

IV. Arbres

→ Arbre binaire (AB)
Représentation SDD d'un AB
Algorithmes de parcours d'un AB
Arbre binaire de recherche (ABR)
Equilibrage et arbres AVL

Exemples

2

- Naturels
 - Organigramme d'une entreprise
 - Résultat d'un tournoi
 - Un chêne
- Informatiques
 - Système de fichiers
 - Langages et compilation
 - Arbre de dérivation
 - Arbre d'analyse
 - Traitements récursifs
 - ...



Utilisation

3

- Structure fondamentale de l'informatique
- Définition et traitement naturellement récurifs
- En précisant une relation d'ordre sur les éléments
 - ▣ Arbre de recherche
 - Ordre horizontal
 - ▣ Structure de tas
 - Ordre vertical
- En précisant une condition d'équilibre
 - ▣ Arbre auto-équilibrés (AVL)
- ...

Définition (théorie des graphes)

4

□ Arbre

▣ Arbre

- graphe connexe acyclique

▣ Arbre binaire

- arbre dont le degré des nœuds est au plus 3

▣ Ce sont des graphes non orientés, sans racine

□ Arbre enraciné

▣ On précise arbitrairement un nœud en tant que racine

- Parmi les nœuds de degré au plus deux

▣ L'orientation (relation parent-enfant) est alors induite

Définition (SDD)

5

- Ensemble hiérarchisé de nœuds
 - ▣ Containers d'élément
- Un nœud **racine** unique ou inexistant (**arbre vide**)
- Chaque nœud non racine possède un unique **parent**
- Chaque nœud peut posséder deux **enfants**
 - ▣ Un **fil gauche** et un **fil droit**
- **A noter**
 - ▣ On reprend la définition formelle
 - arbre binaire enraciné
 - ▣ On précise deux types de relations parent-enfant
 - Soit à droite, soit à gauche

Qualification des nœuds

6

- Par transitivité de la relation parent-enfant
 - **Ancêtre** ou **ascendant** vs. **descendant**
 - Propriété : *la racine est l'ancêtre de tous les nœuds*
- Nœuds **frères** : qui ont le même parent
- **Feuille** ou **nœud externe** : qui n'a pas d'enfant
- **Point double** ou **nœud interne** : qui a deux enfants

Branches, couches

7

- **Branche** (a, b) où b descend de a
 - ▣ Ensemble : a, b et ancêtres de b qui descendent de a
- **Branche extérieure gauche vs. droite**
 - ▣ Branche (racine, feuille la plus à gauche vs. droite)
- **Couche de profondeur n**
 - ▣ Ensemble des nœuds situés à la profondeur n

Mesures

8

- **Distance** entre deux nœuds..
 - .. qui forment une branche (a, b)
 - Nombre d'éléments de la branche moins un
- **Profondeur** d'un nœud
 - Sa distance à la racine
 - Note : la racine est donc de profondeur 0
- **Hauteur** d'un arbre
 - Profondeur de la plus profonde feuille

Sous-arbres, définition récursive

9

- Pour tout nœud
 - ▣ Le fils gauche est la racine du **sous-arbre gauche**
 - ▣ Le fils droit est la racine du **sous-arbre droit**
- Définition alternative (récursive) d'un arbre
 - ▣ Un arbre est un triplet (α, G, D) composé
 - D'un nœud racine α
 - D'un sous-arbre gauche G
 - D'un sous-arbre droit D

Arbres spéciaux

10

- **Arbre vide**

- N'a aucun nœud = est sans racine

- **Arbre entier**

- Dont tous les nœuds sont soit interne, soit feuille

- **Arbre complet**

- Arbre entier
- Pas de feuilles dont les profondeurs diffèrent > 1

- **Arbre parfait**

- Arbre entier
- Pas de feuilles dont les profondeurs diffèrent

Exo – Adressage dans un AB

11

- Soit un arbre binaire parfait de hauteur N
- Soit $n < N$:
 - ▣ Combien de nœuds pour la couche de profondeur n ?
 - ▣ Combien de nœuds pour les couches 0 à $n - 1$?
- Application
 - ▣ Indexation en largeur
 - Indexons les nœuds en partant de 1 pour la racine
 - Puis en suivant un parcours en largeur
 - ▣ Que pouvons nous dire de la relation entre index d'un parent et ceux de ses enfants ?
 - En base 10
 - En binaire
 - ▣ A quoi cela pourrait-il servir ?
- Mêmes questions avec un arbre n -aire

Corrigé – Adressage dans un AB

12

- 2^n nœuds pour la couche de profondeur n
 - ▣ Justification : le nombre double d'une couche à la suivante
- $2^n - 1$ nœuds pour l'ensemble des couches de profondeur $< n$
 - ▣ Justification : $2^0 + 2^1 + \dots + 2^{n-1} = (2^n - 1)/(2 - 1) = 2^n - 1$
- A noter
 - ▣ couche n
 - autant de nœuds que toutes les couches précédentes réunies (et même 1 de +)

Corrigé – Adressage dans un AB

13

□ Indexation

- Après représentation, on observe qu'étant donné un nœud d'index i
 - Son fils gauche a pour index $2i$
 - Son fils droit a pour index $2i + 1$
- Démonstration
 - Utiliser les dénombrements précédents
- Autre utilisation
 - Etant donné l'index i d'un nœud,
 - l'index de son parent est la partie entière de la division euclidienne de i par 2

□ Ce principe sera utilisé pour le tri par tas

- Interprétation d'un tableau en termes d'arbre binaire

Corrigé – Adressage dans un AB

14

- En binaire
 - ▣ On passe de l'écriture binaire de l'index du parent à celle des enfants en ajoutant un chiffre (digit) à droite
 - Un 0 pour le fils gauche
 - Un 1 pour le fils droit
 - ▣ Application
 - L'écriture binaire de l'index d'un nœud est une feuille de route
 - Le premier chiffre, 1, correspond à la racine
 - A chaque chiffre (de gauche à droite), correspond un pas d'itération
 - Si 1, faire un pas à gauche
 - Si 0, faire un pas à droite
- Arbre n-aire
 - ▣ Adapter les raisonnements. Conseil, travailler en base n
 - ▣ Vivement encouragé comme travail personnel
 - ▣ Réservé aux élèves les plus sérieux

IV. Arbres

Arbre binaire (AB)

→ Représentation SDD d'un AB

Algorithmes de parcours d'un AB

Arbre binaire de recherche (ABR)

Equilibrage et arbres AVL

Définition d'un nœud d'arbre

16

- Reproduction de la définition récursive
- Possibilité d'implémentation statique (cf. listes)
 - ▣ Cf. littérature pour qui souhaite approfondir
- Représentation simplement chaînée
 - ▣ La plus naturelle
 - ▣ La plus simple à mettre en œuvre

Définition d'un nœud d'arbre

17

- Structure *nœud* comportant deux champs
 - 1. un élément *e*
 - 2. l'adresse mémoire *sag* du sous-arbre **gauche**
 - 3. l'adresse mémoire *sad* du sous-arbre **droit**



Assembler un arbre

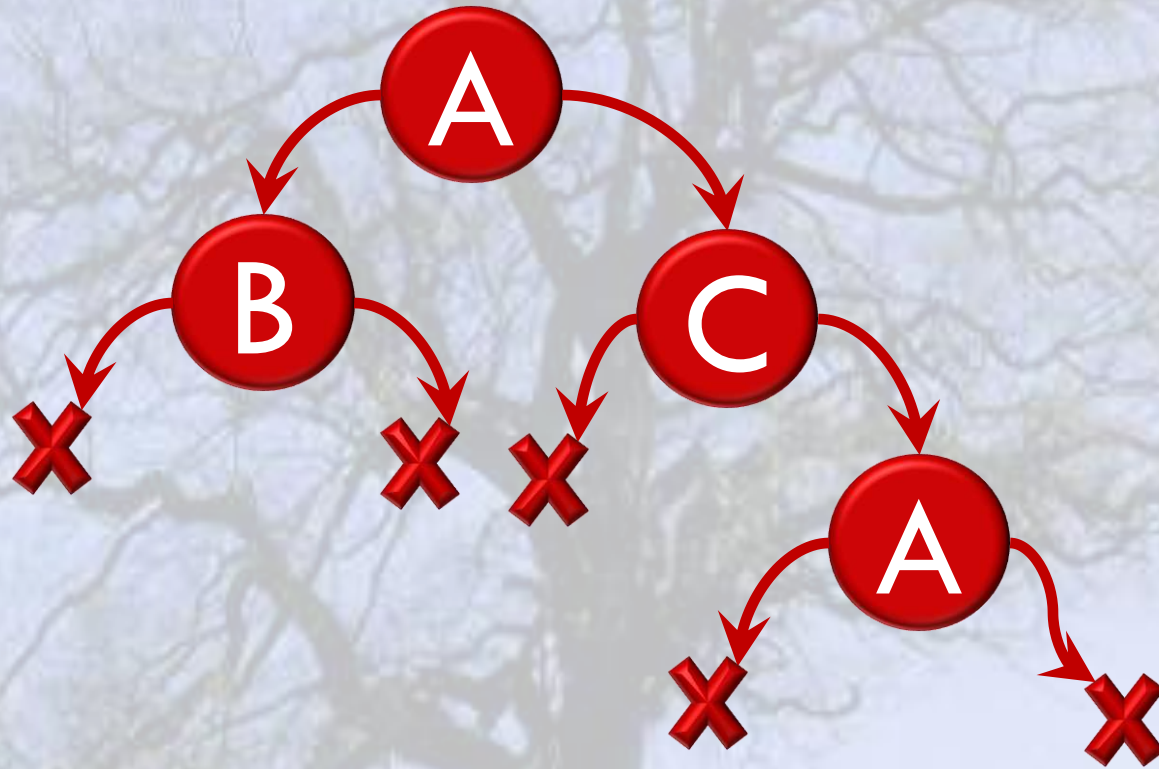
18

- Notez l'identité maillon LDC et nœud d'AB
 - ▣ Les deux pointeurs changent de nom (convention)
 - ▣ Ce qui change fondamentalement
 - La manière d'assembler les instances de ces maillons
 - Graphe acyclique → on s'interdit de former des cycles



Assembler un arbre

19



Définition d'un nœud d'AB

20

En langage algorithmique

Algorithme 50: Définition de la structure d'un *noeud* d'arbre binaire (AB)

```
type... : T
type structure noeud
  | info : T
  | sag, sad : adresse noeud
fin
type adresse noeud : arbre
```

Exemple de traduction en C

```
typedef struct noeud
{
    T info;
    struct noeud *sag, *sad;
} noeud;
typedef noeud *arbre;
```



Deux utilitaires utiles

21

Algorithme 51: *EstArbreVide(a : arbre) : booléen*

Donnée : L'arbre a à évaluer

Résultat : Un booléen

début

 si $a = \text{nul}$ alors retourner vrai
 sinon retourner faux

fin

Algorithme 52: *CréerNoeud(e : T) : arbre*

Donnée : L'élément e à stocker dans la racine de a

Variable locale : L'arbre a créé et initialisé (arbre singleton)

Résultat : a

début

$a \leftarrow \text{réserver noeud}$
 $a \rightarrow \text{info} \leftarrow e$
 $a \rightarrow \text{sag} \leftarrow \emptyset$
 $a \rightarrow \text{sad} \leftarrow \emptyset$
 retourner a

fin

Assembler un arbre

22

Concevoir un
algorithme qui permet
de vérifier qu'un AB
est bien un AB

Il pourrait bien vous avoir trompé et être une LDC, ce vilain contrefacteur !

