centralelille

AUDACE • EXIGENCE • RESPECT

Algorithmique Avancée & Programmation

Un grand merci à Christian Vercauter, rédacteur des documents dont ce poly s'inspire

Séances 4 et 5: fil rouge

Test 4

- QCM moodle
- 20 minutes

Cadrage séance 4 :

- Test papier : questions sur les tris, application du théorème général
  - Surtout théorique
- Présentation du fil rouge de l'année
- Structures de données, principaux algorithmes associés
- Constitution des groupes
  - 4 étudiants par groupe
- Mise en oeuvre la démarche :
  - Conception
  - Développement
  - Evaluation

TEA (indicatif)

- Préparation du travail sous forme d'un document de conception (à définir) :
  - Architecture du code
  - Prototypes des fonctions
  - ...
- Rendre la conception qq jours avant la séance pour que l'enseignant ait le temps de la lire

Après cette séance, vous devez :

- Avoir compris le cahier des charges du fil rouge
- Avoir compris les algorithmes sous-jacents au sujet considéré
- Avoir sélectionné les solutions techniques que vous emploierez, et les avoir organisées dans une architecture bien documentée
- Vous être réparti les tâches de développement à réaliser

Cadrage séance 5 :

- Pas de test en séance
- Retours et conseils individuels sur leur travail de conception
- Séance de développement avec possibilité de demander des conseils à l'intervenant présent

TEA (indicatif)

- Poursuite du travail sur le fil rouge, qui devra être rendu (code + rapport) et sera évalué
  - Utilisation de tests de similarité de code source pour identifier les plagiats éventuels
- Mini-challenge entre les groupes pour déterminer un classement
  - Le classement comptera pour partie dans la note du fil rouge

Fil Rouge 2021 : Arbres
Plan

- Arbres binaires : définitions, propriétés
- Arbres binaires de recherche : ABR
  - Implémentation
  - Parcours d'arbres et affichage graphique
- Arbres équilibrés : AVL
  - Rééquilibrages
  - Implémentation
  - Complexité
- Présentation du fil rouge
- Code Couleur

Fil Rouge 2020
Graphes

- Graphes : définition, représentation
- Algorithmes de recherche de plus courts chemins

Fil Rouge 2022
Arbres partiellement ordonnés

- Arbres : définitions
- Arbres partiellement ordonnés : APO
- Tri pas tas
- Minimier, Minimier indirect
- Codage de Huffman

Fil Rouge 2023
Jeux combinatoires abstraits

- Minimax
- Élagage alpha-beta
- Tables de hachage

centralelille

AUDACE • EXIGENCE • RESPECT

3h30

Arbres binaires : Définitions

Arbre informatique

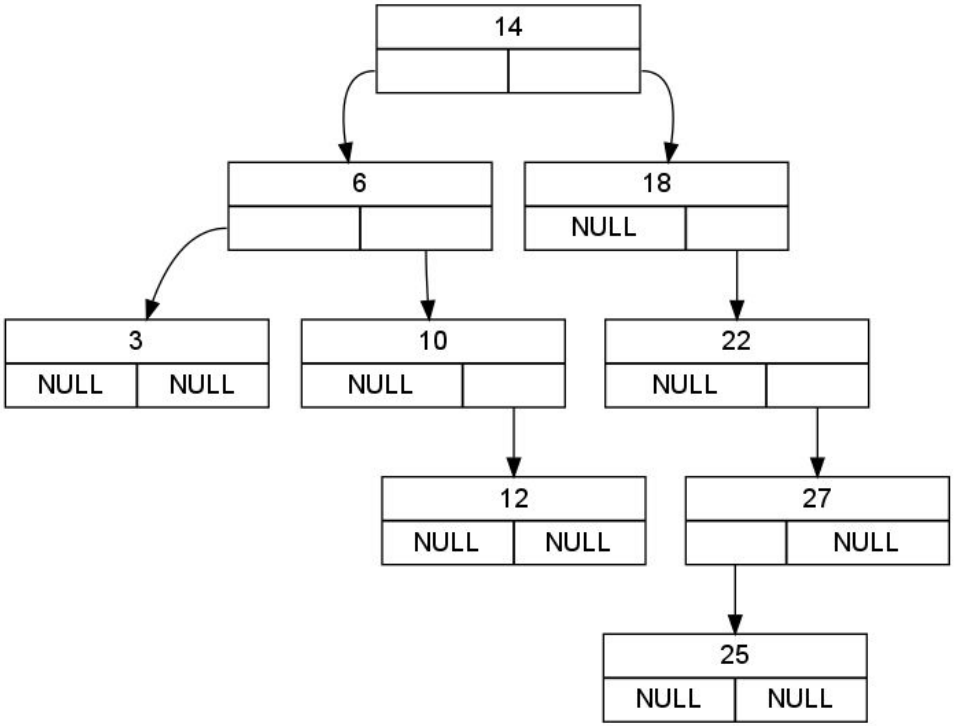
- En informatique, un arbre est une structure de données permettant d'organiser des données de manière hiérarchique :
  - Système de fichiers et de répertoires d'un système d'exploitation
  - Structure d'un document : volume, chapitre, sous-chapitre, paragraphes
  - Syntaxe d'un langage de programmation...
- Analogie avec les arbres généalogiques et arbres végétaux :
  - Nœud père, nœud fils, nœud frère
  - Racine, feuilles, branches, ...

