Cours 7 : Structures de données arborescentes partie 2

Jean-Stéphane Varré

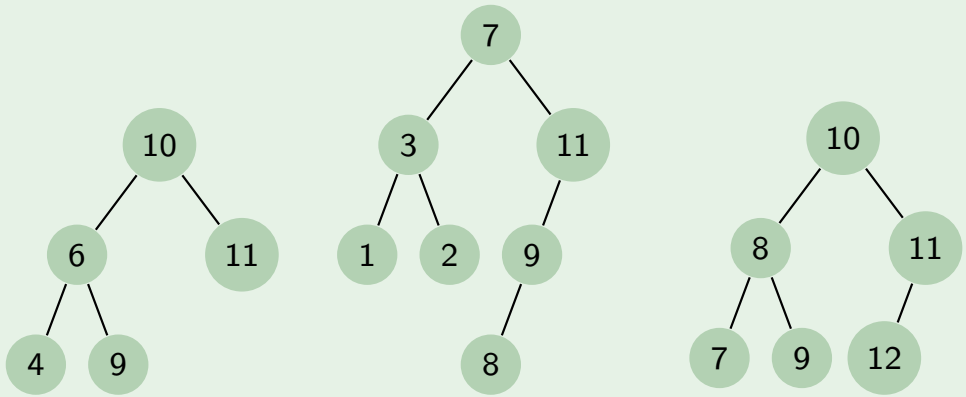
Université Lille 1

jean-stephane.varre@lifl.fr

Rappels sur les arbres binaires de recherche

- soit l'arbre vide
- soit un arbre binaire dont la valeur v associée à chaque noeud n est telle que :
  - toutes les valeurs des noeuds du sous-arbre gauche de n sont inférieures à v
  - toutes les valeurs des noeuds du sous-arbre droit de n sont supérieures à v
  - les sous-arbres gauche et droit de n sont des arbres binaires de recherche

Lesquels sont des ABR ?



Prédicat de validité d'un ABR

```
function estABOb (a : ARBRE) : BOOLEAN;
   // CU a n'est pas un arbre vide
   procedure xestABO(a : ARBRE; out max : ELEMENT; out min : ELEMENT; out res : BOOL
   var
      maxg, ming, maxd, mind : ELEMENT;
      abog, abod : BOOLEAN;
   begin
      abog := true; maxg := racine(a); ming := racine(a);
      if not estArbreVide(gauche(a)) then xestABO(gauche(a),maxg,ming,abog);
      abod := true; maxd := racine(a); mind := racine(a);
      if not estArbreVide(droit(a)) then xestABO(droit(a),maxd,mind,abod);
      max := Math.max(racine(a),Math.max(maxg,maxd));
      min := Math.min(racine(a),Math.min(ming,mind));
      res := abod and abog and (maxg <= racine(a)) and (racine(a) <= mind);
   end {estABO};

var
   maxg, mind : ELEMENT;
   res : BOOLEAN = true;
begin
   if not estArbreVide(a) then
      xestABO(a,maxg, mind, res);
   estABOb := res;
end;
```

Recherche dans un arbre binaire

```
// CU : l'arbre ne contient pas deux fois le meme element
function estPresent (x : ELEMENT; a : ABR) : BOOLEAN;
begin
   if estArbreVide (a) then
      estPresent := False
   else
      if x = racine(a) then
         estPresent := True
      else
         if x < racine(a) then
            estPresent := estPresent(x,gauche(a))
         else
            estPresent := estPresent(x,droit(a));
end {estPresent};
```

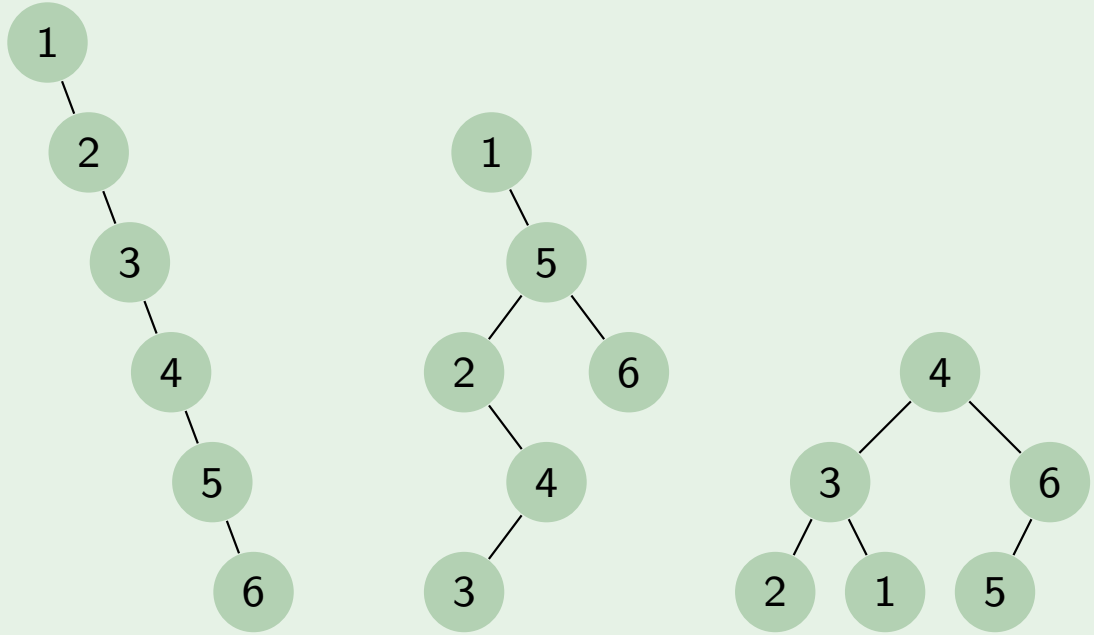
Complexité de la recherche

- meilleur des cas :
  - l'élément recherché est à la racine,
  - $\Omega(1)$
- pire des cas :
  - on parcours une et une seule branche de l'arbre,
  - la plus longue branche est de longueur la hauteur h de l'arbre a
  - $\mathcal{O}(\log h)$

la meilleure performance sera atteinte si l'arbre est équilibré ($h = \mathcal{O}(\log n)$)