IV. Arbres

Arbre binaire (AB)

Représentation SDD d'un AB

→ Algorithmes de parcours d'un AB

Arbre binaire de recherche (ABR)

Equilibrage et arbres AVL

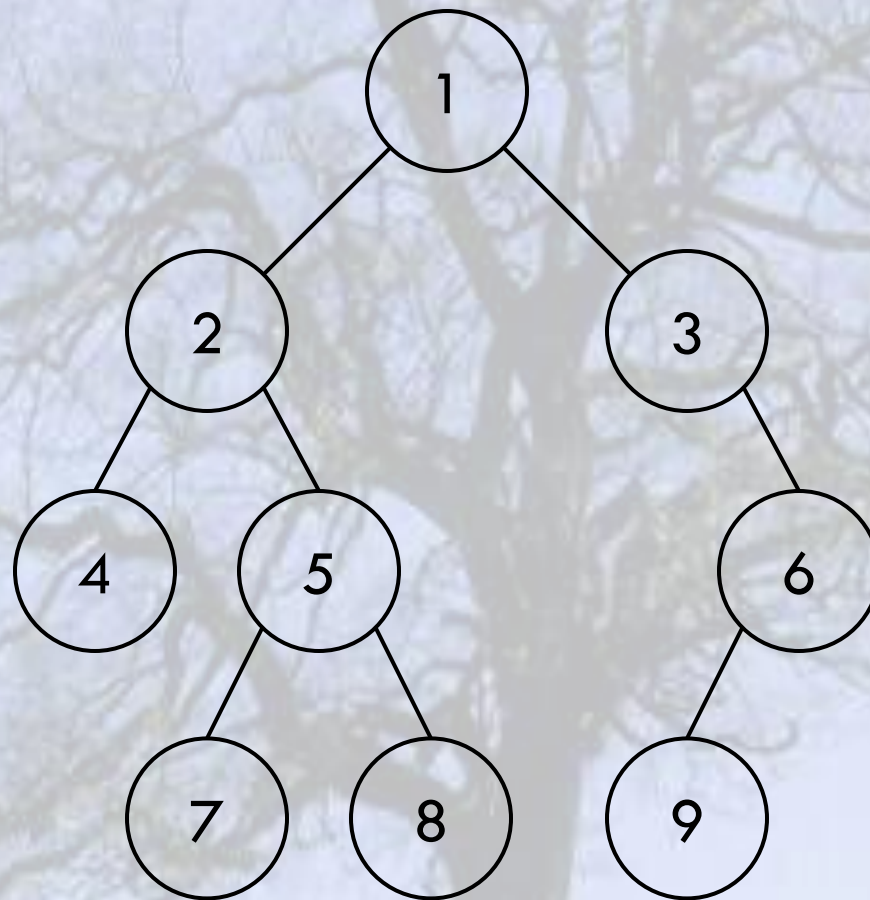
Parcourir les nœuds d'un arbre

24

- Problème
 - ▣ Comment parcourir un AB, i.e.
 - Comment visiter chaque nœud une fois et une seule?
- Deux grand types de parcours
 - ▣ En largeur
 - Niveau par niveau
 - Naturellement itératif
 - ▣ En profondeur
 - Branche par branche
 - Naturellement récursif

Parcours en largeur^(ex. wikipedia)

25



1 2 3 4 5 6 7 8 9

Arbres

AB

Parcourir

Largeur

Parcours en largeur (itératif)

26

Algorithme 53: *ParcoursLargeurGD*(a : arbre) (itératif avec file)

Donnée : L'arbre a parcouru

Variable locale : La file auxiliaire f

début

si $a = \emptyset$ **alors retourner**

InitialiserFile(f)

Enfiler(f, a)

tant que non *EstFileVide*(f) **faire**

$a \leftarrow$ *Défiler*(f)

 // c'est ici que l'on fait qqch avec le nœud courant

si non *EstArbreVide*($a \rightarrow sag$) **alors** *Enfiler*($f, a \rightarrow sag$)

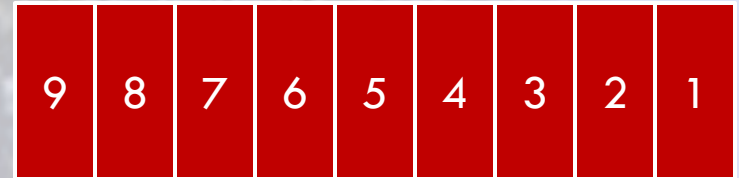
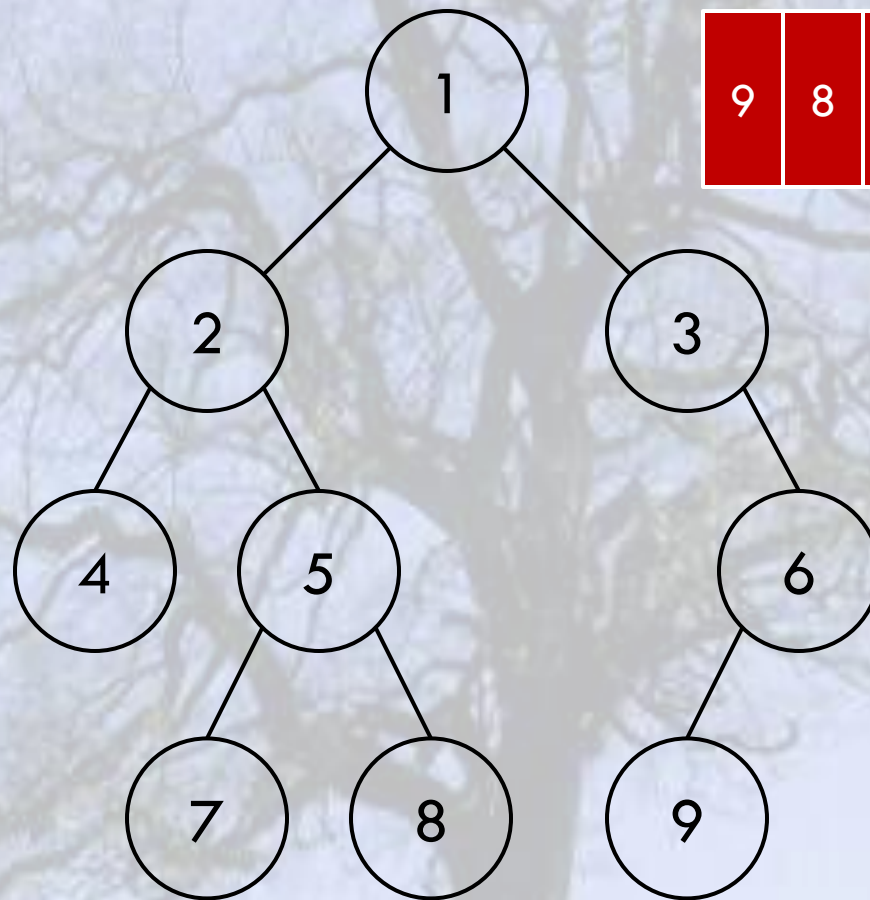
si non *EstArbreVide*($a \rightarrow sad$) **alors** *Enfiler*($f, a \rightarrow sad$)

fin

fin

Parcours en largeur

27



NDLR – Ce slide a été
particulièrement long
à concevoir

1 2 3 4 5 6 7 8 9

Arbres

AB

Parcourir

Largeur

Parcours en largeur

28

- Implémentation récursive ?
 - ▣ Sans lien direct entre frères, ça paraît difficile !
- Pour parcourir de droite à gauche ?
 - ▣ Deux instructions à permuter !
- Pour généraliser à un arbre n -aire ?
 - ▣ A chaque itération de la boucle
 - ▣ Défiler le nœud courant
 - ▣ Enfiler tous ses successeurs directs

FIN SEANCE 8

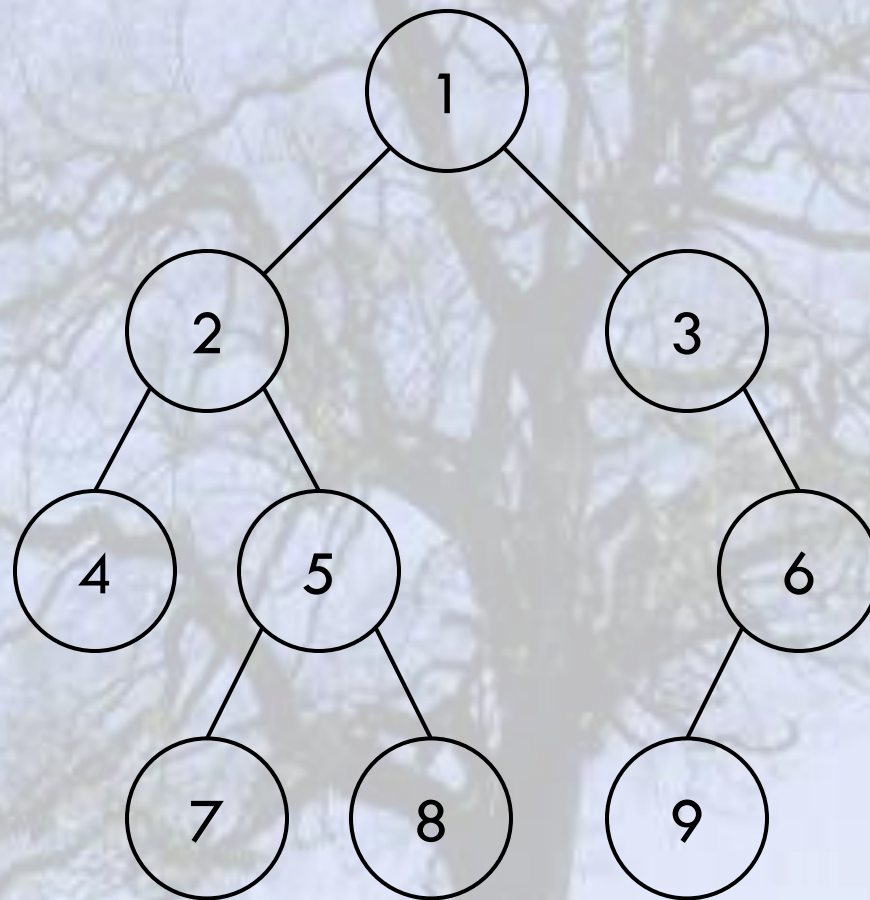
Parcours en profondeur

30

- Conception naturellement réursive
 - ▣ Série de trois instructions
 - Traitement du nœud courant (le parent)
 - Appels récursif sur ses deux enfants
- On distingue trois types de parcours
 - ▣ Selon l'ordre de ces trois instructions
 - Pré-ordre (ou préfixe)
 - parent puis enfants
 - In-ordre (ou infixé)
 - premier enfant, parent, puis second enfant
 - Post-ordre (ou postfixé)
 - enfants puis parent