Arbre : définitions

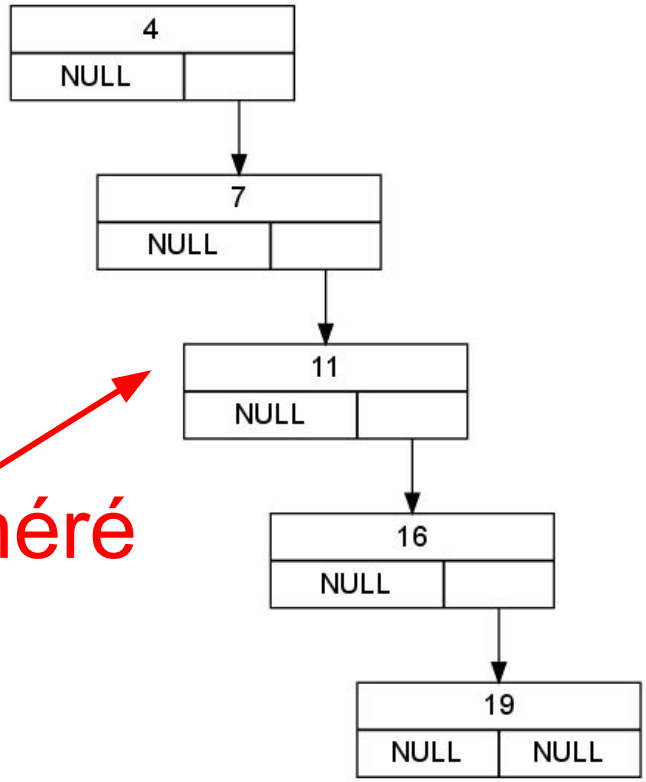
- Nœud : cellule portant l'information et des liens vers d'autres cellules
- Racine : premier nœud de l'arbre, n'a pas de père
- Feuille : nœud n'ayant pas d'enfants
- Nœud interne : nœud ayant au moins un enfant
- Branche : suite de nœuds, du nœud racine jusqu'à une feuille

Arbre Binaire

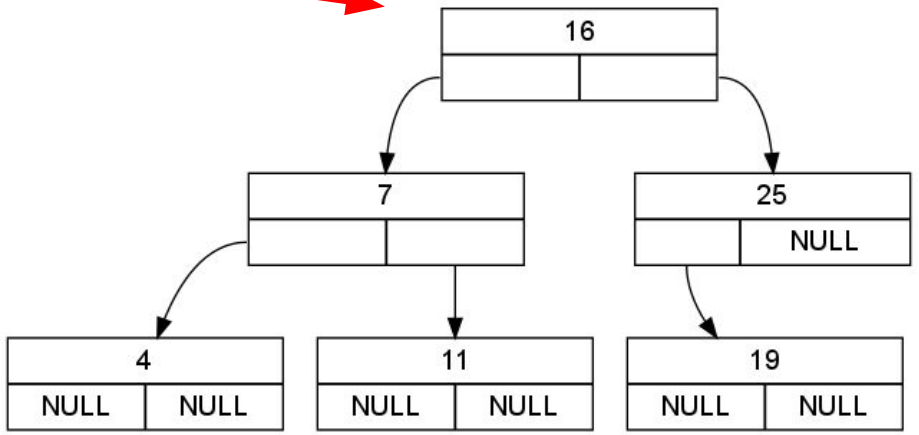
- Un arbre binaire peut être :
  - Un arbre vide
  - Un arbre constitué d'une racine reliée à, au plus, **deux sous-arbres binaires**
- Définition récursive !