Construction d'un arbre binaire

même algorithme que la recherche + création d'une nouvelle feuille



ordre d'insertion	1,2,3,4,5,6	1,5,2,4,3,6	4,3,2,1,6,5

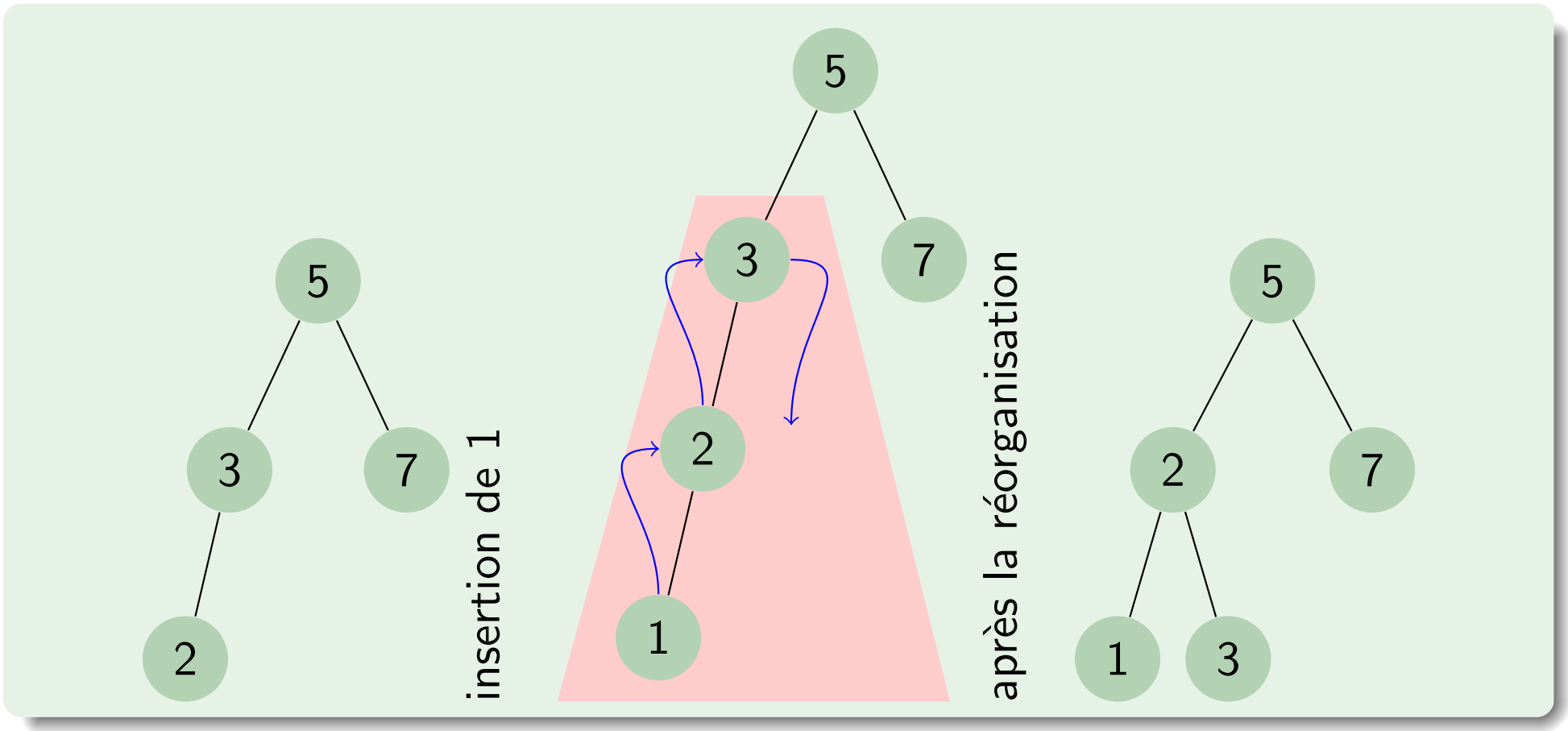
ne conserve pas une hauteur minimale !

Maintien de la hauteur minimale

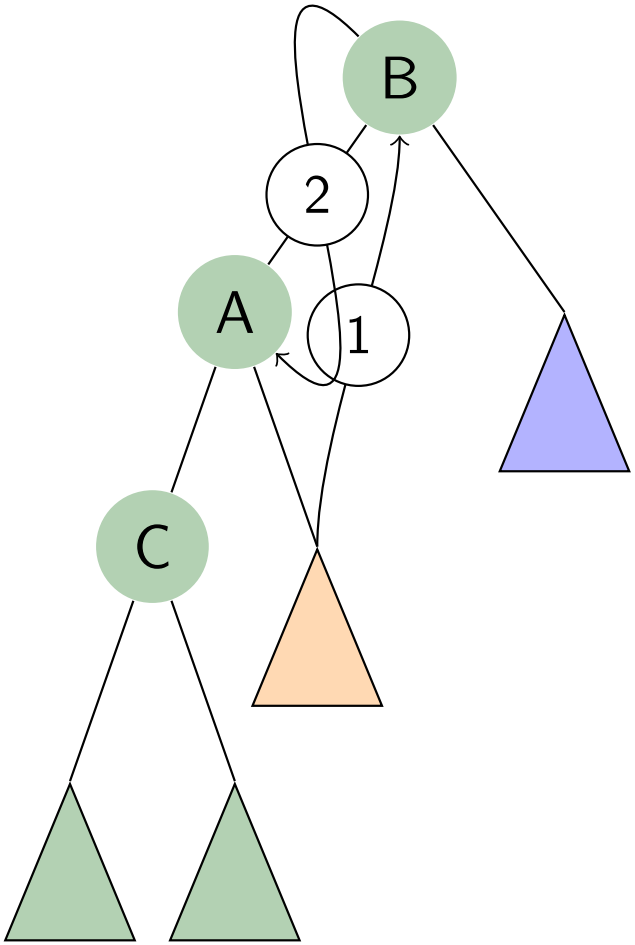
- nécessite de conserver une différence de hauteur de au plus un entre les deux sous-arbres gauche et droit
- ceci ne peut être fait avec la construction d'un ABR : on n'a pas de le choix du sous-arbre dans lequel on insère
- on pourrait réorganiser les valeurs avant de construire l'arbre, mais cela revient à trier, l'utilisation d'un ABR par la suite devient inutile

