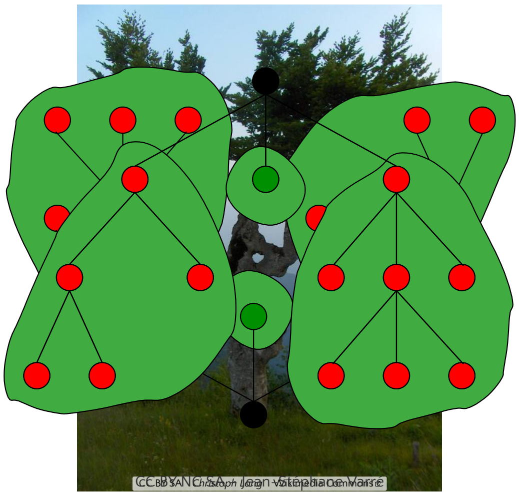
Structure de données arborescentes

L2 informatique
Univ. Lille
Cours qui s'appuie largement sur un cours de Jean-Stéphane Varré

Mikaël Salson
mikael.salson@univ-lille.fr

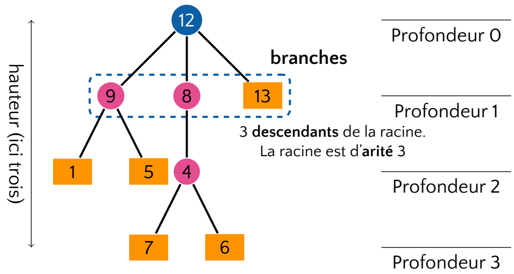


Qu'est-ce qu'un arbre ?

Structure de données **hiérarchique** récursive

Utilisée pour hiérarchiser des données
- arbres généalogiques
- système de fichiers
- XML
- ...

Un peu de vocabulaire sur les arbres



On parle parfois de nœuds pour désigner à la fois les nœuds internes et les feuilles

Un peu de vocabulaire sur les arbres

- **racine :** seul nœud n'ayant pas de parent
- **feuille :** nœud n'ayant aucun descendant
- **nœud interne :** nœud qui n'est pas une feuille
- **arité** d'un nœud : nombre de descendants d'un nœud
- **arité** d'un arbre : nombre de descendants maximal d'un nœud de l'arbre
- **branche :** lien entre un nœud et son parent (ou un nœud et son descendant)
- **chemin :** succession de branches à emprunter pour se rendre d'un nœud n_1 à un nœud n_2 (n_1 est forcément un parent, éventuellement éloigné, de n_2)
- **profondeur** d'un nœud : longueur du chemin entre deux nœuds (*ie*. le nombre de branches empruntées)
- **hauteur** d'un arbre : la profondeur maximale d'un nœud dans l'arbre

Arbre binaire

Arbre binaire

- soit l'arbre vide, noté Δ ;
- soit un triplet (e, g, d), appelé *nœud*, dans lequel
  - e est l'élément, ou encore *étiquette*, de la racine de l'arbre ;
  - g est le sous-arbre gauche de l'arbre (c'est un arbre binaire) ;
  - et d est le sous-arbre droit de l'arbre (c'est un arbre binaire).

Exemple d'arbres binaires

Δ

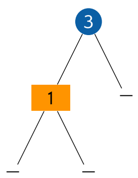
–

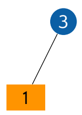
Exemple d'arbres binaires

$(3, \Delta, \Delta)$

Exemple d'arbres binaires

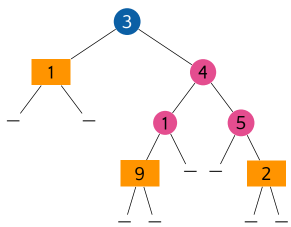
$(3, (1, \Delta, \Delta), \Delta)$

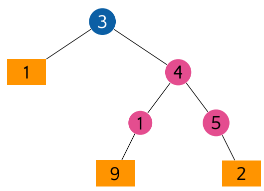




Exemple d'arbres binaires

$(3, (1, \Delta, \Delta), (4, (1, (9, \Delta, \Delta), \Delta), (5, \Delta, (2, \Delta, \Delta))))$





Nombre de nœuds et hauteur d'un arbre binaire

Soit un arbre binaire de hauteur n,
combien a-t-il de nœuds ?