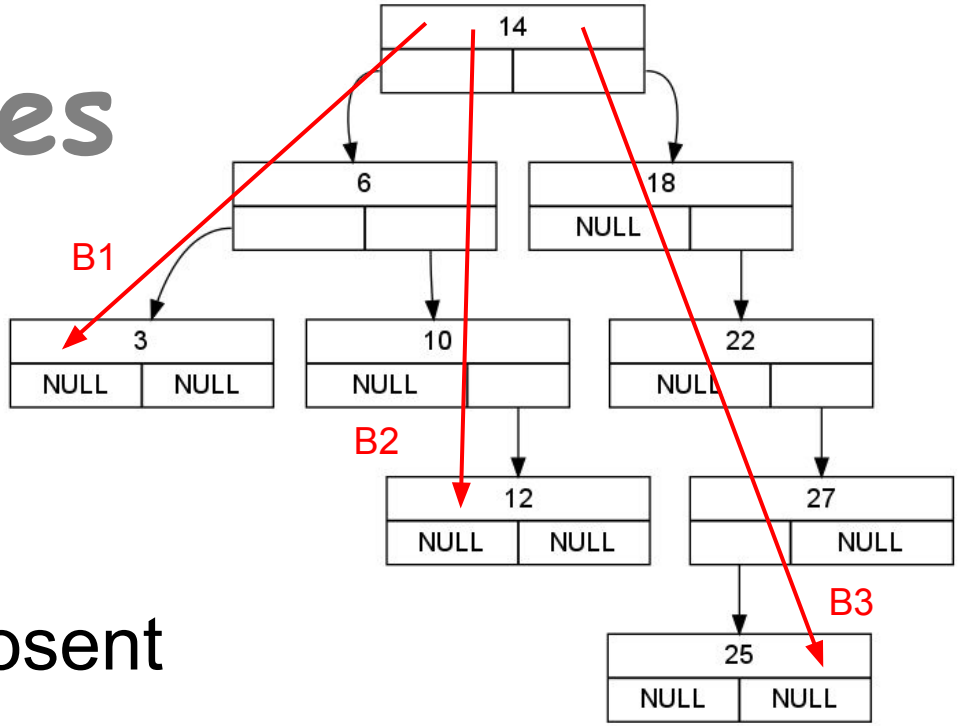
Types d'arbres binaires



- Complet
  - Tous les niveaux sont remplis
- Dégénéré
- Quasi-complet
  - Seul le dernier niveau est incomplet
  - Il peut manquer quelques feuilles



Propriétés des arbres (binaires ou non)



- Longueur d'une branche :
  - Nombre d'arcs qui la composent
- Hauteur d'un arbre :
  - Longueur de la plus longue branche
  - -1 s'il est vide, 0 si un seul nœud
- Profondeur d'un nœud :
  - Nombre d'arcs sur la branche qui mène à ce nœud
  - La racine est un nœud de profondeur 0