Parcours en profondeur pré-ordre

31



1 2 4 5 7 8 3 6 9

Arbres

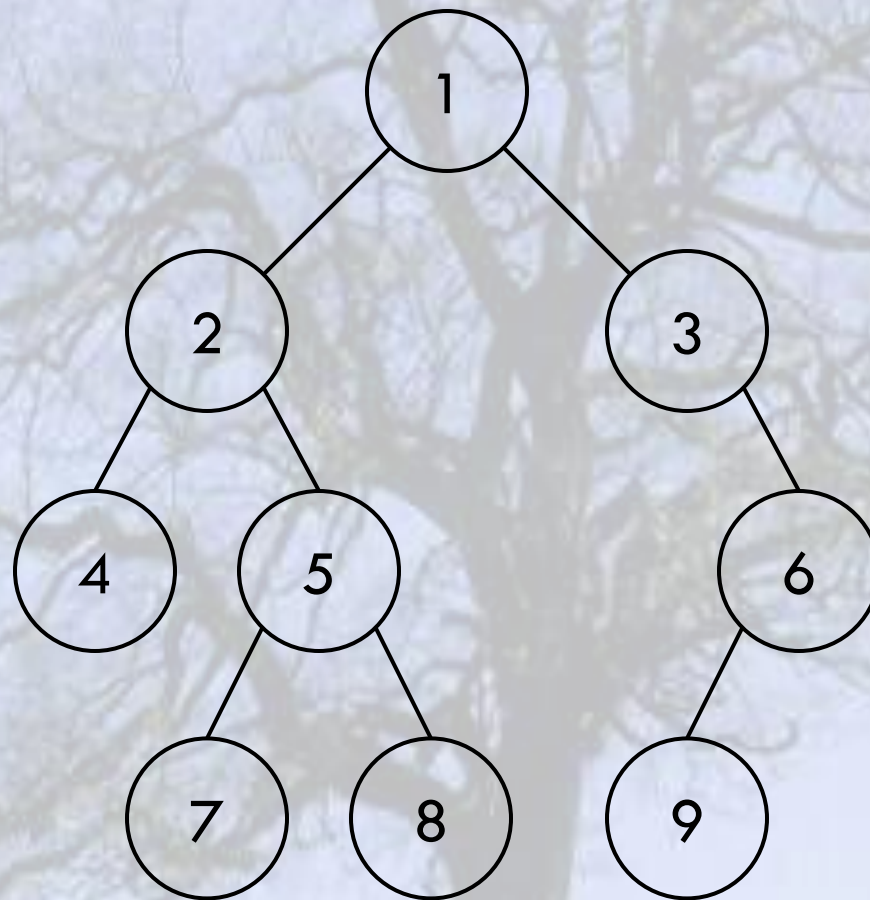
AB

Parcourir

Profondeur

Parcours en profondeur in-ordre

32



4 2 7 5 8 1 3 9 6

Arbres

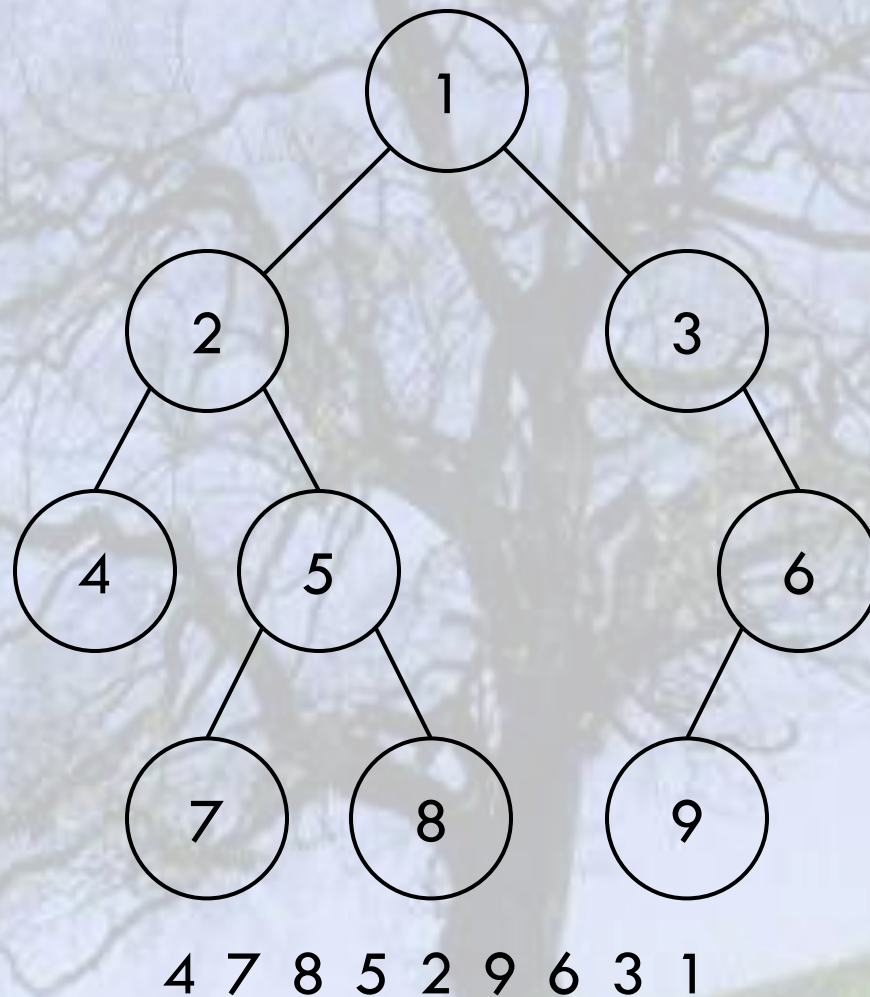
AB

Parcourir

Profondeur

Parcours en profondeur post-ordre

33



Arbres

AB

Parcourir

Profondeur

Parcours en profondeur (récursif)

34

Algorithme 55: *ParcoursPréordre*($a : \text{arbre}$) (récursif)

Donnée : L'arbre a parcouru

début

si $a = \emptyset$ **alors retourner**

 // c'est ici que l'on fait qqch avec le nœud courant a

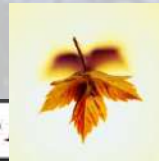
ParcoursPréordre($a \rightarrow \text{sag}$)

ParcoursPréordre($a \rightarrow \text{sad}$)

fin

Parcours en profondeur (itératif)

35



Algorithme 54: *ParcoursProfondeur.*

(*a* : arbre) (itératif avec pile)

Donnée : L'arbre *a* parcouru

Variable locale : La pile auxiliaire *p*

début

si *EstArbreVide(a)* **alors retourner**

InitialiserPile(p)

Empiler(p, a)

tant que non *EstPileVide(p)* **faire**

$a \leftarrow \text{Dépiler}(p)$

 // c'est ici que l'on fait qqch avec le nœud courant

si non *EstArbreVide(a → sad)* **alors** *Empiler(p, a → sad)*

si non *EstArbreVide(a → sag)* **alors** *Empiler(p, a → sag)*

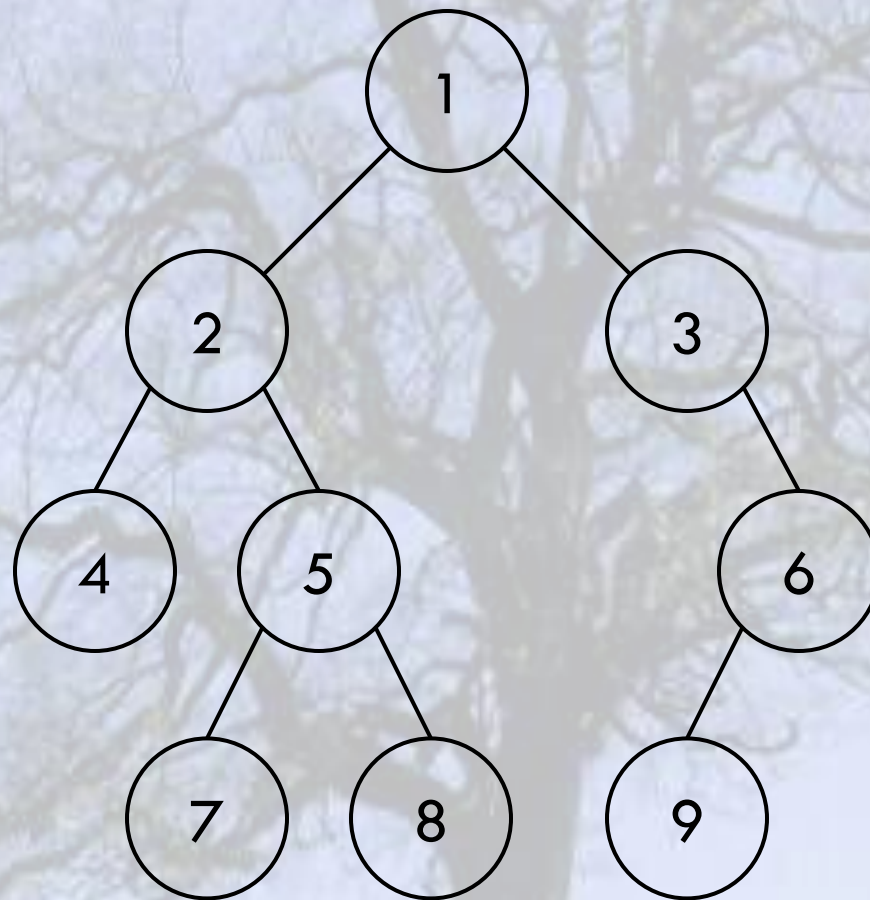
 // Pour faire du post-ordre : permuter les deux dernières instructions

fin

fin

Parcours en profondeur

36



7
8
9

1 2 4 5 7 8 3 6 9

Arbres

AB

Parcourir

Largeur

Exos de réflexion

37

- Exercices de réflexion sur les applications
 - ▣ Donner un exemple de problème qui ne peut se traiter qu'en parcours post-ordre. Justifier.
 - ▣ Quel parcours est réalisé par l'algorithme itératif? Comment l'adapter pour obtenir les deux autres?
 - ▣ Après avoir étudié la section suivante, sur l'ABR : quel parcours restitue exactement les valeurs d'un ABR en ordre croissant?

Corrigé

38

- Libérer la mémoire occupée par un arbre
 - ▣ Il faut libérer les parents après les enfants
 - ▣ Sinon, on perd l'adresse des enfants à supprimer
 - ▣ Par conséquent : traitement post-ordre indispensable
- Le parcours itératif
 - ▣ Est un traitement pré-ordre
 - On traite toujours le parent avant d'empiler les enfants
 - Qui seront donc dépilés et traités ultérieurement
 - ▣ Post et in-ordre en itératif : à vous de jouer !
- Le parcours en ordre infixe restitue exactement les valeurs d'un ABR en ordre croissant

Quelques exemples d'algorithmes

39

- Compter le nombre d'éléments d'un AB
- Libérer la mémoire occupée par un arbre
- Mesurer la hauteur d'un AB
- Soient deux adresses de nœuds
 - ▣ Vérifier qu'ils forment une branche
 - i.e. que le premier est l'ancêtre du second
 - ▣ Retourner cette branche sous forme de LSC
- Créer un arbre parfait de hauteur n dont les nœuds contiennent respectivement $1, 2, \dots, 2^{n+1} - 1$, suivant un parcours en largeur

Compter le nombre d'éléments

40

Algorithme 45: *NombreElements*(a : arbre) : entier (récursif)

Donnée : L'arbre a dont on compte les éléments

Résultat : le nombre d'éléments de a

début

si *EstArbreVide*(a) **alors retourner** 0

sinon retourner $1 + \text{NombreElements}(a \rightarrow \text{sag}) + \text{NombreElements}(a \rightarrow \text{sad})$

fin

Libérer la mémoire

41

Algorithme 56: *Libérer*(a : arbre)

Donnée modifiée : L'arbre a que l'on libère

début

 si $a \neq \emptyset$ alors

Libérer($a \rightarrow sag$)

Libérer($a \rightarrow sad$)

 libérer a

 fin

fin

Mesurer la hauteur

42

Algorithme 57: *Hauteur*(a : arbre)

Donnée : L'arbre a que l'on mesure

Variable locale : h_g et h_d , les hauteurs respectives des fils gauche et droit

Résultat : La hauteur de a

début

 si $a = \emptyset$ alors retourner 0

 sinon

$h_g \leftarrow \text{Hauteur}(a \rightarrow \text{sag})$

$h_d \leftarrow \text{Hauteur}(a \rightarrow \text{sad})$

 si $h_g > h_d$ alors retourner $1 + h_g$

 sinon retourner $1 + h_d$

 fin

fin

Vérifier la branche

43

- Le problème peut être reformulé
 - ▣ Vérifier que le second nœud représente un sous-arbre de l'arbre représenté par le premier

Algorithme 58: *EstSousArbreDe*($a, b : \text{arbre}$) : *booléen*

Donnée : Deux arbres a et b dont on vérifie que le 2^d est sous-arbre du 1^{er}

Résultat : Un booléen vrai ssi b est sous-arbre de a

début

 si $a = \emptyset$ ou $b = \emptyset$ alors retourner *faux*

 si $a = b$ alors retourner *vrai*

 retourner *EstSousArbreDe*($a \rightarrow \text{sag}, b$) ou *EstSousArbreDe*($a \rightarrow \text{sad}, b$)

fin

Construire la branche

44

□ L'idée

- 1. Descendre récursivement dans toutes les branches
 - Le nœud cible, et seulement lui, initie une liste (la queue)
- 2. En remontant, compléter par ajout préfixe (tête)
 - Ssi l'un des deux fils n'a pas retourné une liste nulle

Construire la branche

45

Algorithme 1: *RetournerBranche*($a, b : \text{arbre}$) : *liste*

```
Donnée      : Un arbre  $a$  et  $b$  l'un de ses sous-arbres
Variable locale : La liste  $l$  retournée par l'un des deux appels
Variable locale : La liste  $m$  qui représente le maillon construit à l'échelon local
Résultat    : La branche  $(a, b)$  sous forme de LSC, i.e.  $m$ 
début
  // Cas d'initialisation de la branche (sa queue)
  si  $a = b$  alors
     $m \leftarrow \text{réserver maillon}$ 
     $m \rightarrow \text{info} \leftarrow a \rightarrow \text{info}$ 
     $m \rightarrow \text{succ} \leftarrow \emptyset$ 
    retourner  $m$ 
  fin
  // On appelle, sinon, récursivement sur le fils gauche, puis le fils droit
   $l \leftarrow \emptyset$ 
  si non EstArbreVide( $a \rightarrow \text{sag}$ ) alors  $l \leftarrow \text{RetournerBranche}(a \rightarrow \text{sag}, b)$ 
  si  $l = \emptyset$  alors
    si non EstArbreVide( $a \rightarrow \text{sag}$ ) alors  $l \leftarrow \text{RetournerBranche}(a \rightarrow \text{sad}, b)$ 
  fin
  // pour, seulement si l'un d'entre eux possède bien  $b$  comme sous-arbre :
  si  $l = \emptyset$  alors retourner  $\emptyset$ 
  // compléter le travail :
   $m \leftarrow \text{réserver maillon}$ 
   $m \rightarrow \text{info} \leftarrow a \rightarrow \text{info}$ 
   $m \rightarrow \text{succ} \leftarrow l$ 
  retourner  $m$ 
fin
```

