg(a)
sinon a
fsi

- On ne rencontrera pas les autres cas de déséquilibres

Adjonction

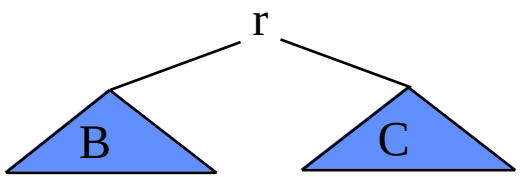
- Conservation hauteur

  ajout dans B, h(B') = h(B)

  ajout dans C, h(C') = h(C)

  => A' est H-équilibré de même hauteur

A =



- Augmentation hauteur

déséq A →	-1	0	+1
ajout dans B h(B')=h(B)+1	0 h(A') = h(A)	+1 h(A')=h(A)+1	+2 h(A')=h(A)+1
ajout dans C h(C')=h(C)+1	-2 h(A')=h(A)+1	-1 h(A')=h(A)+1	0 h(A') = h(A)

rotation à gauche ou droite-gauche
déséquilibre (C') = -1 ou +1

rotation à droite ou gauche-droite
déséquilibre (B') = -1 ou +1

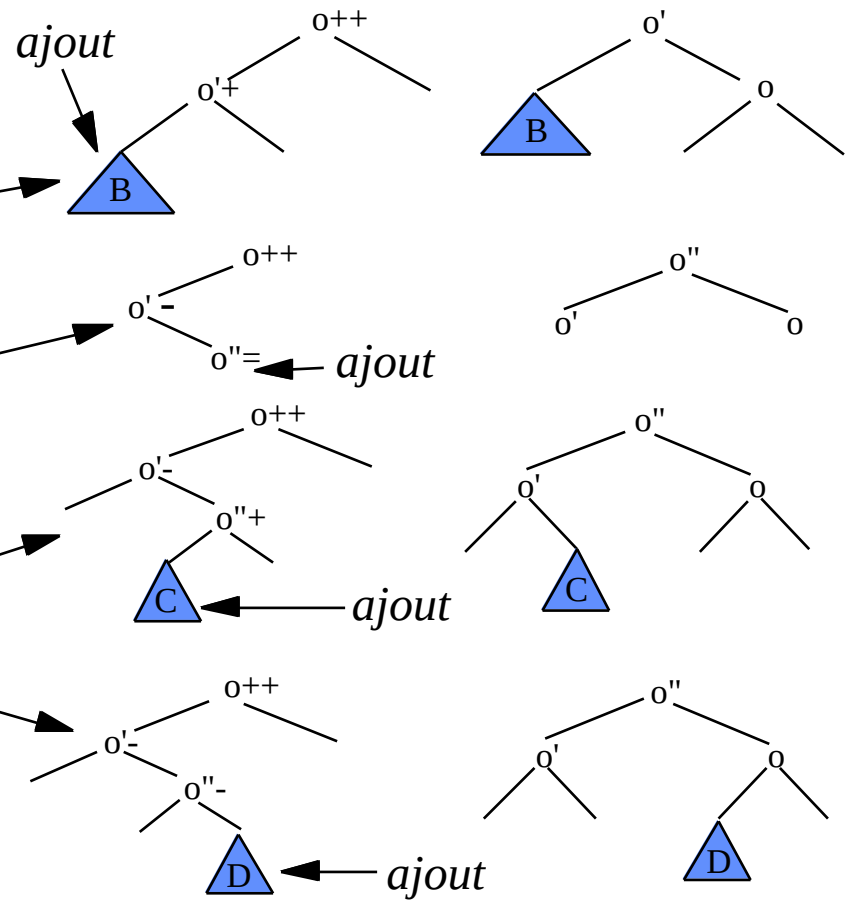
=> h(A'') = h(A')-1 = h(A) donc au plus une rotation

Spécification de l'adjonction

extension type arbre-avl
opérations
ajout-avl : nœud × arbre-avl /→ arbre-avl
préconditions
ajout-avl(o, a): ¬ (la_clé(o) a)
sémantique
ajout-avl(o, a) = **si** a = Ø **alors** <o, Ø, Ø>
sinon soit a = <r, g, d>;
si la_clé(o) < la_clé(r) **alors**
rééquilibrage(<r, ajout-avl(o, g), d>)
sinon rééquilibrage(<r, g, ajout-avl(o, d)>) **fsi**
fsi

Ajustement du chemin

- Une rotation change le chemin de la racine à la feuille:
  - rotation à droite:
    o disparait du chemin
  - rotation gauche-droite, o" est la feuille:
    o et o' disparaissent du chemin
  - rotation gauche-droite, ajout dans C:
    o" prend la place de o' et o disparait
  - rotation gauche-droite, ajout dans D:
    o' disparait et o" passe au dessus de o
  - les déséquilibres permettent de distinguer les cas