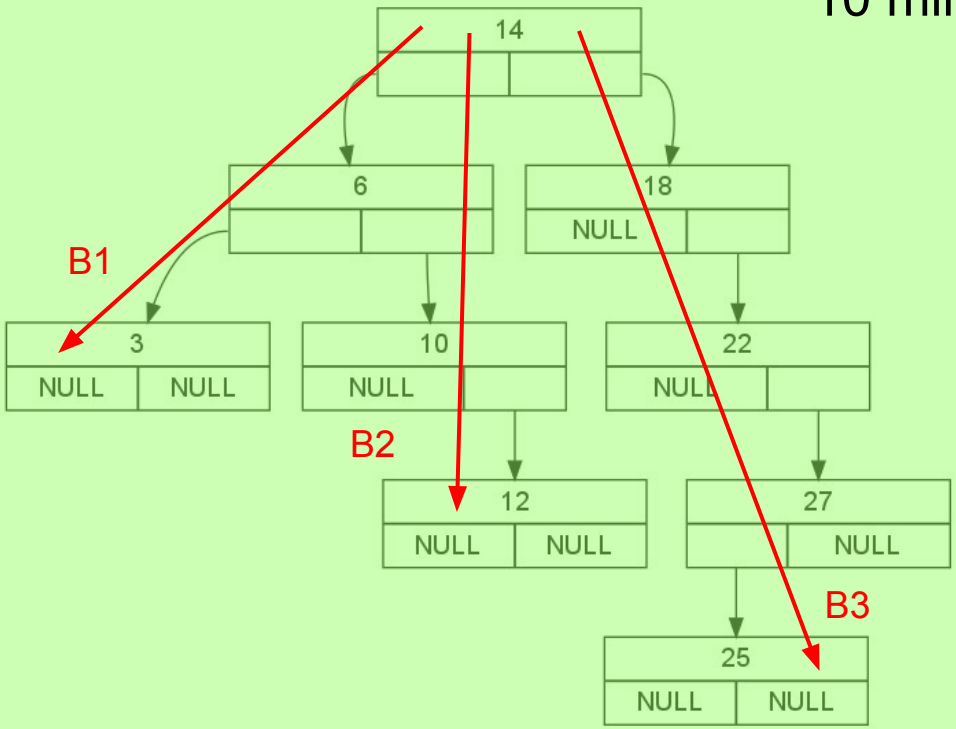
Exercice
Propriétés des arbres binaires

Longueur d'une branche : Nombre d'arcs qui la composent
Hauteur d'un arbre : Longueur de la plus longue branche (-1 s'il est vide)
Profondeur d'un nœud : Nombre d'arcs sur la branche qui mène à ce nœud
La racine est un nœud de profondeur 0

3h20

10 min

- Longueur de chaque branche ?
- Hauteur de l'arbre ?
- Profondeur de chaque nœud ?

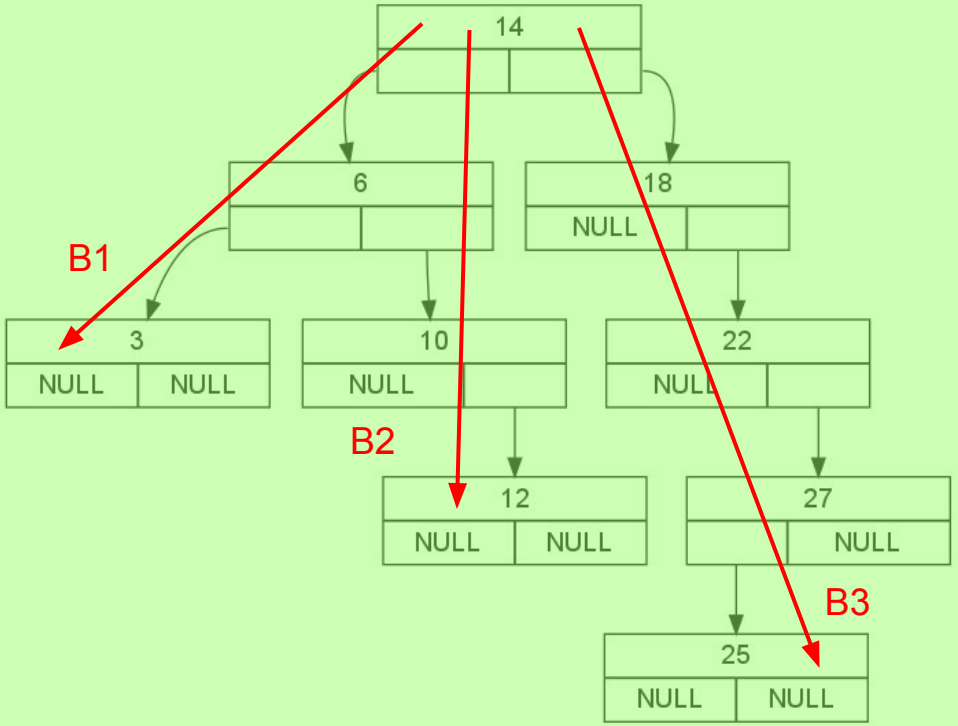


Exercice
Propriétés des arbres binaires

Longueur d'une branche : Nombre d'arcs qui la composent
Hauteur d'un arbre : Longueur de la plus longue branche (-1 s'il est vide)
Profondeur d'un nœud : Nombre d'arcs sur la branche qui mène à ce nœud
La racine est un nœud de profondeur 0

- L(B1) = 2
- L(B2) = 3
- L(B3) = 4

  ⇒ Hauteur de l'arbre = 4
- P(14) = 0
- P(6) = P(8) = 1
- P(3) = P(10) = P(22) = 2
- P(12) = P(27) = 3
- P(25) = 4