Créer un arbre parfait

46

Algorithme 60: *NouvelArbreParfait*(n : entier naturel, e : entier positif)

Variable locale : L'arbre parfait a à créer
Donnée : La hauteur n de a
Donnée : L'élément e à stocker dans le nœud racine de a
Résultat : a
début
 // Note : appel initial *NouvelArbreParfait*($n, 1$)
 $a \leftarrow \text{CréerNoeud}(e)$
 si $n > 0$ **alors**
 $a \rightarrow \text{sag} \leftarrow \text{NouvelArbreParfait}(n - 1, 2 \times e)$
 $a \rightarrow \text{sad} \leftarrow \text{NouvelArbreParfait}(n - 1, 2 \times e + 1)$
 fin
 retourner a
fin

FIN SEANCE 9

IV. Arbres

Arbre binaire (AB)

Représentation SDD d'un AB

Algorithmes de parcours d'un AB

→ Arbre binaire de recherche (ABR)

Equilibrage et arbres AVL

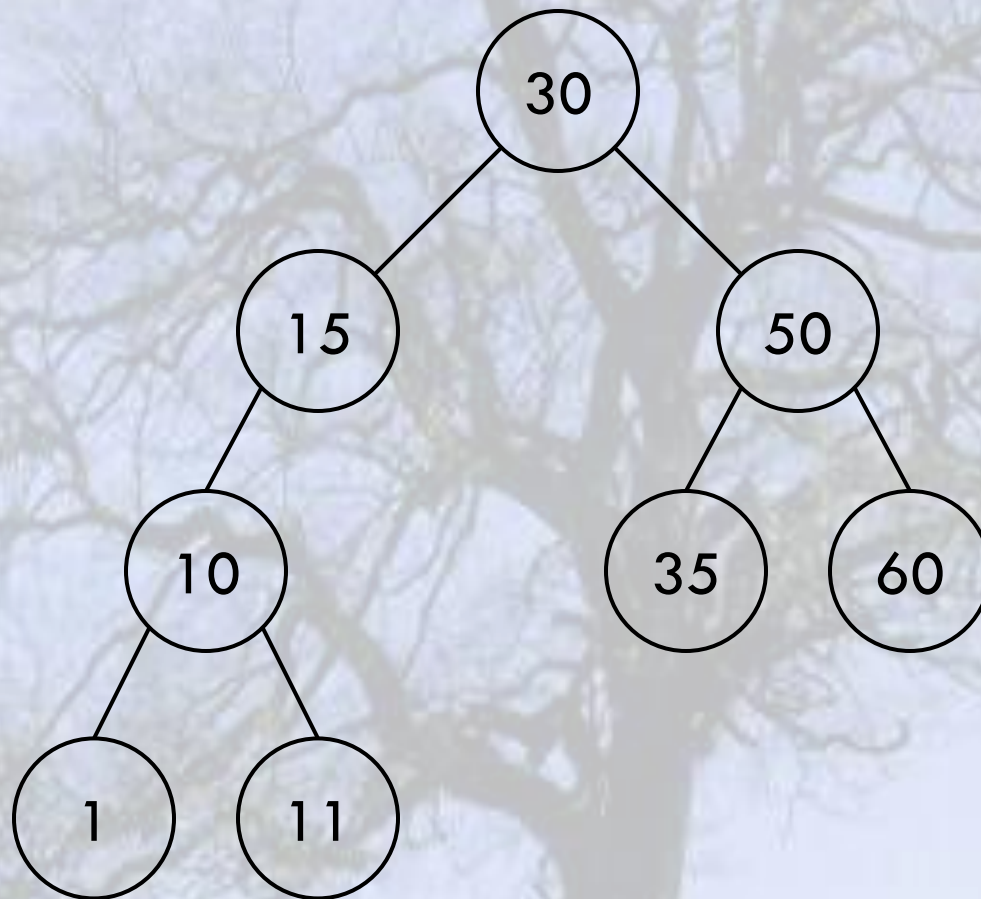
Arbre Binaire de Recherche (ABR)

49

- C'est un AB tel pour tout nœud a
 - ▣ $\max(\text{sag}(a)) \leq a \leq \min(\text{sad}(a))$ (ordre croissant)
 - ▣ $\max(\text{sad}(a)) \leq a \leq \min(\text{sag}(a))$ (ordre décroissant)
- La relation porte (est-ce utile de le préciser ?)
 - ▣ Sur les éléments des nœuds et non pas sur les adresses
- Il s'agit d'une relation d'ordre (bon ordre)
 - ▣ Elle en induit une par niveau
 - Les frères sont bien ordonnés
 - ▣ Si on impose $<$, relation d'ordre totale
- Pour consultation de la littérature
 - ▣ EN : Binary Search Tree (BST)

ABR

50



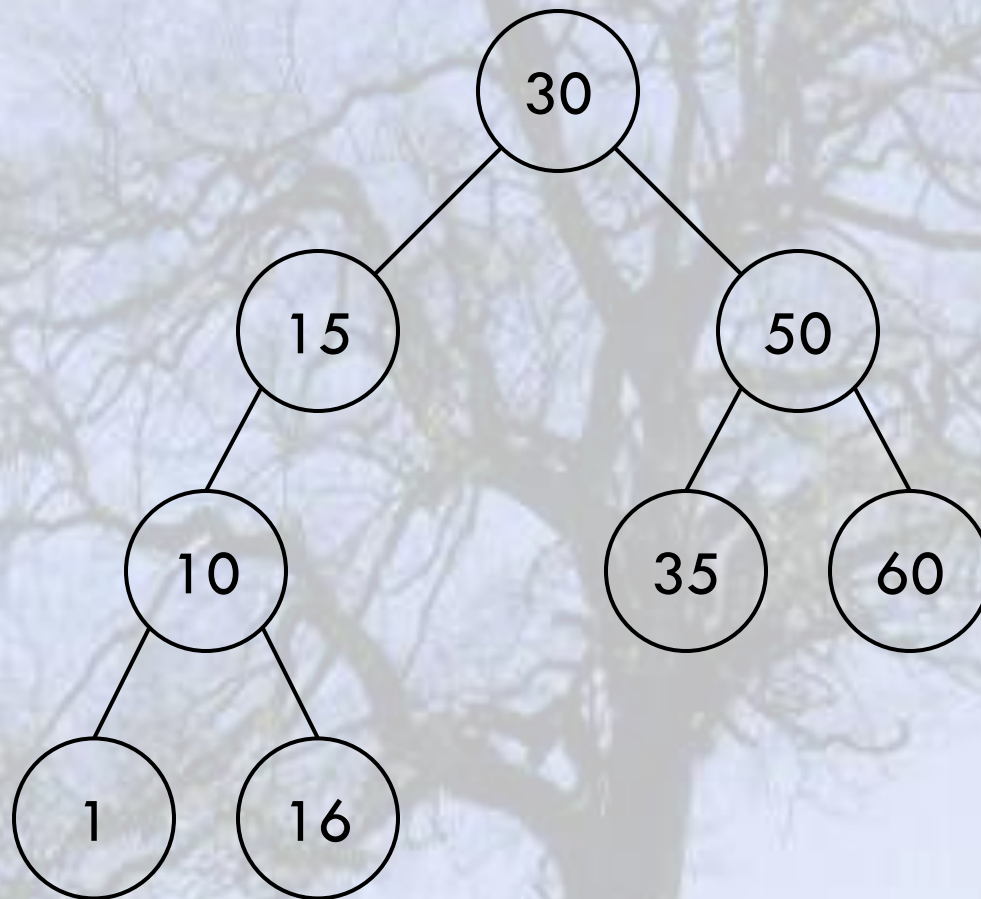
Arbres

ABR

Exemple

Non ABR

51



Arbres

ABR

Contrex

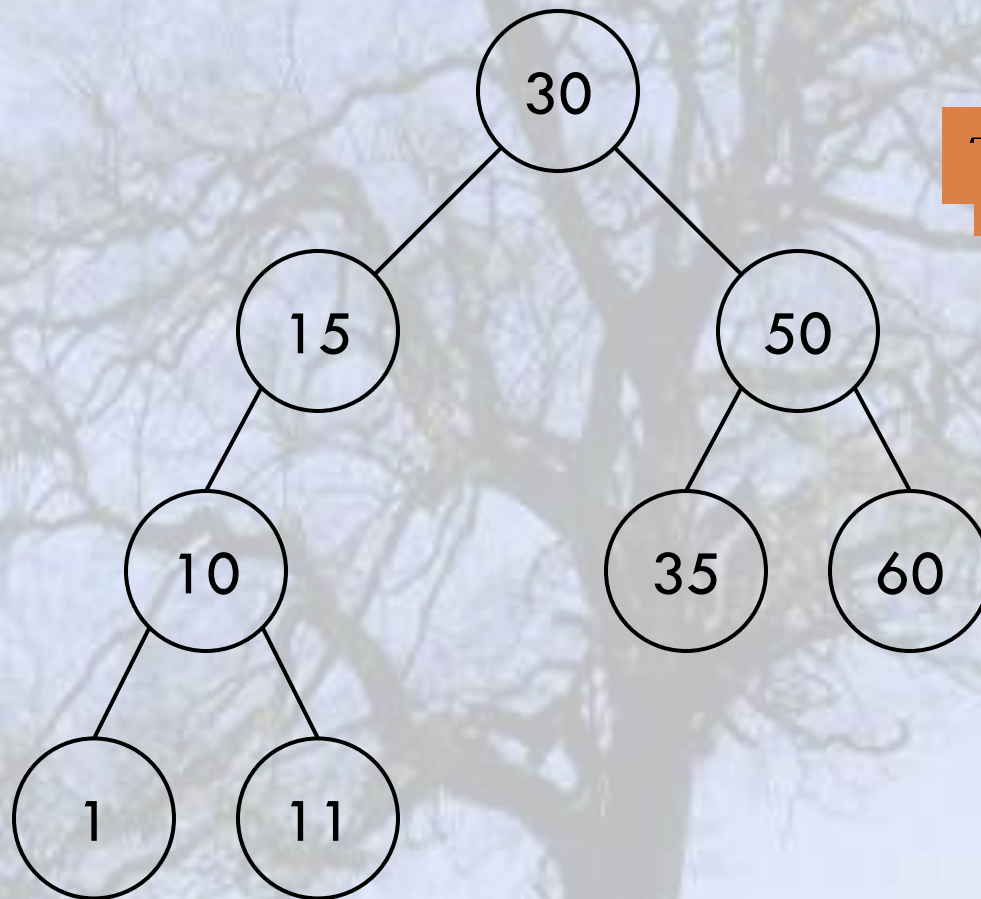
Rechercher un élément

52

- Simple dichotomie
 - ▣ Plus besoin d'algorithme récursif et coûteux
 - ▣ Une boucle d'itération suffit
 - ▣ Gain : $O(n) \rightarrow O(\ln n)$ en moyenne
 - ▣ Mais on retombe en $O(n)$
 - si l'arbre est totalement déséquilibré ($\sim$ parcours de liste)
- Note sur l'exercice « adressage binaire »
 - ▣ Même type de parcours avec un AB quelconque
 - Si l'on dispose de l'adresse (positionnelle) de la destination

Illustration : on recherche 11

53



11 < 10 ? F → D

L'algorithme (version récursive)

54

- Si l'on souhaite compter le nombre d'occurrences
 - ▣ Ajouter un compteur
 - ▣ Si ordre strict, c'est inutile

Algorithme 61: *RechercheABR*($a : ABR, e : T$)