Au moins $n+1$ et au plus $2^{n+1}-1$
(donc $\Omega(n)$ et $\mathcal{O}(2^n)$)

Soit un arbre binaire de n nœuds,
quelle est sa hauteur ?

Au moins $\lfloor \log_2 n \rfloor$ et au plus $n-1$
(donc $\Omega(\log n)$ et $\mathcal{O}(n)$)

Type de données abstrait arbre

Opérations fondamentales sur un arbre binaire

- Construire un arbre vide
- Construire un arbre à un nœud
- Accéder à la valeur de la racine
- Accéder au sous-arbre gauche
- Accéder au sous-arbre droit
- Savoir si un arbre est vide

Structure de données : par exemple récursive, sur le même modèle qu'une liste chainée, mais avec deux références `droit` et `gauche` au lieu d'avoir une référence vers l'élément suivant uniquement

Notez qu'une liste peut être vue comme un arbre d'arité 1

Comment parcourir un arbre binaire ?

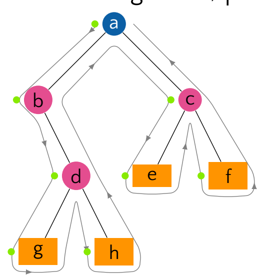
Deux manières de parcourir un arbre binaire

En profondeur commencer par explorer les profondeurs

En largeur explorer les nœuds présents à un même niveau

Parcours en profondeur

À partir de la racine, parcourir le sous-arbre gauche, puis le sous-arbre droit

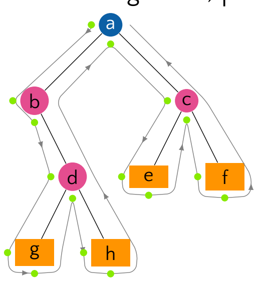


Parcours préfixe
lister le nœud courant, plonger à gauche puis à droite

a, b, d, g, h, c, e, f

Parcours en profondeur

À partir de la racine, parcourir le sous-arbre gauche, puis le sous-arbre droit

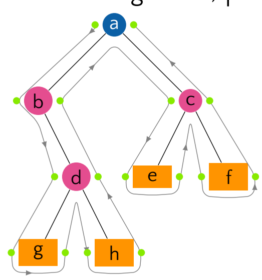


Parcours infixe
plonger à gauche, lister le nœud courant, plonger à droite

b, g, d, h, a, e, c, f

Parcours en profondeur

À partir de la racine, parcourir le sous-arbre gauche, puis le sous-arbre droit



Parcours suffixe
plonger à gauche, puis à droite, puis lister le nœud courant

g, h, d, b, e, f, c, a

Algorithme des parcours en profondeur

```
Entrées : a: arbre
Procédure AffichagePrefixe (a)
  si a n'est pas vide alors
    Afficher a.valeur;
    AffichagePrefixe (a.gauche);
    AffichagePrefixe (a.droite);
  fin
fin
```

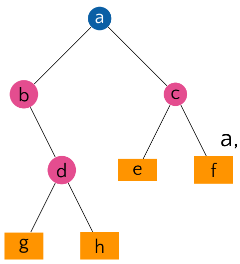
Et sans récursivité ?

Algorithme non récursif de parcours en profondeur

```
Entrées : a: arbre
Procédure AffichagePrefixe (a)
  p: une pile d'arbres;
  Empiler a dans p;
  tant que p n'est pas vide faire
    s ← sommet de p;
    Dépiler p;
    si s n'est pas vide alors
      Afficher s.valeur;
      Empiler s.droite dans p;
      Empiler s.gauche dans p;
    fin
  fin
fin
```


Parcours en largeur

Parcours de chaque niveau de gauche à droite



a, b, c, d, e, f, g, h

Comment faire ?

Algorithme du parcours en largeur

```
Entrées : a: arbre
Procédure AffichageLargeur (a)
  f: une file d'arbres;
  Enfiler a dans f;
  tant que f n'est pas vide faire
    t ← tête de f;
    Défiler f;
    si t n'est pas vide alors
      Afficher t.valeur;
      Enfiler t.gauche dans f;
      Enfiler t.droite dans f;
    fin
  fin
fin
```


Autres algorithmes sur les arbres binaires

Comment calculer le nombre de nœuds dans un arbre ?

Comment calculer la hauteur d'un arbre ?