inventer une nouvelle manière d'insérer les valeurs

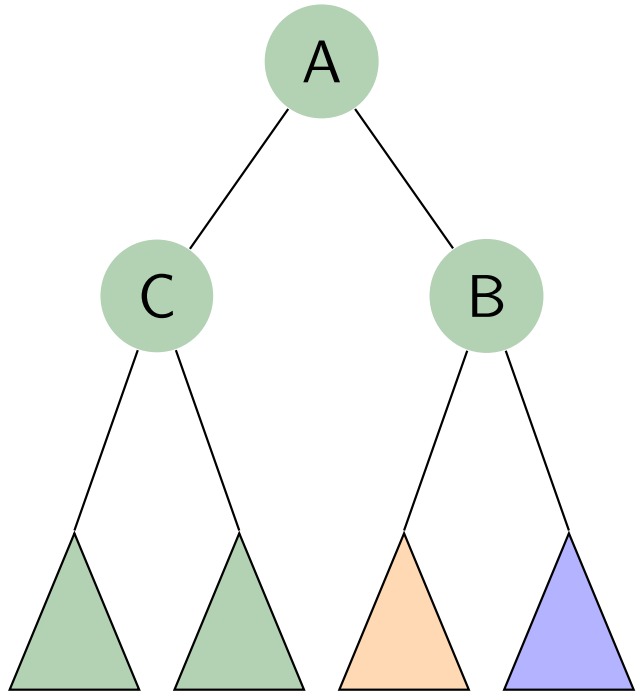
Exemple



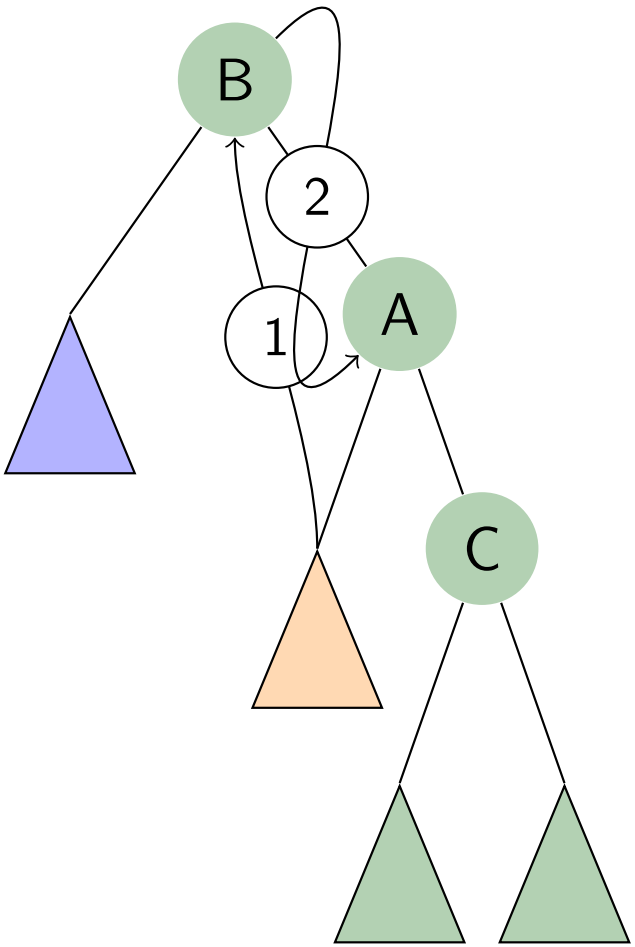
La rotation droite



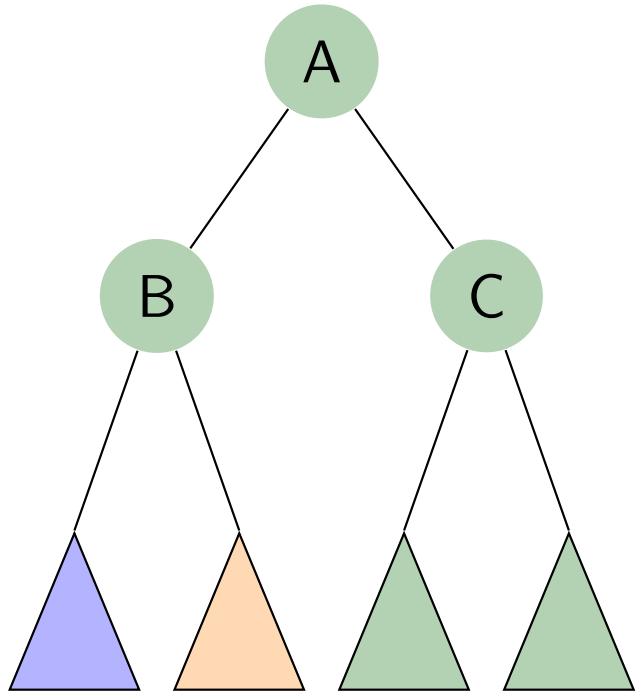
1. le fils droit de A remplace le fils gauche de B
2. le sous-arbre B devient le fils droit de A
3. A devient la nouvelle racine



La rotation gauche



1. le fils gauche de A remplace le fils droit de B
2. le sous-arbre B devient le fils gauche de A
3. A devient la nouvelle racine



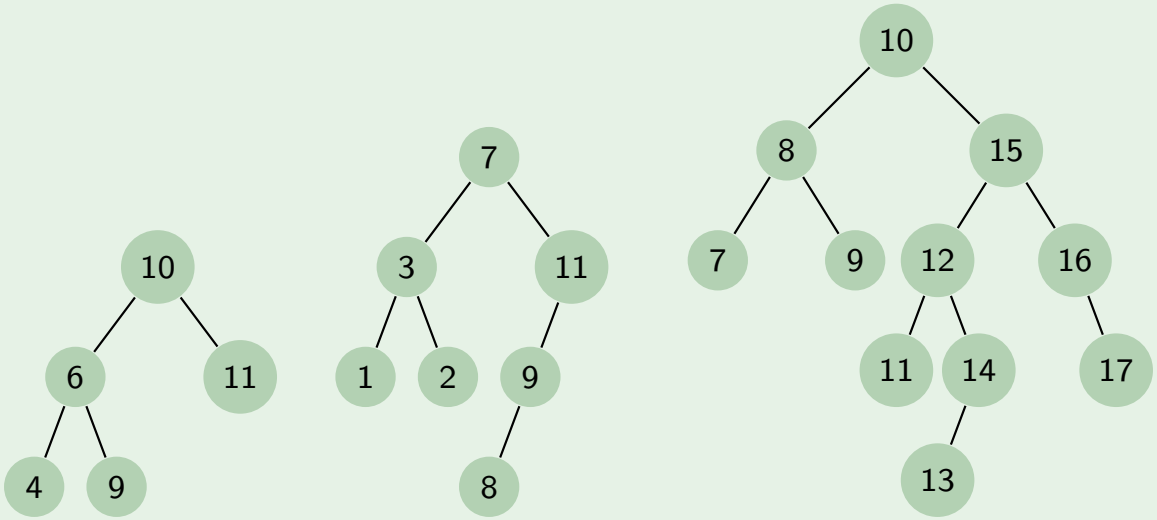
Les AVL

AVL = G.M. **A**del'son-**V**elski et E.M. **L**andis

Définition : un AVL est un arbre binaire, éventuellement vide, tel que

- la différence de hauteur entre les sous-arbres gauche et droit d'un noeud ont une différence de hauteur d'au plus un
- les sous-arbres gauche et droit sont des AVL

Lesquels sont des AVL ?