Donnée : L'ABR a dans lequel on effectue la recherche

Donnée : L'élément e recherché

Résultat : un booléen vrai ssi e est un élément de a

début

 si $a = \emptyset$ alors retourner *faux*

 si $a \rightarrow info = e$ alors retourner *vrai*

 si $e < a \rightarrow info$ alors *RechercheABR*($a \rightarrow sag, e$)

 sinon *RechercheABR*($a \rightarrow sad, e$)

fin

L'algorithme (version itérative)

55

Algorithme 62: *RechercheABR*($a : ABR, e : T$) (itératif)

Donnée : L'ABR a dans lequel on effectue la recherche

Donnée : L'élément e recherché

Résultat : un booléen vrai ssi e est un élément de a

début

tant que $a \neq \emptyset$ **faire**

si $a \rightarrow info = e$ **alors retourner** *vrai*

si $e < a \rightarrow info$ **alors** $a \leftarrow a \rightarrow sag$

sinon $a \leftarrow a \rightarrow sad$

fin

retourner *faux*

fin

Ajouter un élément

56

- Ajout au niveau des feuilles
 - ▣ 1. Si l'arbre est vide, on crée une racine
 - ▣ 2. Sinon on itère jusqu'au point d'insertion
 - Place vide sag ou sad dont on mémorise le parent
 - On crée le nœud et on l'insère
- C'est donc une adaptation de la recherche

Ajouter un élément

57

Algorithme 63: *AjoutABR*($a : ABR, e : T$)

Donnée modifiée : L'ABR a dans lequel on effectue l'ajout

Variable locale : Une variable b pour itérer a (en donnée modifiée)

Variable locale : L'adresse p du parent du point d'insertion

Donnée : L'élément e à ajouter

début

 si *EstArbreVide*(a) alors $a \leftarrow \text{NouveauNoeud}(e)$

 sinon

$b \leftarrow a$

 tant que non *EstArbreVide*(b) faire

$p \leftarrow b$

 si $e < b \rightarrow \text{info}$ alors $b \leftarrow b \rightarrow \text{sag}$

 sinon $b \leftarrow b \rightarrow \text{sad}$

 fin

$b \leftarrow \text{NouveauNoeud}(e)$

 si $e < p \rightarrow \text{info}$ alors $p \rightarrow \text{sag} \leftarrow b$

 sinon $p \rightarrow \text{sad} \leftarrow b$

 fin

fin

Supprimer un élément

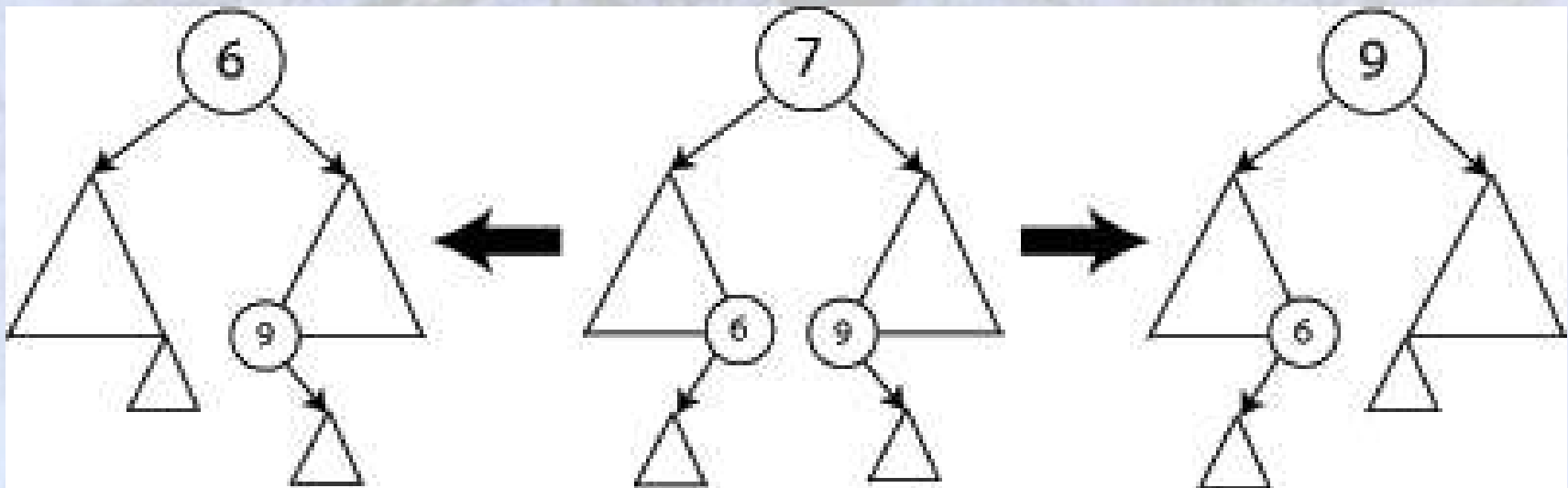
58

- A) Pour supprimer une feuille, RAS
- B) Pour supprimer un nœud à un seul enfant
 - ▣ Raccorder l'enfant en lieu et place du parent supprimé
- C) Sinon, c'est un peu plus compliqué, mais à peine
 - ▣ Soit a le nœud à supprimer et :
 - $m = \max(a \rightarrow sag)$, $M = \min(a \rightarrow sad)$
 - Observons que m et M ne peuvent avoir deux enfants !!!
 - 1. Permuter la valeur du nœud à supprimer avec m ou avec M
 - La structure d'ABR est préservée !!!
 - 2. Supprimer celui de m ou M qu'on a permuté avec a
 - On se ramène à l'un des deux cas A) ou B) !!!

Supprimer un élément : illustration

59

- Le cas C) (source : wikipedia – copyleft)
- Réalisation : en TD



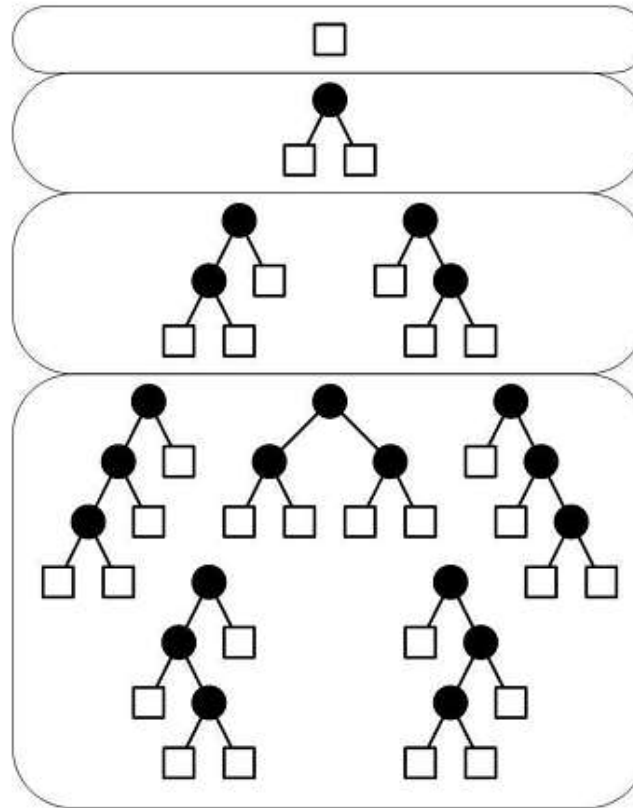
Problème

60

- Soient n éléments strictement ordonnés
- Combien existe-t-il d'ABR 2 à 2 distincts qui contiennent exactement ces n éléments
- Coup de pouce :
 - ▣ 1. Construire les premiers cas et les compter b_n
 - ▣ 2. Formuler une relation de récurrence
 - ▣ 3. La démontrer

61

En étudiant les premiers cas, on peut facilement se convaincre que $b_0 = b_1 = 1$, $b_2 = 2$, $b_3 = 5$:



Légende : ● représente un nœud, et □ représente un sous-arbre vide.

Corrigé 2/3

62

De l'établissement de ces premiers cas se dégage un procédé de construction d'un ensemble de possibilités de rang n à partir des ensembles de rangs inférieurs, et une relation de récurrence entre b_{n+1} et $b_0, \dots, b_n$.

Pour construire l'ensemble des possibilités de rang $n+1$, on fixe le sommet, puis on considère toutes les façons de répartir les n autres éléments en deux sous-ensembles (les éléments du sous-arbre gauche et ceux du sous-arbre droit), puis, pour chacune de ces partitions $n = p + q, (p, q) \in \llbracket 0, n \rrbracket^2$, on considère respectivement les b_p et b_q façons de construire respectivement le sous-arbre gauche et le sous-arbre droit.

Cela nous conduit à énoncer la relation de récurrence suivante :

$$b_{n+1} = \sum_{k=0}^n b_k b_{n-k}$$

Corrigé 3/3

63

Etant donnée la série $b(z) = \sum_{n=0}^{+\infty} b_n z^n$, observons que :

$$b^2(z) = \sum_{n=0}^{+\infty} \left(\sum_{k=0}^n b_k b_{n-k} \right) z^n = \sum_{n=0}^{+\infty} b_{n+1} z^n = \sum_{n=1}^{+\infty} b_n z^{n-1}$$

$$\Leftrightarrow z b^2(z) = \sum_{n=1}^{+\infty} b_n z^n = b(z) - b_0$$

$$\Leftrightarrow z b^2(z) - b(z) + 1 = 0$$

A vous de terminer la preuve qui établit que :

$$\forall n \in \mathbb{N}, b_n = \frac{(2n)!}{n!(n+1)!} = \frac{1}{2n+1} C_{2n+1}^n$$

FIN SEANCE 10

IV. Arbres

Arbre binaire (AB)

Représentation SDD d'un AB

Algorithmes de parcours d'un AB

Arbre binaire de recherche (ABR)

→ Equilibrage et arbres AVL

Problématique

66

- Opérations sur un ABR de profondeur p
 - ▣ Basées sur dichotomie
 - Au pire, traversée de p nœuds
 - Si l'ABR est parfait
 - $n = 2^{p+1} - 1$ nœuds en tout
 - Donc complexité $\sim O(\ln n)$
 - Si l'ABR est une simple branche
 - $n = p$
 - Donc complexité $\sim O(n)$
- Déséquilibre → dégradation des performances

Problématique

67

- Cas : ajouts dans un ABR
 - ▣ Rappel : nouvelles feuilles
 - ▣ Ex. Ajout de N éléments supérieurs à la racine
 - Tous ces éléments sont ajoutés dans le SAD
 - L'arbre se déséquilibre fortement à droite
- ➔ Besoin d'une technique dynamique d'équilibrage
 - ▣ But
 - Maintenir l'équilibre au fur et à mesure des ajouts
 - ▣ Moyen
 - Restructurer (si nécessaire) l'ABR après chaque ajout

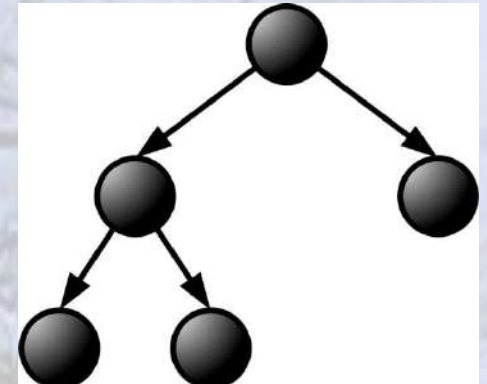
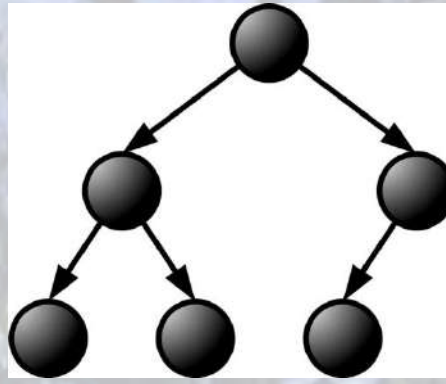
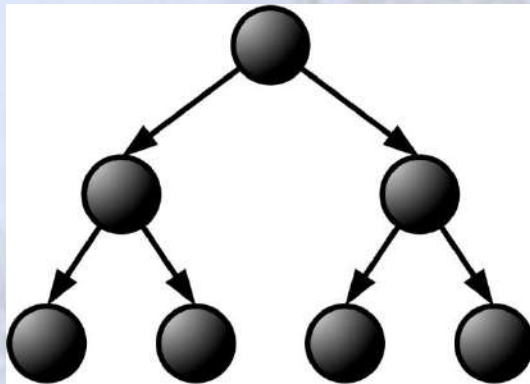
Arbre équilibré

68

- Equilibre parfait (resp. partiel)
 - *Un AB est parfaitement (resp. Partiellement) équilibré ssi, pour chacun de ses sous-arbres, la différence entre le nombre de nœuds (resp. la hauteur) du SAG et du SAD est au plus 1.*
- Formellement
 - Pour tout nœud α de l'arbre
 - $|n_{sad}(\alpha) - n_{sag}(\alpha)|$ (resp. $|h_{sad}(\alpha) - h_{sag}(\alpha)|$) ≤ 1
- Attention
 - Comme pour le caractère d'ABR d'un AB
 - Vérification récursive de la propriété indispensable (err. Fréquente)
- Notes
 - Propriété purement structurale
 - Elle n'est pas liée au caractère d'ABR de l'AB
 - Equilibre parfait → Equilibre partiel
 - Réciproque fausse

Arbre équilibré

69



- Lesquels sont partiellement (resp. parfaitement) équilibrés ?