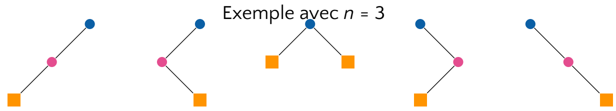
Construction d'arbres binaires

Combien a-t-on d'arbres différents à n nœuds ?

Exemple avec n = 1

Exemple avec n = 2

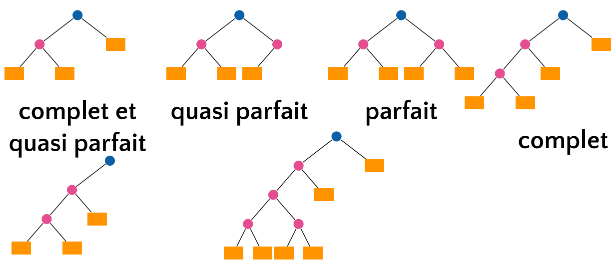
Exemple avec n = 3



$C_n = \sum_{i=0}^{n-1} C_i C_{n-i-1}$ possibilités (avec $C_0 = 1$), soit $\frac{(2n)!}{n!(n+1)!}$

Quelques arbres binaires caractéristiques

complet	**parfait**	**quasi-parfait**
tous les nœuds internes ont deux descendants	complet dont toutes les feuilles sont à la même profondeur	feuilles à profondeur $h-1$ ou h, tous les nœuds internes possèdent deux descendants (sauf éventuellement 1), feuilles de profondeur h à gauche



Arbres binaires de recherche

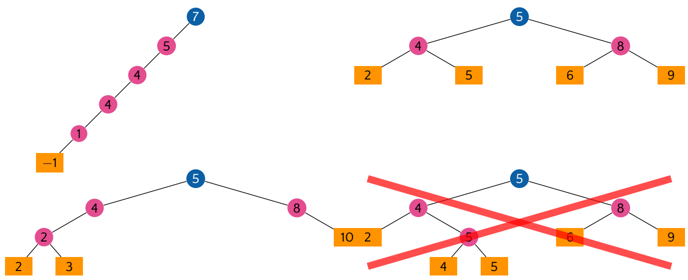
Arbres permettant de rechercher des éléments

Définition

Un **arbre binaire de recherche** (ou arbre **ordonné**) est soit :

- un arbre vide ;
- soit un arbre dont les deux sous-arbres sont des arbres binaires de recherche et l'étiquette à la racine est :
  - supérieure ou égale à toutes les étiquettes des nœuds du sous-arbre gauche ;
  - strictement inférieure à toutes les étiquettes des nœuds du sous-arbre droit.

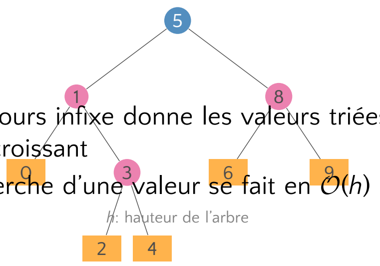
Exemples d'arbres binaires de recherche



Déterminer si un arbre binaire est un ABR

Entrées : a: arbre
Fonction EstABR *(a)*
  si *a est vide* **alors**
    retourner vrai;
  sinon
    si EstABR *(a.gauche)* ***et*** EstABR *(a.droite)* **alors**
      si *a.element* $\geq$ max(*a.gauche*)
      et *a.element* $<$ min(*a.droite*) **alors**
        retourner vrai;
      fin
    fin
    retourner faux;
  fin
fin

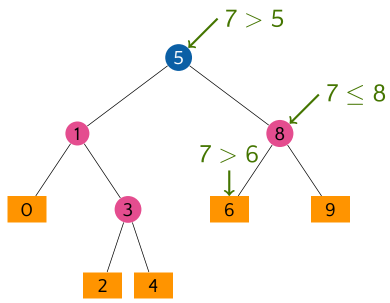
Propriétés des ABR



- Un parcours infixe donne les valeurs triées dans l'ordre croissant
- La recherche d'une valeur se fait en $\mathcal{O}(h)$

h: hauteur de l'arbre

Insertion dans un ABR



Où insérer la valeur 7 ?