Exercice corrigé
Propriétés des arbres binaires

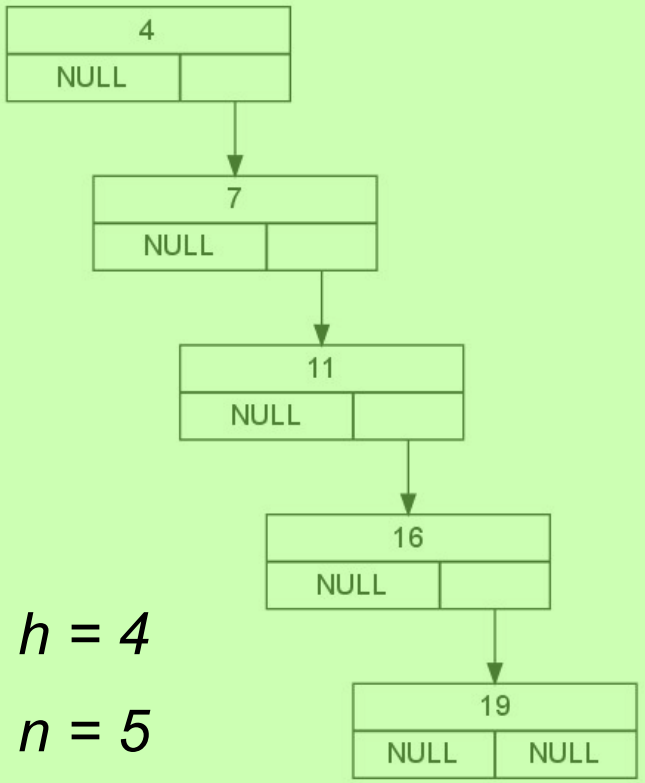
3h10

20 min

- Nombre minimum de noeuds d'un arbre binaire de hauteur h ?
- Nombre de nœuds d'un arbre binaire complet de hauteur h ?
- Nombre maximum de feuilles d'un arbre binaire de hauteur h ?
- Nombre de nœuds d'un arbre binaire quasi-complet de hauteur h ?
- Hauteur minimale d'un arbre binaire de n nœuds ?
- Hauteur maximale ?

Exercice
Propriétés des arbres binaires

- Nombre minimum de nœuds d'un arbre binaire de hauteur h ?
  - Cas d'un arbre dégénéré
  - $n_{min} = h+1$

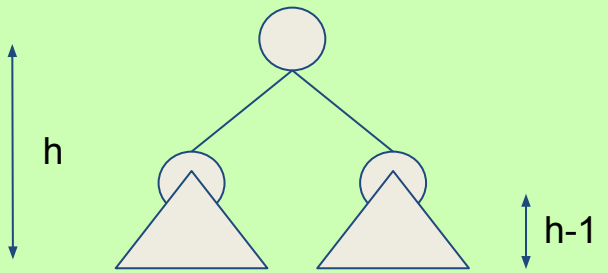


h = 4

n = 5

Exercice
Propriétés des arbres binaires

- Nombre de nœuds d'un arbre binaire complet de hauteur h ?
  - C'est le nombre maximum de nœuds que l'on peut implanter dans un arbre de hauteur h
  - $n_{max}(h) = 2^{h+1} - 1$
- Par récurrence :
  - $n_{max}(0) = 1$
  - $n_{max}(1) = 3$ ...
  - $n_{max}(h) = 1 + 2 * n_{max}(h-1)$



Exercice
Propriétés des arbres binaires

- Nombre maximum de feuilles d'un arbre binaire de hauteur h ?
  - Construction d'un arbre complet
  - $f_{max}(h) = 2^h$
- A chaque niveau supplémentaire, on ajoute 2 fois plus de feuilles qu'au niveau précédent
- $f_{max}(0) = 1$
- $f_{max}(1) = 2$
- $f_{max}(h) = 2 * f_{max}(h-1)$

Exercice
Propriétés des arbres binaires

- Nombre de nœuds d'un arbre binaire quasi-complet de hauteur h ?
- Seul le dernier niveau est incomplet
- Le sous-arbre racine sans ses feuilles est complet
  - Il est de hauteur h-1
- $n_{qc}(h) = n_{max}(h-1)$ + nombre de noeuds du dernier niveau (feuilles)
- $n_{qc}(h) = 2^h - 1$ + nombre de feuilles d'un arbre de hauteur h
- Nombre maximum de feuilles d'un arbre de hauteur h : 2^h
- $2^h - 1 + 1 \leq n_{qc}(h) \leq 2^h - 1 + 2^h$
- $2^h \leq n_{qc}(h) \leq 2^{h+1} - 1$

Exercice
Propriétés des arbres binaires

- Hauteur minimale d'un arbre binaire de n nœuds ?
  - On remplit tous les niveaux au maximum pour réduire la hauteur de l'arbre
    ⇒ Arbre quasi-complet : $2^h \leq n_{qc}(h) \leq 2^{h+1} - 1$
- $2^{h_{min}} \leq n \leq 2^{h_{min}+1} - 1$
- $2^{h_{min}} \leq n$ et $n+1 \leq 2^{h_{min}+1}$
- $h_{min} \leq \log_2(n)$ et $\log_2(n+1) \leq h_{min}+1$
- $\log_2(n+1) - 1 \leq h_{min} \leq \log_2(n)$ *NB : h_{min} est un entier !*
- $h_{min} = \lceil \log_2(n+1)-1 \rceil = \lceil \log_2(n+1) \rceil - 1 = \lfloor \log2(n) \rfloor$