Algorithme de vérification

70

- Algorithmes *NombreElements* et *Hauteur* déjà vus
- Algorithme (un plus performant en TD) :

Algorithme 64: *EstEquilibré*($a : AB$)

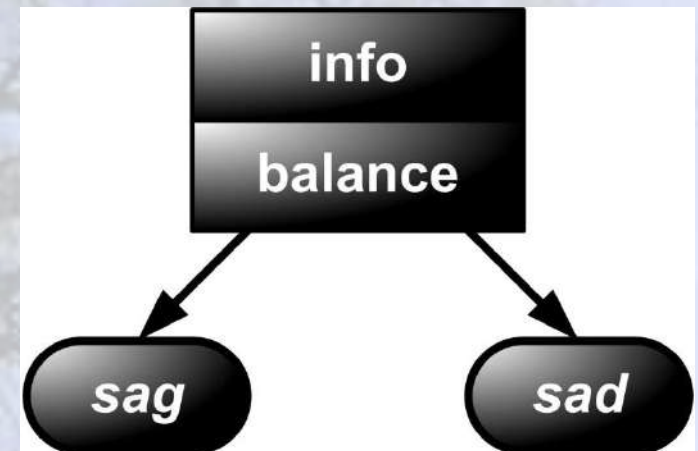
Donnée : L'ABR a dont on vérifie l'équilibre parfait
Résultat : Un booléen qui indique si a est équilibré ou non
début
 si *EstArbreVide*(a) **alors retourner** *vrai*
 si $|\text{NombreEléments}(a \rightarrow sad) - \text{NombreEléments}(a \rightarrow sag)| > 1$ **alors**
 retourner *faux*
 fin
 si non *EstEquilibré*($a \rightarrow sag$) **alors retourner** *faux*
 retourner *EstEquilibré*($a \rightarrow sad$)
fin

- Equilibre partiel : remplacer NE par H !

Modèle AVL

71

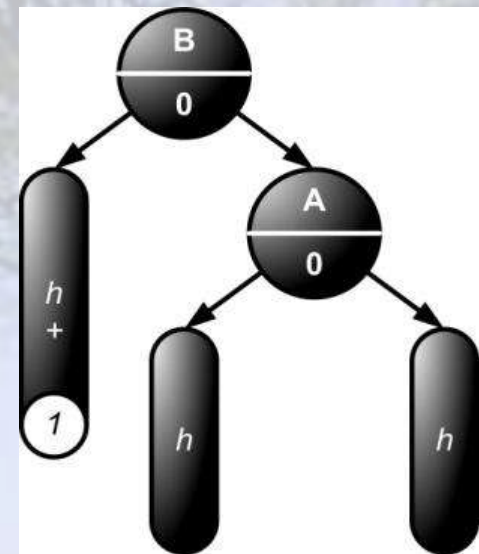
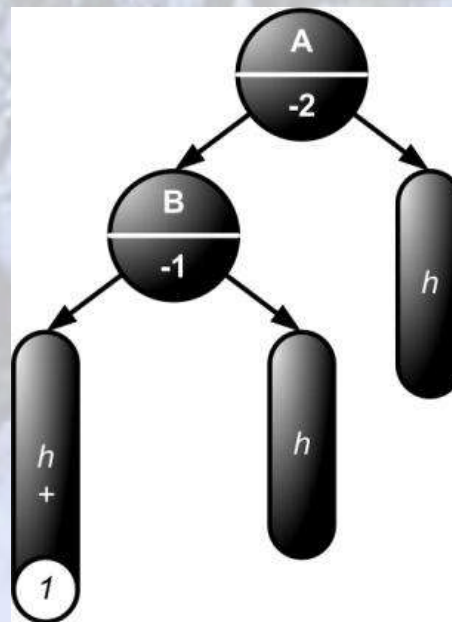
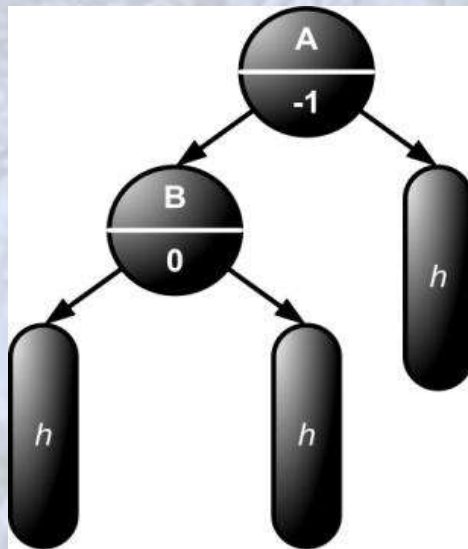
- Principe
 - ▣ Arbre capable de maintenir un équilibre partiel
 - ▣ **Andelson-Velskii** et **Landis** (auteurs, ...)
- Spécification
 - ▣ Structure de maillon d'AB
 - Ajout d'un champ pour mémoriser
 - Le *facteur d'équilibrage*
 - = état d'équilibre
 - i.e. valeur locale de $h_{sad}(a) - h_{sag}(a)$
 - ▣ Algorithmes d'ajout / suppression
 - Objectif : maintenir balance, en tout nœud
 - dans le domaine $\{-1, 0, 1\}$



Rotation Gauche-Gauche (GG)

72

- Arbre plus haut par la gauche (facteur d'éq. = -1)
 - ▣ Ajout d'une feuille dans le SAG du SAG
 - ▣ ➔ Rotation GG
 - A devient fils droit de B
 - Le SAD de B devient le SAG de A



Arbres

AVL

Equilibrage

Application à l'AVL

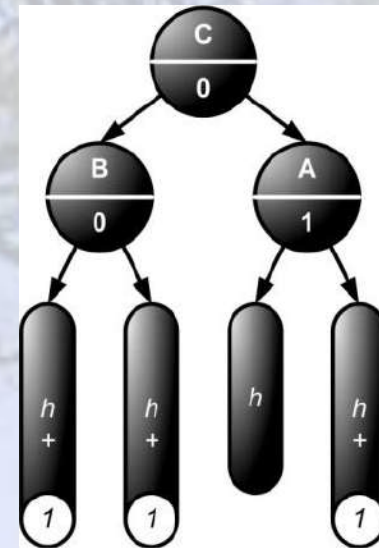
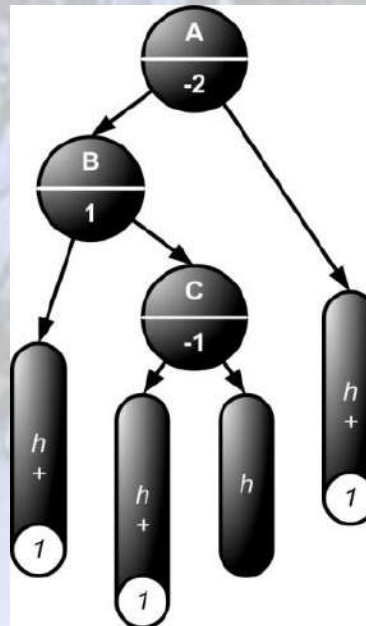
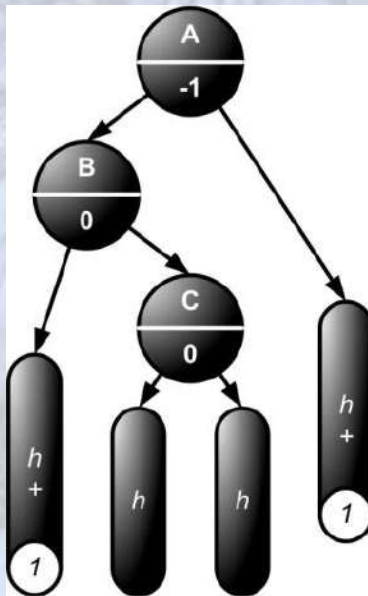
73

- Après ajout, soit l'ascendant du nœud ajouté
 - ▣ Le plus proche du nœud ajouté
 - ▣ Dont le facteur d'équilibrage devient ± 2
- 4 cas à distinguer
 - ▣ Selon que la nouvelle feuille est ajoutée
 - Respectivement dans l'un des 4 sous-arbres
 - $SAG(SAG(\alpha))$, $SAD(SAG(\alpha))$, $SAD(SAD(\alpha))$, $SAG(SAD(\alpha))$
 - ▣ En fait deux, à la symétrie verticale près
 - GG Gauche-Gauche
 - GD Gauche-Droite
 - DD Droite-Droite
 - DG Droite-Gauche

Rotation Gauche-Droite (GD)

74

- Arbre plus haut par la gauche (facteur d'éq. = -1)
 - ▣ Ajout d'une feuille dans le SAD du SAG (cas 1)
 - ▣ ➔ Rotation GD
 - A et B deviennent respectivement fils droit et gauche de C
 - La SAG et le SAD de C deviennent respectivement les SAD de B et SAG de A



Arbres

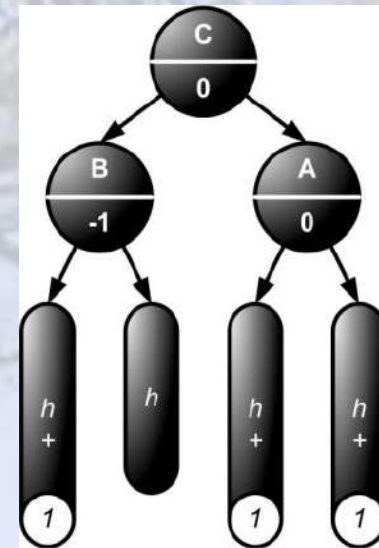
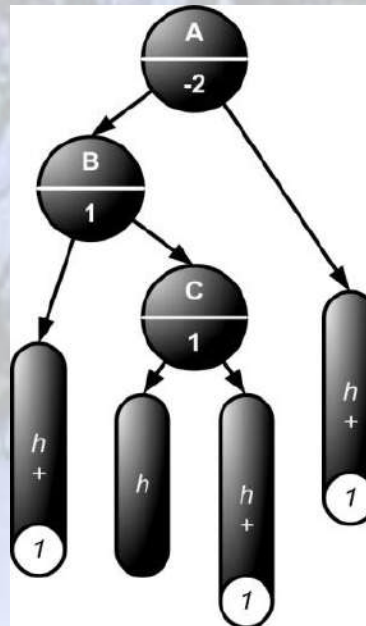
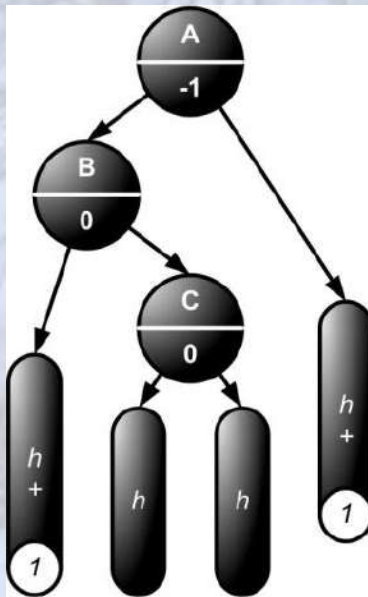
AVL

Equilibrage

Rotation Gauche-Droite (GD)