Algorithme d'insertion dans un ABR

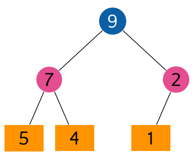
Entrées : a: arbre
Entrées : v: valeur à insérer
Fonction InsertABR *(a, v)*
  si *a est vide* **alors**
    retourner Créer(v, Δ, Δ);
  sinon
    si $v \leq a.element$ **alors**
      $a_g \leftarrow$ InsertABR (a.gauche, v);
      Remplacer a.gauche par a_g;
    sinon
      $a_d \leftarrow$ InsertABR (a.droite, v);
      Remplacer a.droite par a_d;
    fin
    retourner a;
  fin
fin

Structure de données : tas

Le tas

Définition

Un **tas** est un arbre binaire quasi-parfait qui est
- soit un arbre vide
- soit dont la valeur de la racine est maximale dans l'arbre et les sous-arbres gauche et droit sont également des tas



- La seconde valeur maximale est nécessairement un des deux descendants de la racine
- La hauteur d'un tas de n nœuds est $h = \lfloor \log_2 n \rfloor$

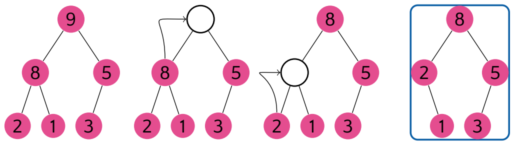
Trier avec un tas — le tri par tas

? **Comment faire ?**

Idée : extraire l'étiquette maximale, puis la seconde, etc.

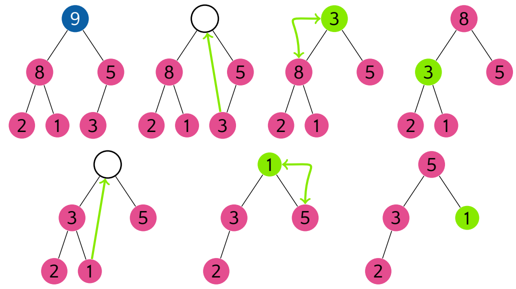
1. extraire la racine,
2. remonter la seconde étiquette maximale à la racine
3. recommencer récursivement

Exemple de remontée d'un maximum

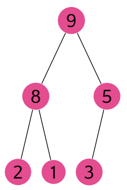


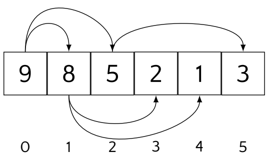
Ce n'est pas un tas !

Remplacement du maximum par la dernière feuille



Implantation d'un tas

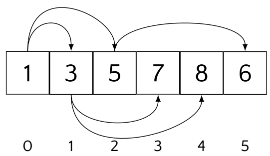


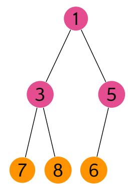


Les descendants d'un nœud représenté à l'indice i se trouvent en positions $2i + 1$ et $2i + 2$

Création d'un tas

Comment transformer ce tableau pour qu'il représente un tas ?





Algorithme de réorganisation d'un tas

Entrées : t: tas sous la forme d'un tableau
Entrées : pos: position dans le tas à laquelle réorganiser
Entrées : n: taille du tas tas
Fonction ReorganiserTas *(t, pos, n)*

```
fini = Faux;
pos_max = pos;
tant que non fini faire
    gauche = 2 × pos + 1;
    droite = 2 × pos + 2;
    si gauche < n et t[pos_max] < t[gauche] alors
        pos_max = gauche;
    fin
    si droite < n et t[pos_max] < t[droite] alors
        pos_max = droite;
    fin
    fini = (pos == pos_max);
    si non fini alors
        echanger (t, pos, pos_max);
    fin
    pos = pos_max;
fin
```
fin

Complexité de la création d'un tas

- Il faut itérer sur les nœuds internes (la moitié du tableau) : $\Theta\left(\frac{n}{2}\right) = \Theta(n)$
- Au pire il faut redescendre une valeur dans une feuille : $\mathcal{O}(\log n)$ … mais ce n'est pas toujours $\log_2 n$

- En fait on traite des sous-arbres de hauteur 1 à $\lfloor \log_2 n \rfloor$.
- Il y a plusieurs sous-arbres de hauteur 1, mais un seul de hauteur $\lfloor \log_2 n \rfloor$.
- Soit h la hauteur de l'arbre, il y a 2^{h-i} sous-arbres de hauteur i
- Chaque sous-arbre est réorganisé en $\mathcal{O}(i)$

$$\sum_{i=0}^{h} 2^{h-i} \times \mathcal{O}(i) = \mathcal{O}(n)$$

Le tri par tas

Le tas permet

- un accès en temps constant au maximum;
- la suppression du maximum.