Cf. feuille de calcul :
Hauteur minimale d'un arbre binaire de n nœuds

$\lceil n \rceil$: plus petit entier supérieur ou égal à n
$\lfloor n \rfloor$: plus grand entier inférieur ou égal à n

$h_{min} = \lceil \log_2(n+1)-1 \rceil = \lfloor \log2(n) \rfloor$

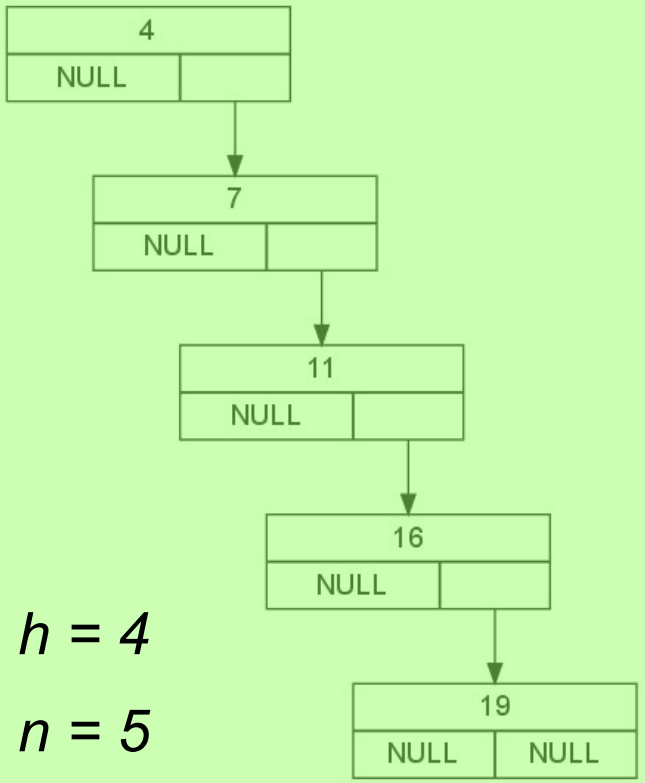
- $\log_2(n+1) - 1 \leq h_{min} \leq \log_2(n)$
- $\log_2(n) - \log_2(n+1) + 1 = 1 - \log_2(1+1/n) < 1$
  - Donc il n'y a qu'un seul entier au plus dans l'intervalle $[\ \log_2(n+1)-1,\ \log_2(n)\ ]$
- Par les propriétés des parties entières et puisque h_{min} est un entier :
- $\log_2(n+1) - 1 \leq \lceil \log2(n+1)-1 \rceil \leq h_{min} \leq \lfloor \log2(n) \rfloor \leq \log\_2(n)$
  - Il y a trois entiers dans l'intervalle considéré, ils sont donc égaux...

Exercice
Propriétés des arbres binaires

- Hauteur maximale d'un arbre binaire de n nœuds ?
- Cas d'un arbre dégénéré
- $h_{max}(n) = n-1$

- De façon générale :

  $\lfloor \log_2(n) \rfloor \leq h < n$



h = 4

n = 5

Complexité dans les arbres binaires

- La plupart des opérations réalisées sur un arbre binaire ont un coût proportionnel à la hauteur de l'arbre binaire
- Coût en **Ω(** log(n) **)** et **O(** n **)**
  - La complexité est en **Θ(** log(n) **)** si l'arbre est équilibré
  - La complexité se dégrade en **Θ(** n **)** si l'arbre est déséquilibré
- Rendre optimales les opérations sur un arbre binaire de recherche, c'est réduire sa hauteur
  - On applique des transformations pour rééquilibrer l'arbre binaire après insertion ou suppression : Arbres AVL