Propriétés dans les AVL

- si n est la taille de l'arbre, alors la hauteur est $\lfloor \log_2(n) \rfloor$
- l'insertion d'une valeur dans l'arbre aboutit à l'ajout d'une feuille et donc à un déséquilibre de hauteur d'au plus 2
- la suppression d'une valeur dans l'arbre aboutit à la suppression d'une feuille ou d'un noeud et donc à un déséquilibre de hauteur d'au plus 2

On définit pour chaque noeud une valeur de déséquilibre :

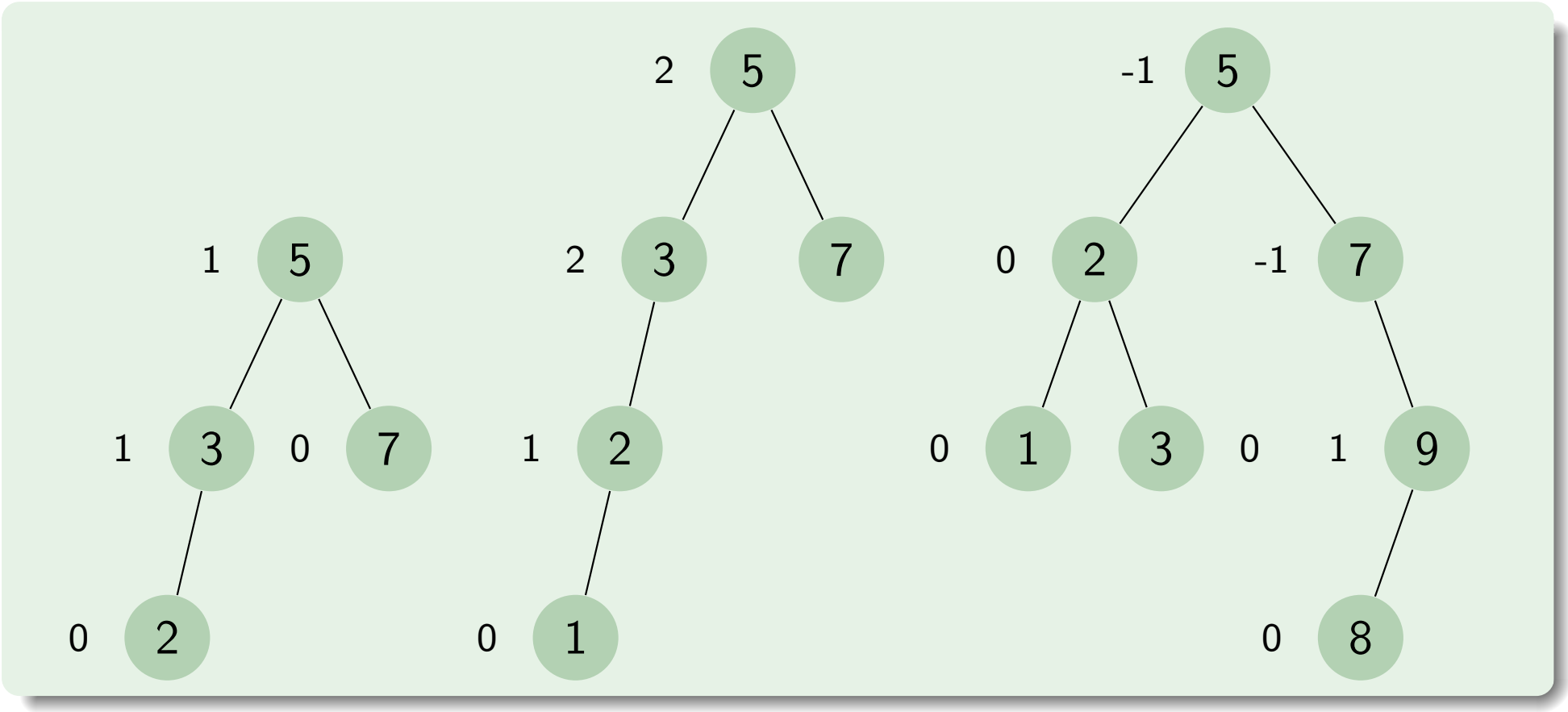
$$d(n) = h(\text{fils gauche}(n)) - h(\text{fils droit}(n))$$

qui vaut zéro pour une feuille.