75

- Arbre plus haut par la gauche (facteur d'éq. = -1)
 - ▣ Ajout d'une feuille dans le SAD du SAG (cas 2)
 - ▣ ➔ Rotation GD
 - A et B deviennent respectivement fils droit et gauche de C
 - La SAG et le SAD de C deviennent respectivement les SAD de B et SAG de A



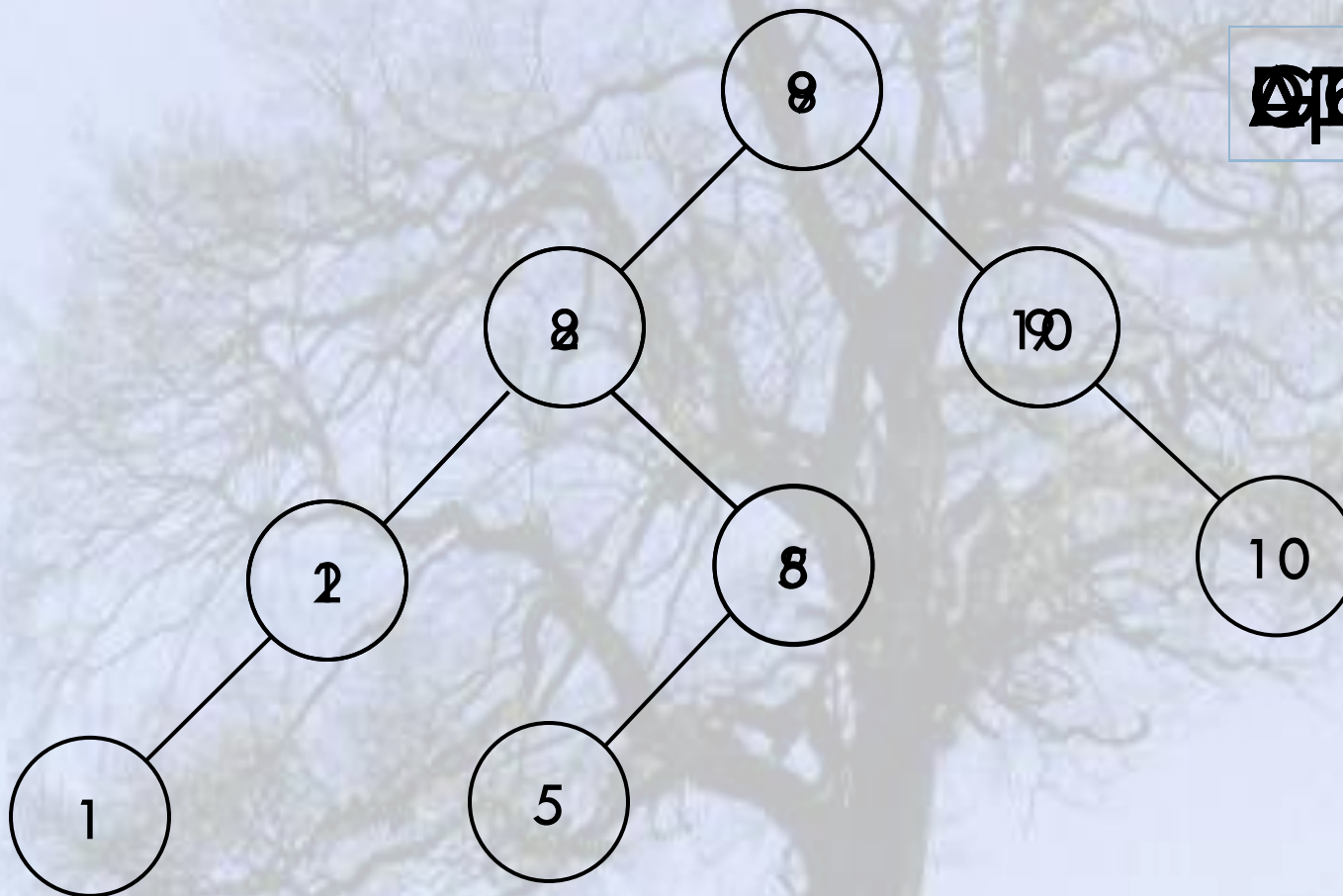
Arbres

AVL

Equilibrage

Exemple 1 (animé)

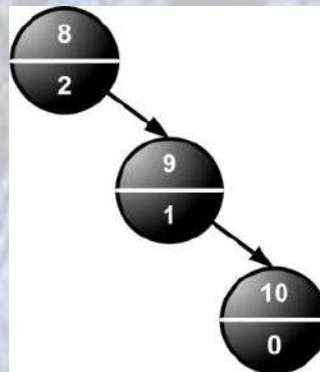
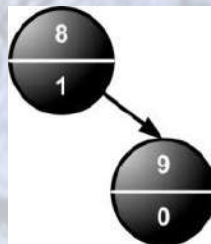
76



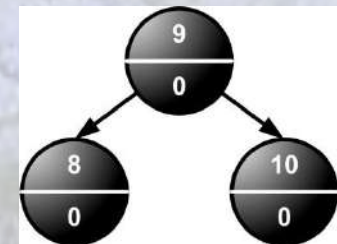
Exemple 2 (mode story board)

77

- Ajout, dans cet ordre, de :
 - ▣ 8, 9, 10, 2, 1, 5, 3, 6, 4, 7, 11, 12



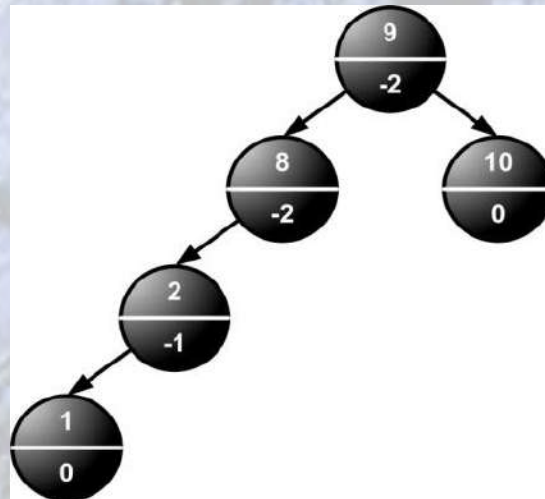
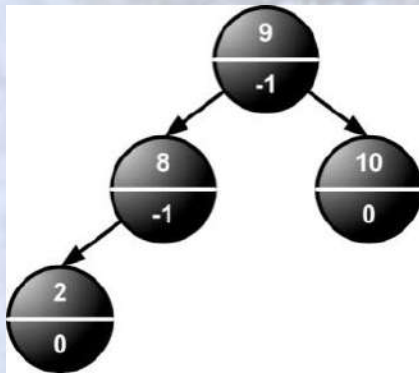
→ DD →



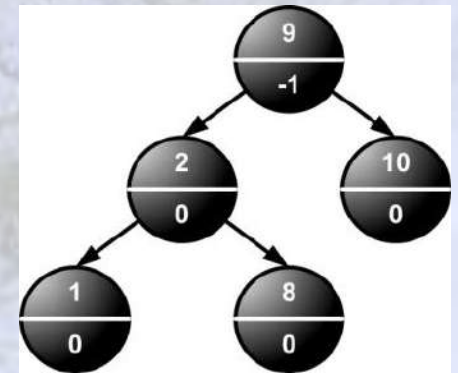
Exemple 2 (mode story board)

78

- Ajout, dans cet ordre, de :
 - ▣ 8, 9, 10, **2**, **1**, 5, 3, 6, 4, 7, 11, 12



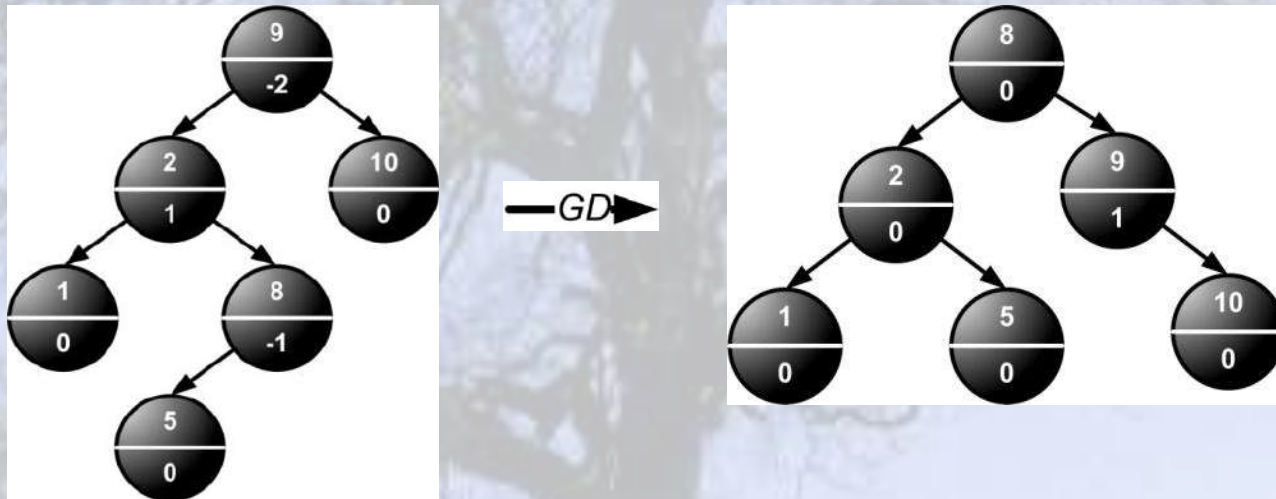
→ GG →



Exemple 2 (mode story board)

79

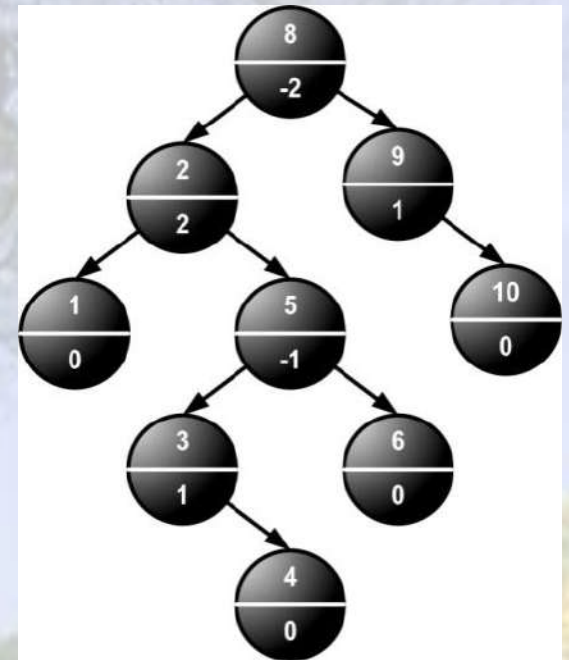
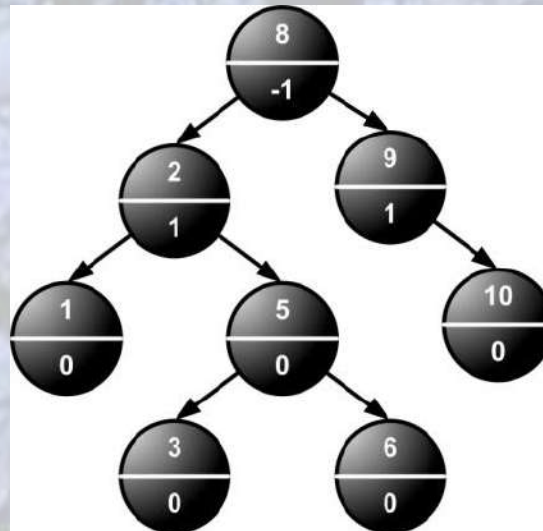
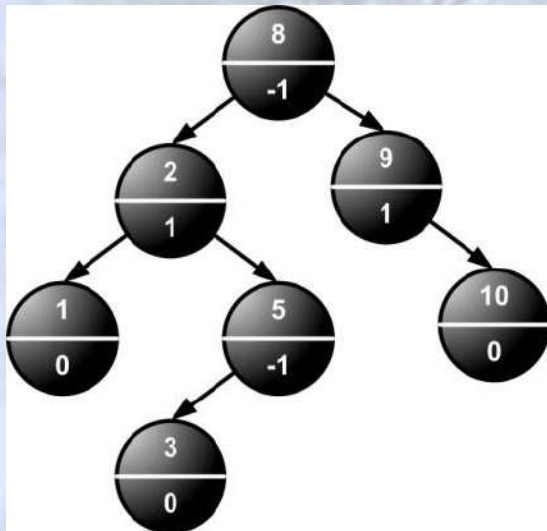
- Ajout, dans cet ordre, de :
 - ▣ 8, 9, 10, 2, 1, 5, 3, 6, 4, 7, 11, 12