centralelille

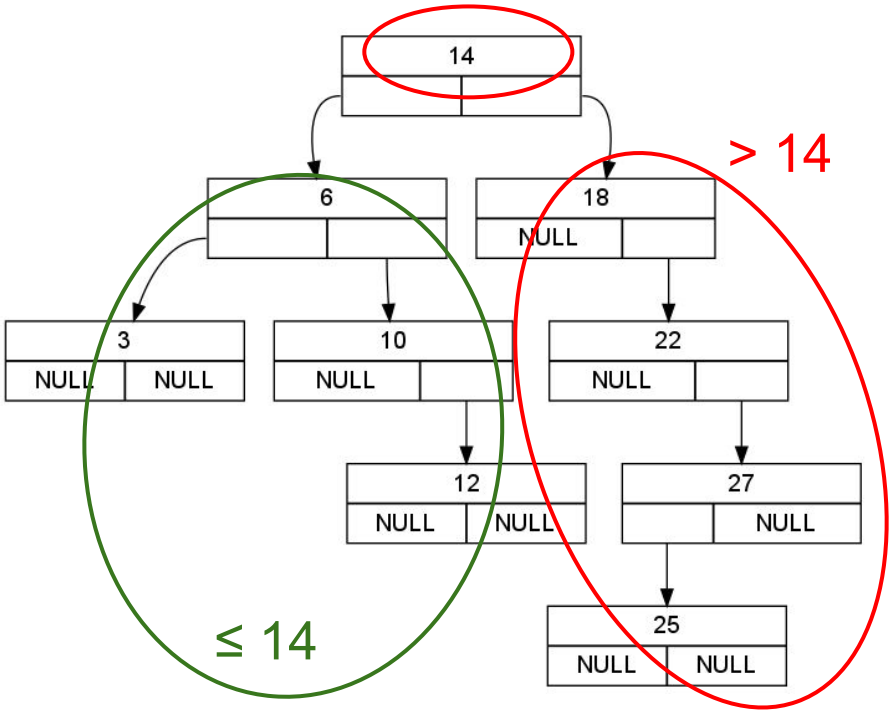
AUDACE • EXIGENCE • RESPECT

2h50

Arbres binaires de recherche (ABR)

Arbre binaire de recherche (ABR) Binary Search Tree (BST)

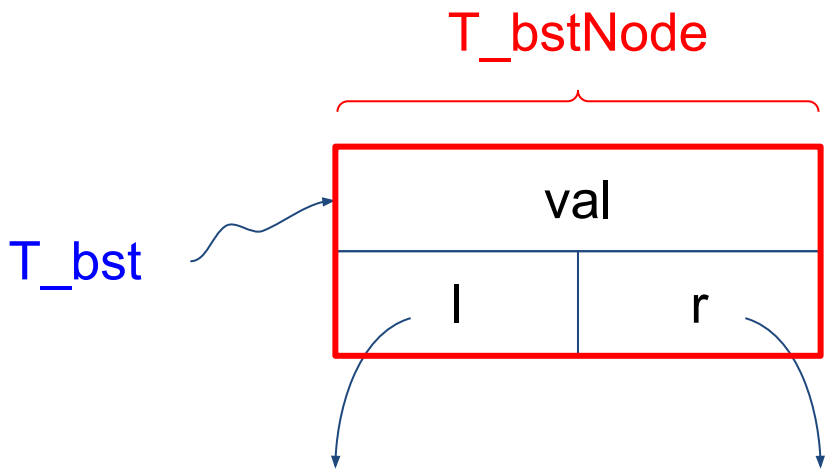
- Un arbre binaire de recherche (ABR) est un arbre binaire doté d'une relation d'ordre :
  - Les valeurs portées par les nœuds du sous-arbre gauche sont inférieures ou égales à celle de la racine
  - Les valeurs portée par les noeuds du sous-arbre droit sont strictement supérieures à la racine
  - Ces propriétés doivent être vérifiées pour chacun de ses nœuds



ABR : implémentation

- Structure dynamique et auto-référente

```
typedef struct tNode{
   T_elt val;
   struct tNode *l;
   struct tNode *r;
} T_bstNode, *T_bst;
// typedef T_bstNode * T_bst;
// T_bstNode ⇔ ABR                T_node ⇔ listes
```



Exercice corrigé
Construction d'un noeud

2h45

15 min

- T_bstNode * newNodeBST(T_elt e)
  - Alloue un noeud pour y stocker un élément
- Que faire si T_elt est/contient l'adresse d'une zone de mémoire qui pourrait être réutilisée ?
  - Cas d'une chaîne de caractères stockée dans un buffer (cf. fgets()) ⇒ Utiliser strdup()
  - Cas d'un tableau d'objets quelconques ⇒ Utiliser memcpy()

  ⇒ Enrichir le module de gestion de T_elt pour prendre en compte ces situations : T_elt eltdup(T_elt)

  ⇒ Ajouter également une fonction permettant de comparer deux T_elt : int eltcmp(T_elt, T_elt)