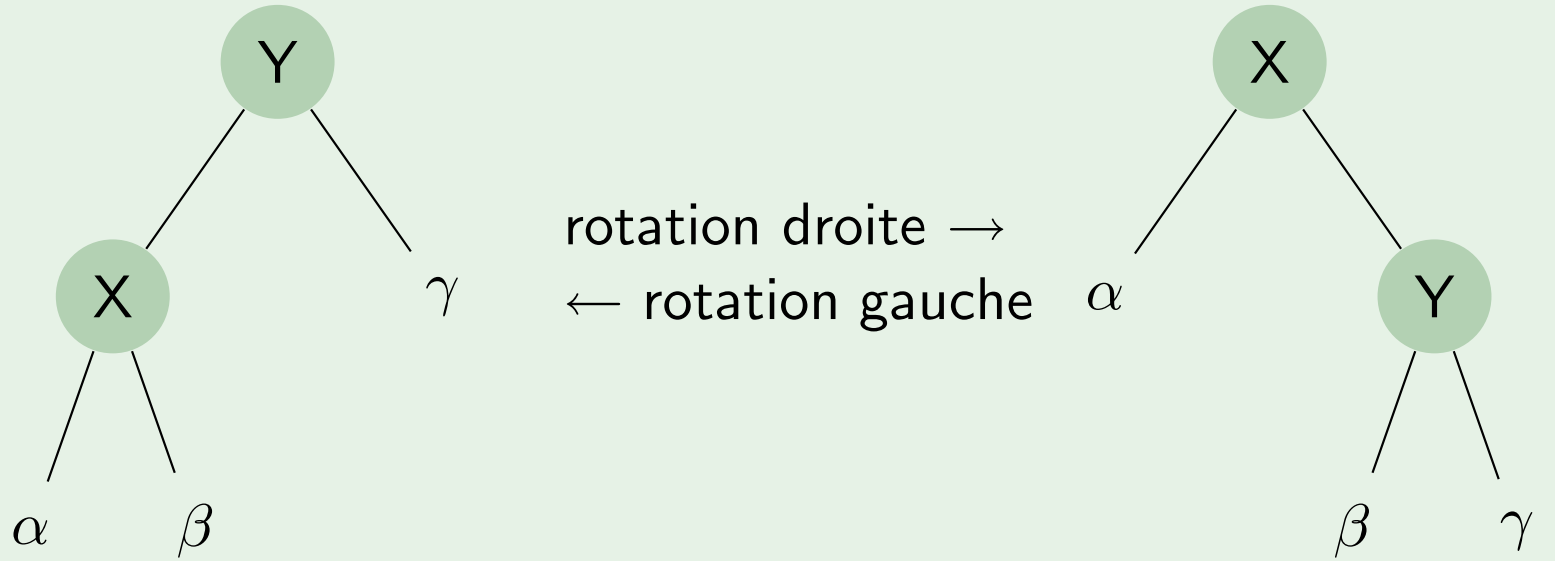
Dans un AVL, le déséquilibre de chaque noeud vaut -1, 0 ou $+1$.

- déséquilibre positif: h de gauche $>$ h de droite
- déséquilibre négatif: h de gauche $<$ h de droite

Exemples



Propriétés des rotations



- Après une rotation droite autour du sommet Y, on a :
  - $d'(X) = d(X) - 1 + \min(d'(Y), 0)$
  - $d'(Y) = d(Y) - 1 - \max(d(X), 0)$
- Après une rotation gauche autour du sommet X, on a :
  - $d'(X) = d(X) + 1 - \min(d(Y), 0)$
  - $d'(Y) = d(Y) + 1 + \max(d'(X), 0)$

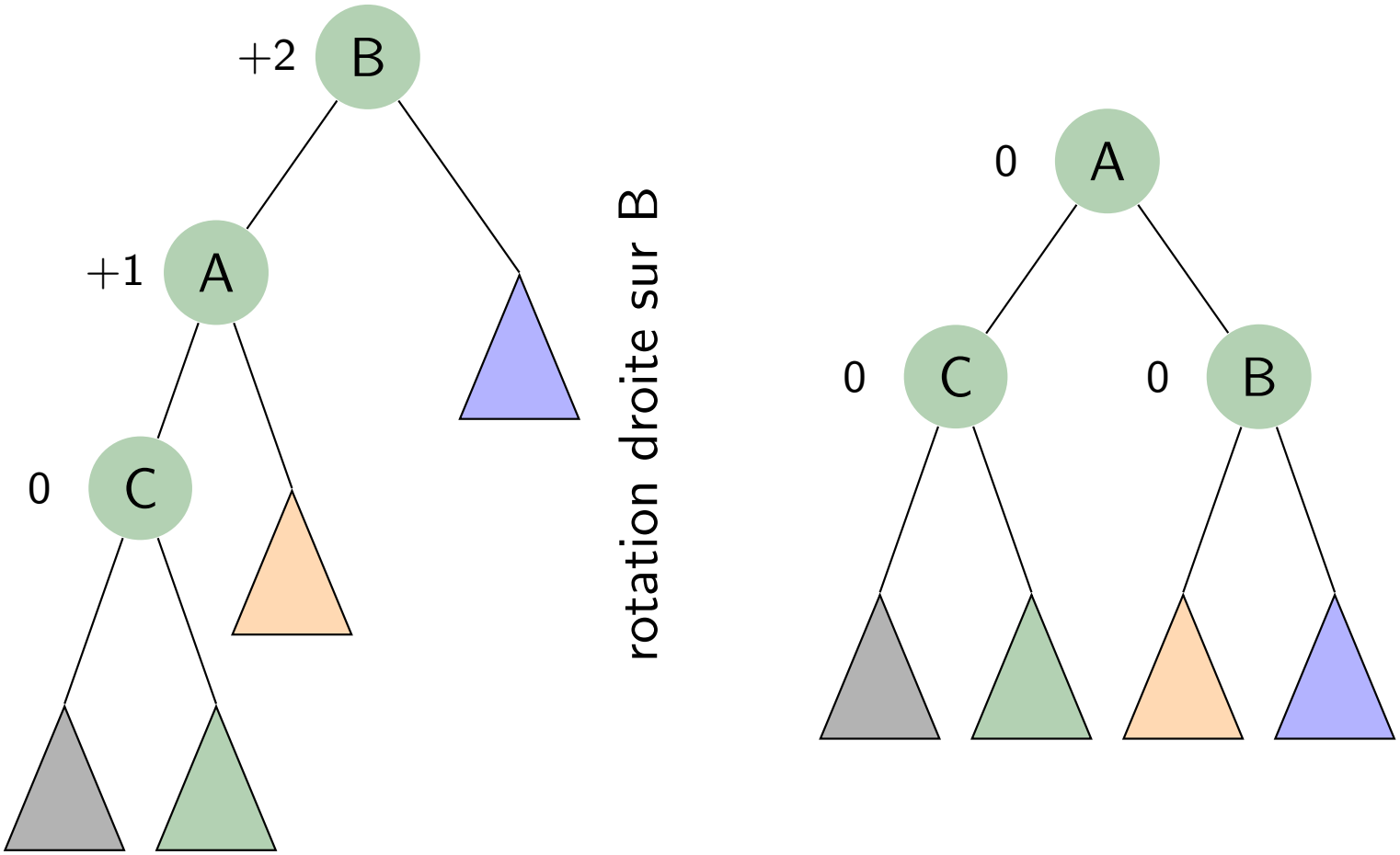
Insertion dans un AVL, cas simples

- si il n'y a pas de déséquilibre, alors la hauteur du sous-arbre gauche ou droit est augmentée au plus de 1
- si le déséquilibre de la racine est $+1$ et que l'ajout se fait dans le sous-arbre droit, alors la hauteur du sous-arbre droit après insertion est augmentée d'au plus 1
- si le déséquilibre de la racine est -1 et que l'ajout se fait dans le sous-arbre gauche, alors la hauteur du sous-arbre gauche après insertion est augmentée d'au plus 1