Implantation de l'adjonction

- Recherche de la place de la feuille
- création de la feuille, et rattachement à l'extrêmité du chemin
- remontée du chemin
  - correction du déséquilibre
  - ajustement du chemin éventuel
  - rééquilibrage éventuel
  - arrêt si le déséquilibre est devenu nul sur le nœud
- complexité: en $\Theta(\log_2 n)$ pour comparaisons, modifications du déséquilibre, au plus une rotation

Suppression

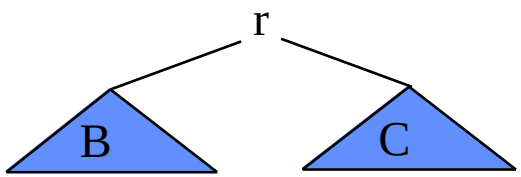
- Conservation hauteur

  suppression dans B, h(B') = h(B)

  suppression dans C, h(C') = h(C)

  => A' est H-équilibré de même hauteur

A =



- Diminution de hauteur

déséq A →	-1	0	+1
suppr dans B h(B')=h(B)-1	-2 /h(A') = h(A)	-1 h(A')=h(A)	0 h(A')=h(A)-1
suppr dans C h(C')=h(C)-1	0 h(A')=h(A)-1	+1 h(A')=h(A)	+2 h(A') = h(A)

rotation à gauche ou droite-gauche
déséquilibre (C) = -1, +1 ou 0

rotation à droite ou gauche-droite
déséquilibre (B) = -1, +1 ou 0

h(A") = h(A)-1 => poursuite

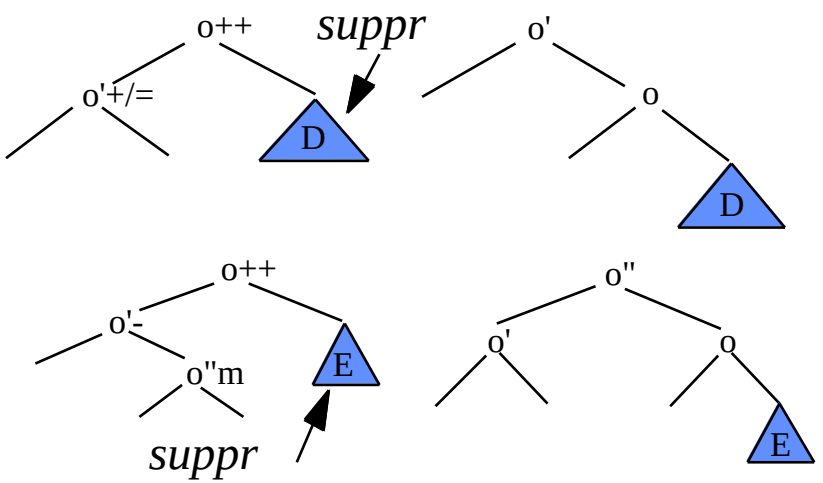
h(A") = h(A) => arrêt

Spécification de la suppression

extension type arbre-avl
opérations **préconditions**
sup-gauche-avl : arbre-avl /→ arbre-avl a ≠ Ø
supprimer-avl : clé × arbre-avl /→ arbre-avl c a
sémantique
sup-gauche-avl(<o, g, d>) = **si** g = Ø **alors** d
sinon rééquilibrage(<o, sup-gauche-avl(g), d>) **fsi**
supprimer-avl(c, <o, g, d>) =
si c < la_clé(o) **alors** rééquilibrage(<o, supprimer-avl(c, g), d>)
sinsi c > la_clé(o) **alors** rééquilibrage(<o, g, supprimer-avl(c, d)>)
sinsi g = Ø **alors** d
sinsi d = Ø **alors** g
sinon rééquilibrage(<gauche(d), g, sup-gauche-avl(d)>)
fsi

Ajustement du chemin

- rotation est en dessous du nœud
  - pas de changement du chemin
- rotation sur le chemin de la racine au nœud:
  - rotation à droite:
    o' apparait au dessus de o, avec un sens "à droite"
  - rotation gauche-droite:
    o" apparait au dessus de o, avec un sens "à droite"
  - les déséquilibres permettent de distinguer les cas



Implantation de la suppression

- suppression du nœud courant (extrêmité du chemin)
- si le nœud a moins de deux fils
  - ♦ suppression du nœud et recherche du suivant dans l'ordre infixe
- si le nœud a deux fils
  - ♦ recherche extrêmité du bord gauche du sous arbre droit
  - ♦ échange des valeurs
  - ♦ suppression de l'extrêmité
- remonter le chemin
  - ♦ corriger le déséquilibre, ajuster le chemin et rééquilibrer
- complexité: en $\Theta(\log_2 n)$ pour comparaisons, modifications du déséquilibre et rotations

Conclusion

- Dans les arbres AVL,
  recherche
  adjonction
  suppression
  ont une complexité au pire en $\Theta(\log_2 n)$
- On constate expérimentalement:
  en moyenne 1 rotation pour 2 adjonctions
  en moyenne 1 rotation pour 5 suppressions
  => la suppression n'est pas plus coûteuse que l'adjonction!