Servons-nous de ces propriétés pour trier

Entrées : t: tableau
Fonction TriParTas *(t)*

```
heapify(t);            /* Transformation du tableau en tas */
longueur = longueur de t;
tant que longueur > 1 faire
    max = extraireMaximum(t);
    ReorganiserTas(t, 0, longueur);
    longueur− = 1;
    t[longueur] = max;
fin
```
fin

Complexités

- heapify : $\Theta(n)$
- extraireMaximum: $\Theta(1)$
- ReorganiserTas : $\mathcal{O}(\log n)$
- TriParTas : $\mathcal{O}(n \log n)$

Espace : $\Theta(1)$

Arbres équilibrés

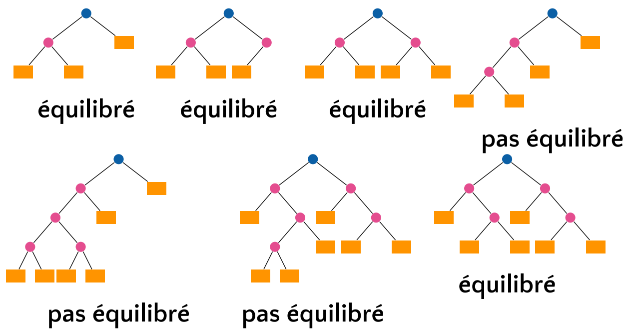
Construction d'arbres binaires équilibrés

Arbres binaires équilibrés

Un arbre binaire est dit **équilibré** soit si

- il est vide;
- les deux sous-arbres sont équilibrés et leurs hauteurs diffèrent de 1 au plus.

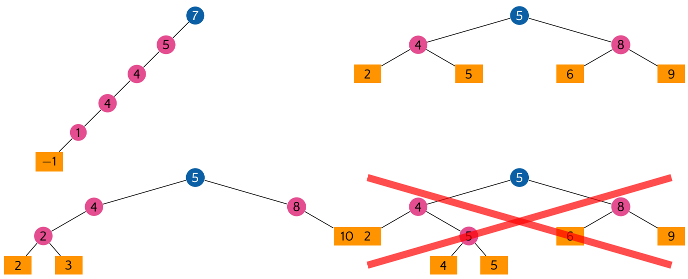
Équilibrés ou pas ?



Les ABR ne sont pas forcément équilibrés

Et c'est dommage

Exemples d'arbres binaires de recherche



Recherche dans un arbre binaire de recherche

Entrées : a: un arbre binaire de recherche
Entrées : v: une valeur à insérer
Fonction RechercheABR *(t, v)*
  si *a est vide* **alors**
    retourner Faux;
  fin
  si *a.element* = *v* **alors**
    retourner Vrai;
  fin
  si $v < a.element$ **alors**
    retourner RechercheABR (a.gauche, v);
  sinon
    retourner RechercheABR (a.droite, v);
  fin
fin

Meilleur des cas : $\mathcal{O}(1)$

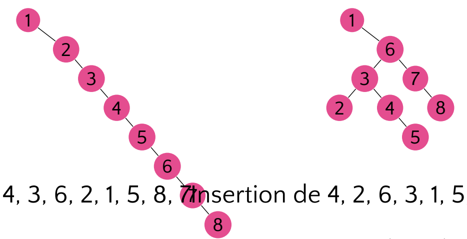
Pire des cas : $\mathcal{O}(n)$
(en tout cas $\mathcal{O}(h)$)

avec h la hauteur de l'arbre

Quelle complexité en temps ?

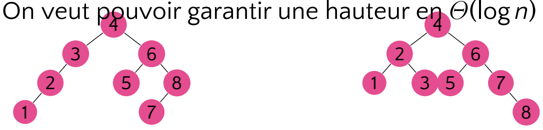
La hauteur d'un arbre binaire de recherche dépend de l'ordre d'insertion

Insertion de 1, 2, 3, 4, 5, 6, 7, 8 Insertion de 1, 6, 3, 4, 7, 5, 2, 8



Insertion de 4, 3, 6, 2, 1, 5, 8, 7 Insertion de 4, 2, 6, 3, 1, 5, 7, 8

On veut pouvoir garantir une hauteur en $\Theta(\log n)$!