Dans tous ces cas, l'arbre reste équilibré

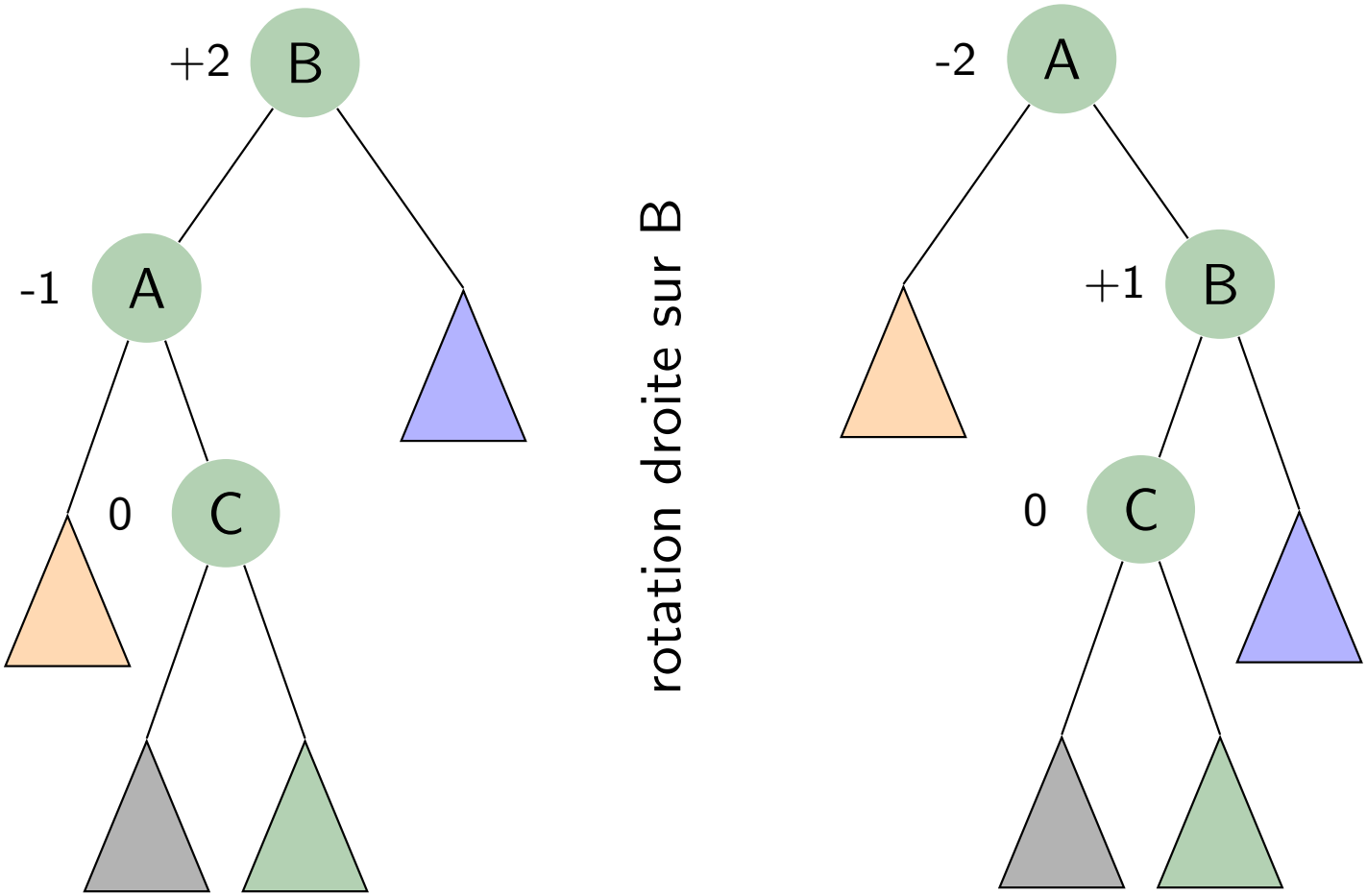
Insertion dans un AVL, cas 1a

Déséquilibre de +1, ajout dans le fils gauche, déséquilibre résultant de +2, à gauche du fils gauche



Insertion dans un AVL, cas 1b

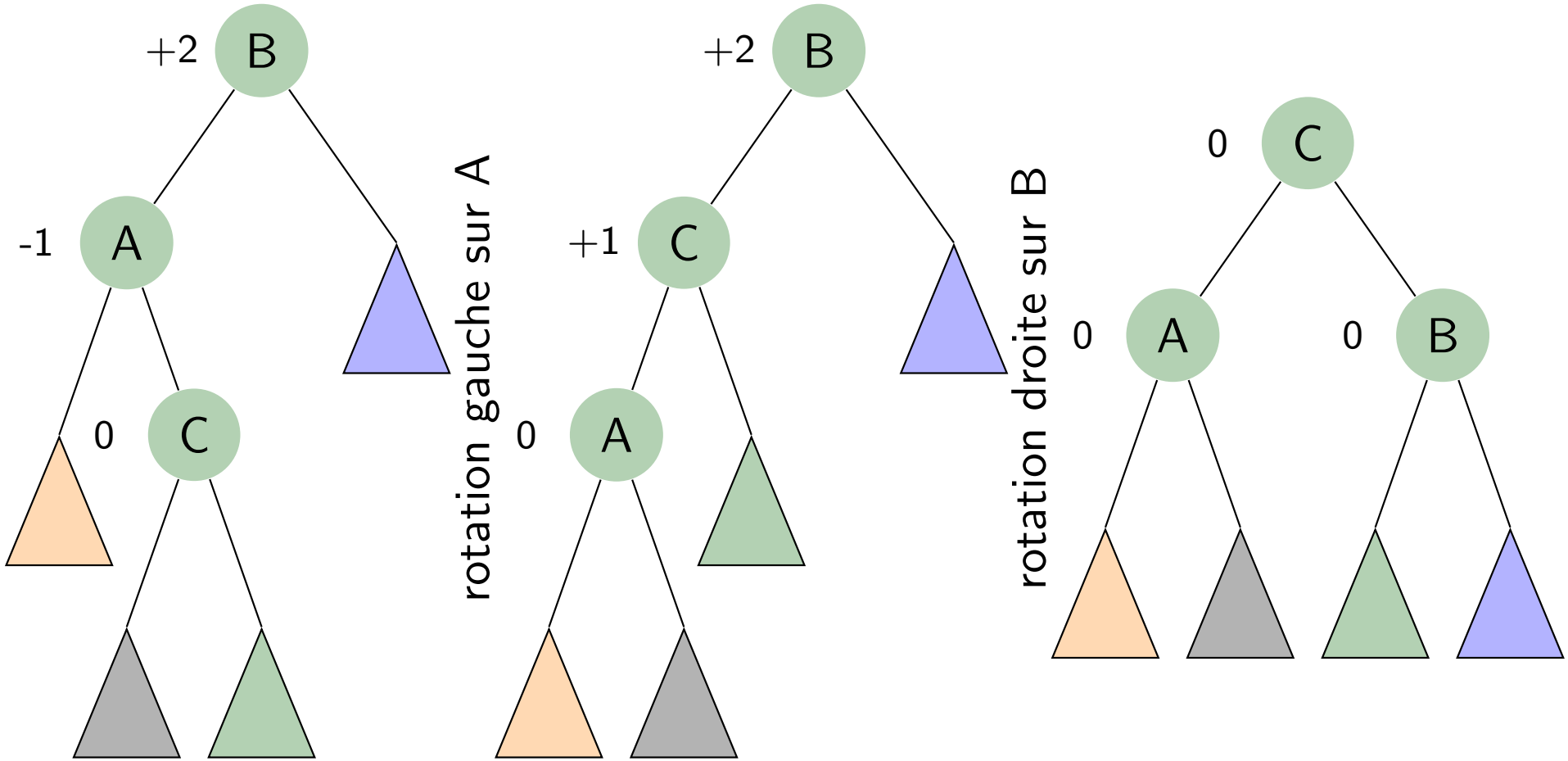
Déséquilibre de +1, ajout dans le fils gauche, déséquilibre résultant de +2, à droite du fils gauche