Exemple 2 (mode story board)

80

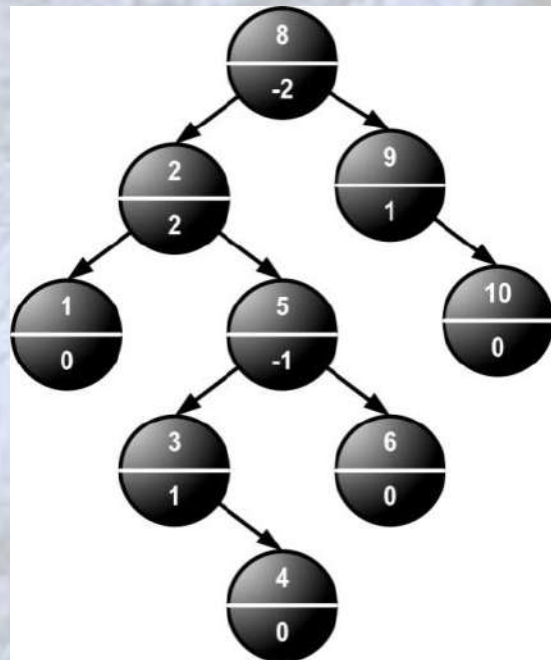
- Ajout, dans cet ordre, de :
 - ▣ 8, 9, 10, 2, 1, 5, 3, 6, 4, 7, 11, 12



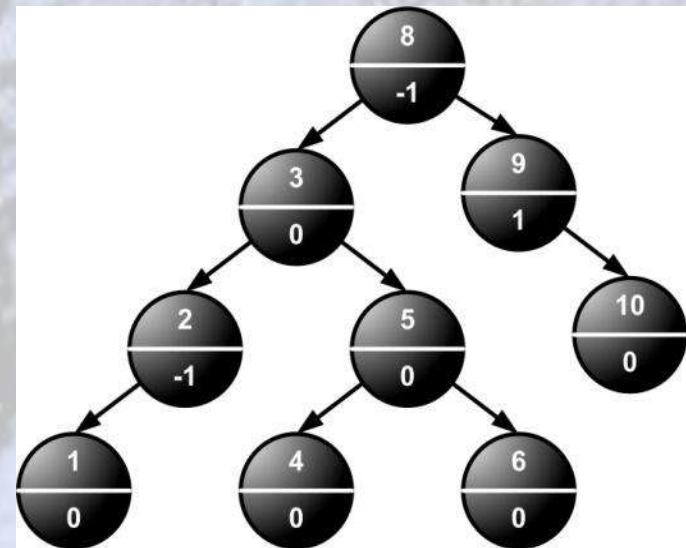
Exemple 2 (mode story board)

81

- Ajout, dans cet ordre, de :
 - ▣ 8, 9, 10, 2, 1, 5, 3, 6, 4, 7, 11, 12



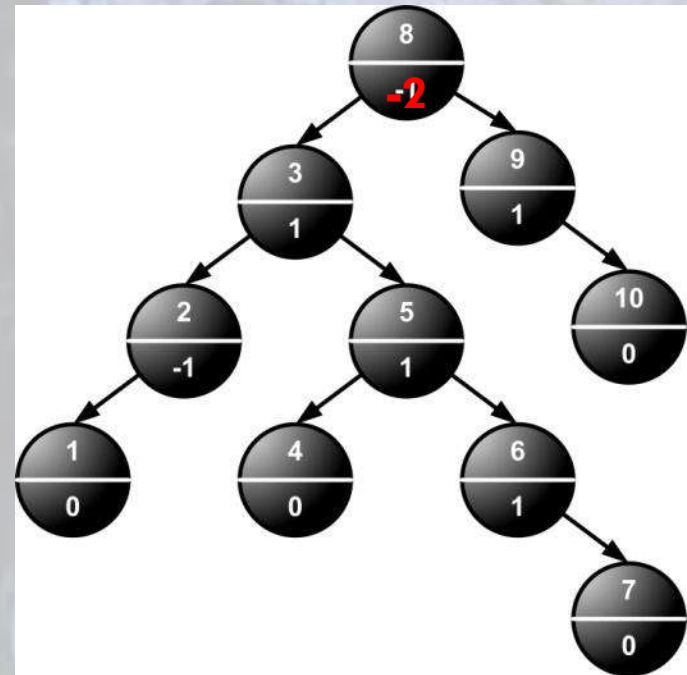
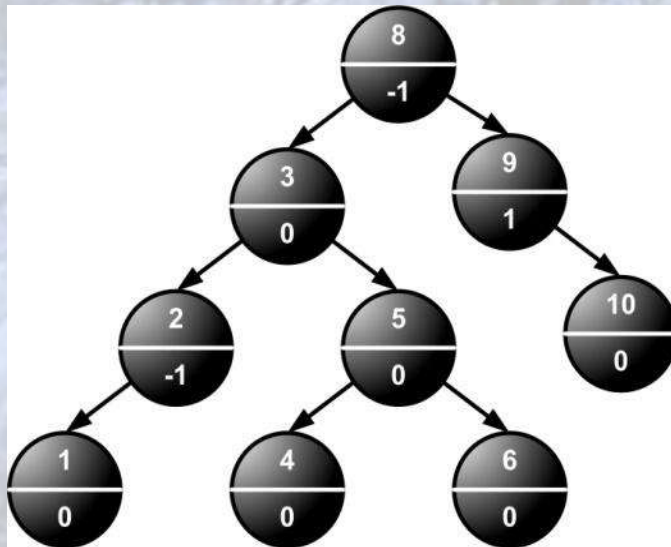
→ DG →



Exemple 2 (mode story board)

82

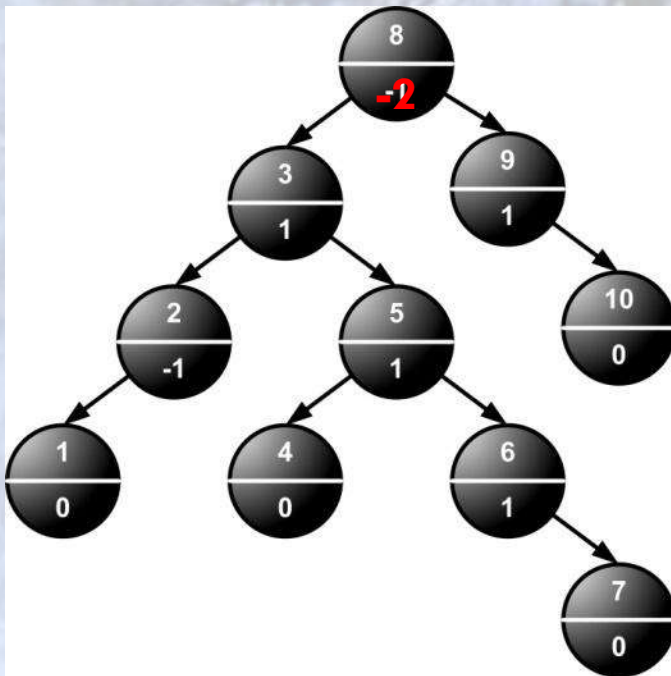
- Ajout, dans cet ordre, de :
 - ▣ 8, 9, 10, 2, 1, 5, 3, 6, 4, **7**, 11, 12



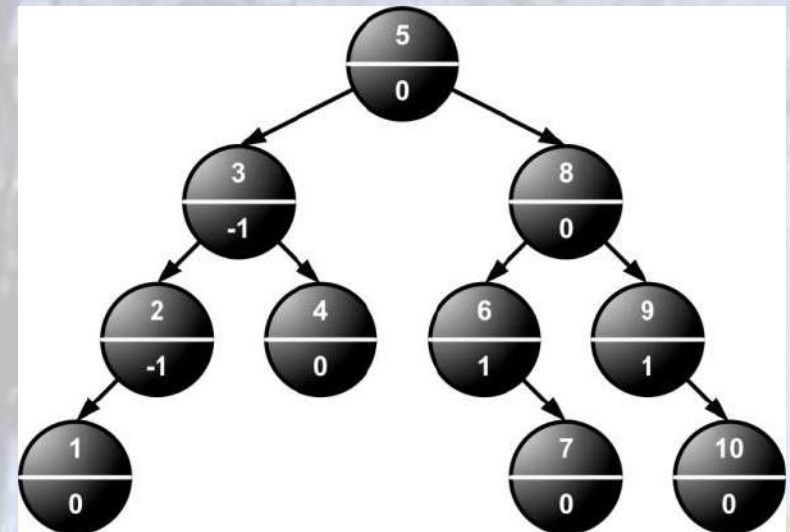
Exemple 2 (mode story board)

83

- Ajout, dans cet ordre, de :
 - ▣ 8, 9, 10, 2, 1, 5, 3, 6, 4, **7**, 11, 12



→ GD →



Arbres

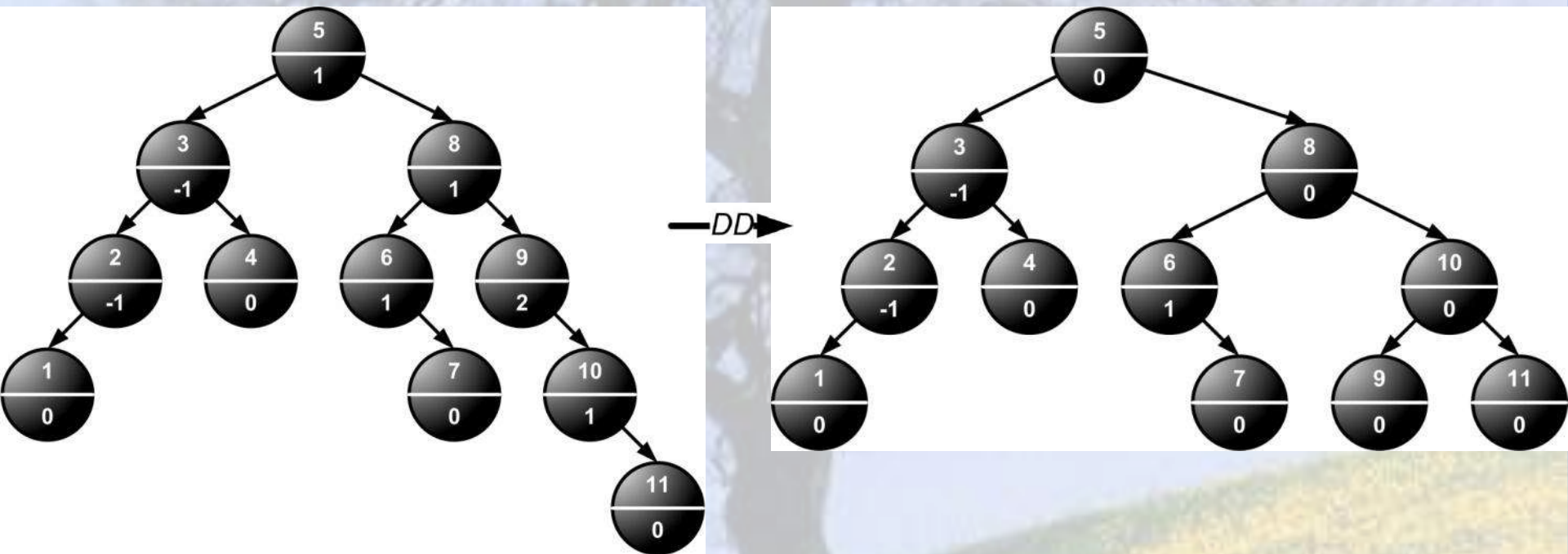
AVL

Equilibrage

Exemple 2 (mode story board)

84

- Ajout, dans cet ordre, de :
 - ▣ 8, 9, 10, 2, 1, 5, 3, 6, 4, 7, **11**, 12



Arbres

AVL

Equilibrage

Exemple 2 (mode story board)

85

- Ajout, dans cet ordre, de :
 - ▣ 8, 9, 10, 2, 1, 5, 3, 6, 4, 7, 11, 12