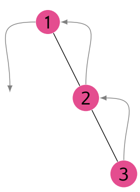


Rééquilibrer un arbre déséquilibré

Un arbre vide est équilibré

Un arbre à 1 ou 2 nœuds également

Mais pas forcément avec 3 nœuds

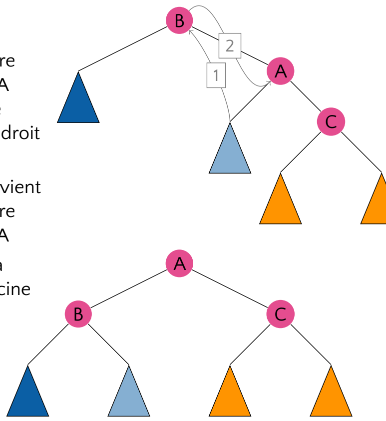


Dans ce cas, réorganisons l'arbre !

Le principe reste valable avec plus de nœuds

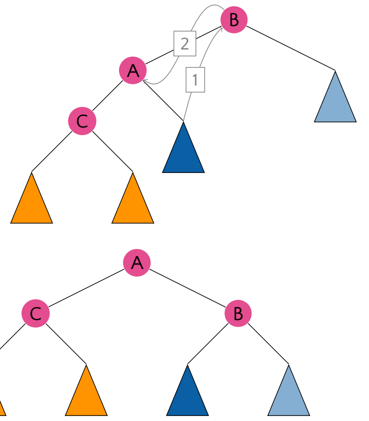
Rotation gauche dans un arbre binaire

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



Rotation droite dans un arbre binaire

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



Le parcours infixe est préservé après une rotation gauche ou droite

Les arbres AVL

Introduits par Georgy Adelson-Velsky et Evguenii Landis en 1962

An algorithm for the organization of information

Plusieurs définitions de l'équilibre

La définition donnée plus tôt dans le cours pour un arbre équilibré est celle utilisée pour les arbres AVL. On peut trouver d'autres définitions d'arbres équilibrés.

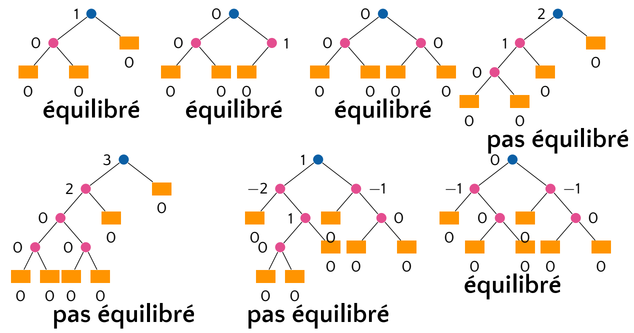
Déséquilibre d'un nœud

Le déséquilibre d'un nœud correspond à la différence de hauteur entre le sous-arbre gauche et le sous-arbre droit.

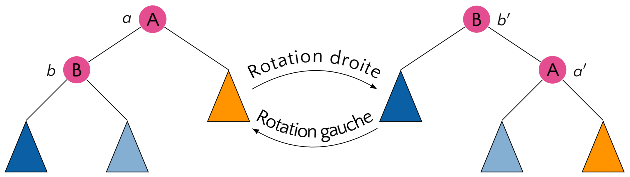
$$d(n) = hauteur(n.gauche) - hauteur(n.droite)$$

Le déséquilibre dans un arbre AVL ne peut valoir que -1, 0 ou 1

Déséquilibres dans les arbres binaires



Effet des rotations sur les déséquilibres



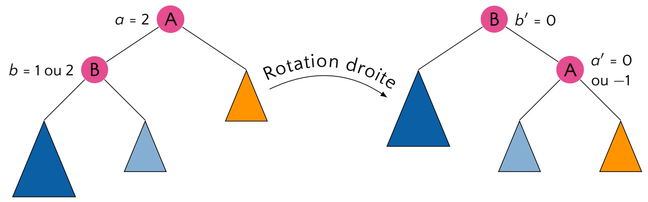
En déduire les déséquilibres après une rotation

droite: $a' = a - 1 - \max(b, 0) < a$ et $b' = b - 1 + \min(a', 0) < b$

gauche: $b = b' + 1 - \min(a', 0) > b'$ et $a = a' + 1 + \max(b, 0) > a'$

Rééquilibrer un AVL

Par définition un AVL est équilibré.
Suite à une insertion, il peut y avoir un déséquilibre (2 ou -2).
Selon le déséquilibre, il y a plusieurs cas à gérer