Ne fonctionne pas !

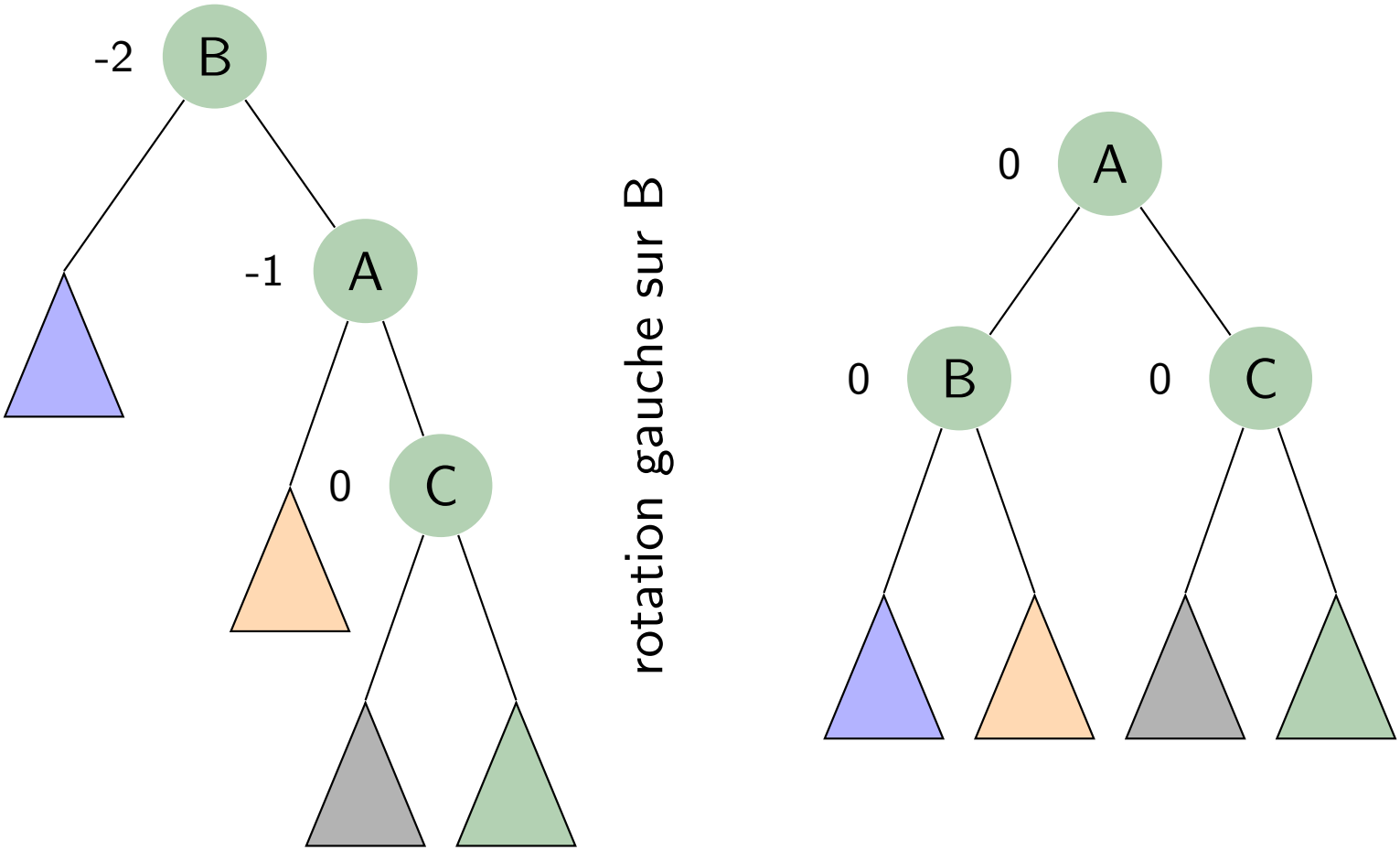
Insertion dans un AVL, cas 1b

Déséquilibre de +1, ajout dans le fils gauche, déséquilibre résultant de +2, à droite du fils gauche



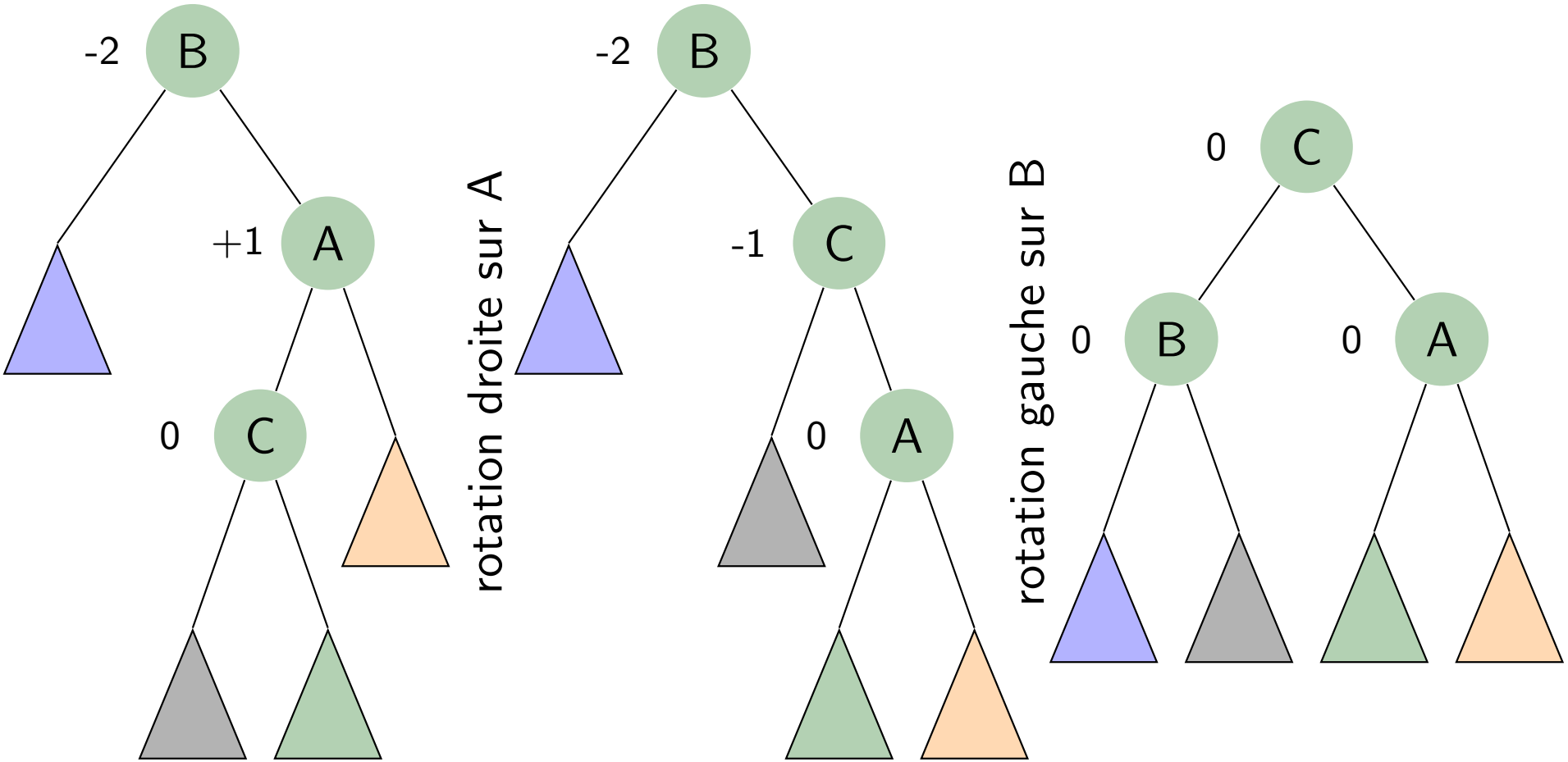
Insertion dans un AVL, cas 2a

Déséquilibre de -1, ajout dans le fils droit, déséquilibre résultant de -2, à droite du fils droit



Insertion dans un AVL, cas 2b

Déséquilibre de -1, ajout dans le fils droit, déséquilibre résultant de -2, à gauche du fils droit



Implémentation

```
type
   ELEMENT = CARDINAL;
   AVL = ^NOEUD;
   NOEUD = record
      valeur : ELEMENT;
      filsg : AVL;
      filsd : AVL;
      hauteur : INTEGER;
   end {NOEUD};

   function hauteur (a : AVL) : INTEGER;
   begin
      if a = NIL then
         hauteur := -1
      else
         hauteur := a.hauteur;
   end {hauteur};
```

Rotations

```
procedure rotation_gauche (var a : AVL);
var
   d : AVL;
begin
   d := a.filsd;
   a.filsd := d.filsg;
   d.filsg := a;
   a := d;
   a.filsg.hauteur := max(hauteur(a.filsg.filsg),hauteur(a.filsg.filsd)) + 1;
   a.hauteur := max (hauteur(a.filsg),hauteur(a.filsd)) + 1;
end {rotation_gauche};

procedure rotation_droite (var a : AVL);
var
   g : AVL;
begin
   g := a.filsg;
   a.filsg := g.filsd;
   g.filsd := a;
   a := g;
   a.filsd.hauteur := max(hauteur(a.filsd.filsg),hauteur(a.filsd.filsd)) + 1;
   a.hauteur := max (hauteur(a.filsg),hauteur(a.filsd)) + 1;
end {rotation_droite};
```

Insertion

- insérer comme dans les ABR
- mettre à jour la hauteur de chaque noeud visité le long du chemin
- ré-équilibrer chaque noeud visité

```
procedure inserer (var a : AVL; valeur : ELEMENT);
begin
   if a = NIL then begin
      new(a);
      a.valeur := valeur;
      a.filsd := NIL;
      a.filsg := NIL;
      a.hauteur := 1;
   end else begin
      if valeur < a.valeur then
         inserer(a.filsg,valeur)
      else
         inserer(a.filsd,valeur);
      a.hauteur := max(hauteur(a.filsg),hauteur(a.filsd)) + 1;
      equilibrer(a);
   end {if};
end {inserer};
```

Ré-équilibrage

```
procedure equilibrer (var a : AVL);
begin
   if hauteur(a.filsg) - hauteur(a.filsd) = 2 then begin
      if hauteur(a.filsg) < hauteur(a.filsd) then
         rotation_gauche(a.filsg);
      rotation_droite(a);
   end else if hauteur(a.filsg) - hauteur(a.filsd) = -2 then begin
      if hauteur(a.filsd) < hauteur(a.filsg) then
         rotation_droite(a.filsd);
      rotation_gauche(a);
   end {if};
end {equilibrer};
```

Complexité de l'insertion

- complexité d'une rotation : $\Theta(1)$
- complexité d'un équilibrage d'un noeud : $\Theta(1)$ (au maximum 2 rotations)
- complexité de l'insertion : $\mathcal{O}(\log n)$ (au maximum un équilibrage par noeud traversé)