$a' = a - 1 - \max(b, 0) < a$ et $b' = b - 1 + \min(a', 0) < b$

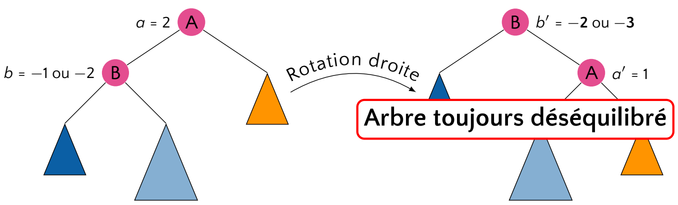
$a' = 1 - b$

Si $b = 1$: $a' = 0$ et $b' = 1 - 1 + 0 = 0$

Si $b = 2$: $a' = -1$ et $b' = 2 - 1 - 1 = 0$

Rééquilibrer un AVL

Par définition un AVL est équilibré.
Suite à une insertion, il peut y avoir un déséquilibre (2 ou -2).
Selon le déséquilibre, il y a plusieurs cas à gérer



$a' = a - 1 - \max(b, 0) < a$ et $b' = b - 1 + \min(a', 0) < b$

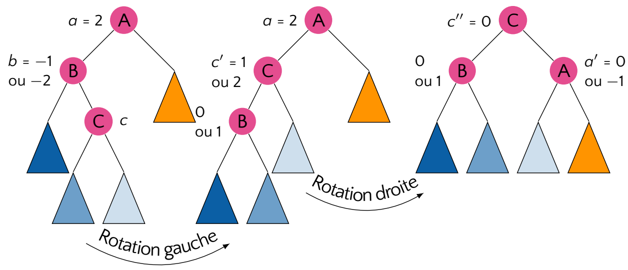
$a' = 1$

Si $b = -1$: $b' = -1 - 1 + 0 = -2$

Si $b = -2$: $b' = -2 - 1 + 0 = -3$

Rééquilibrer un AVL

Par définition un AVL est équilibré.
Suite à une insertion, il peut y avoir un déséquilibre (2 ou -2).
Selon le déséquilibre, il y a plusieurs cas à gérer



Rotation gauche-droite

Rééquilibrage après une insertion

Après une insertion

Soit aucun déséquilibre n'a été introduit

Soit l'arbre est déséquilibré à un endroit (déséquilibre de ± 2):

- Soit le déséquilibre est de 2
  - Dans le sous-arbre gauche : si le sous-sous-arbre droit est plus haut que le sous-sous-arbre gauche, **rotation gauche** du sous-arbre gauche
  - Faire une **rotation droite** du nœud déséquilibré
- Soit le déséquilibre est de -2
  - Dans le sous-arbre droit : si le sous-sous-arbre gauche est plus haut que le sous-sous-arbre droit, **rotation droite** du sous-arbre droit
  - Faire une **rotation gauche** du nœud déséquilibré

Calcul du déséquilibre

Après une insertion comment recalculer **efficacement le déséquilibre** ?

Pour calculer le déséquilibre, il faut connaître la hauteur de chaque nœud $\rightarrow$ stockons-la.

La hauteur change (éventuellement) pour les nœuds présents sur le chemin entre le nœud inséré et la racine.

Aucun changement pour tous les autres nœuds

Complexité du recalcul ? $\Theta(\log n)$

Complexité de l'insertion dans un AVL

- Trouver l'endroit où insérer : $\Theta(\log n)$
- Insérer : $\mathcal{O}(1)$
- Recalculer les hauteurs : $\Theta(\log n)$
- Rééquilibrer un nœud : $\mathcal{O}(1)$
- Nombre de rééquilibrages : $\mathcal{O}(\log n)$
- Complexité totale de l'insertion : $\Theta(\log n)$

Complexité avec les AVL

- recherche : $\Theta(\log n)$
- insertion : $\Theta(\log n)$