Exercice corrigé
Insertion d'une valeur dans l'arbre

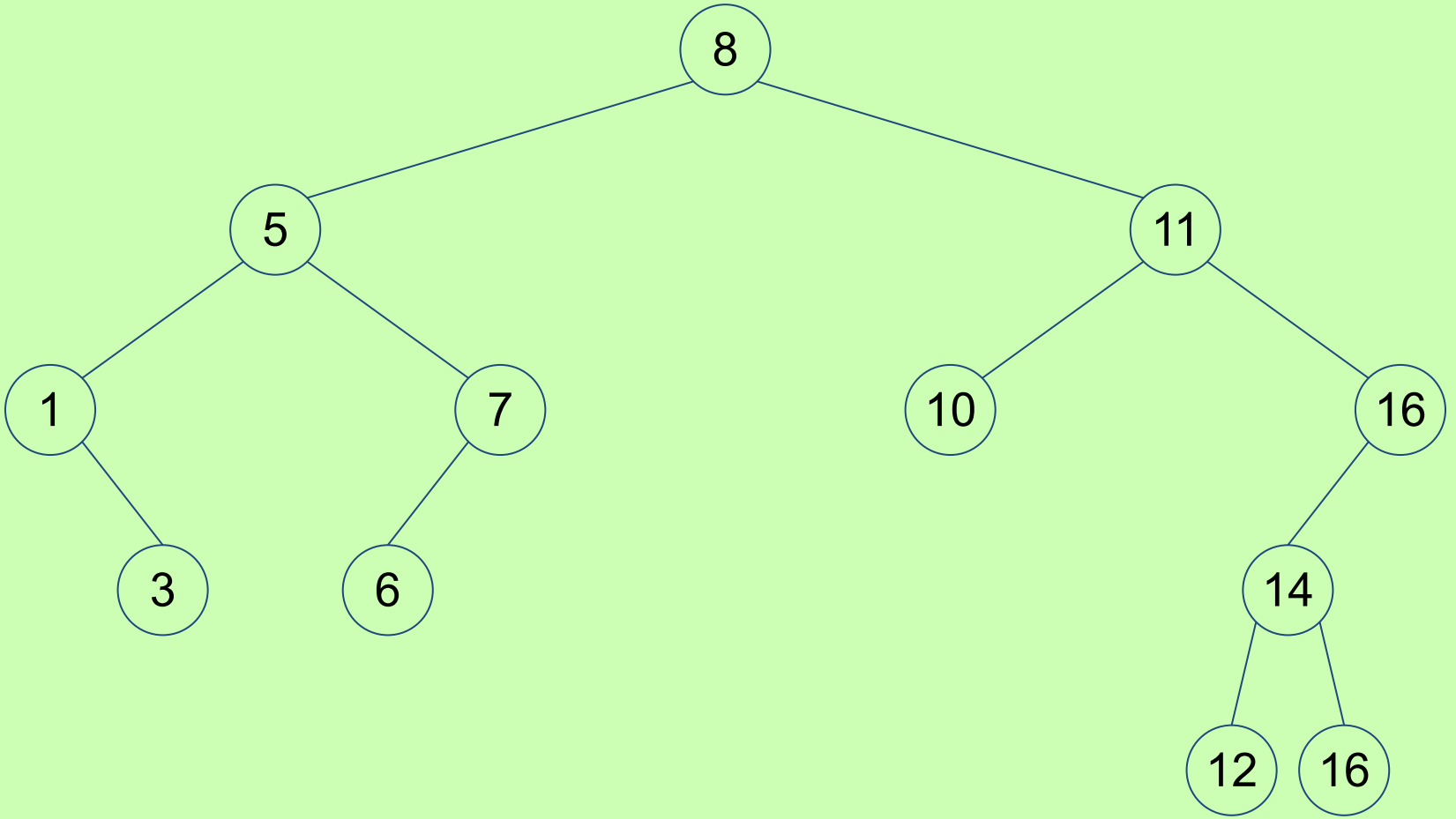
2h30

15 min

- Représenter graphiquement, l'ABR obtenu après insertions successives des éléments suivants :
  8, 11, 16, 5, 7, 14, 6, 1, 3, 10, 12, 16

- T_bst insertBST(T_bst root, T_elt e)
- Insertion d'une valeur dans l'arbre en respectant la relation d'ordre entre les noeuds
  - Définition récursive
  - NB : rien ne garantit que l'ABR obtenu soit équilibré !

8, 11, 16, 5, 7, 14, 6, 1, 3, 10, 12, 16

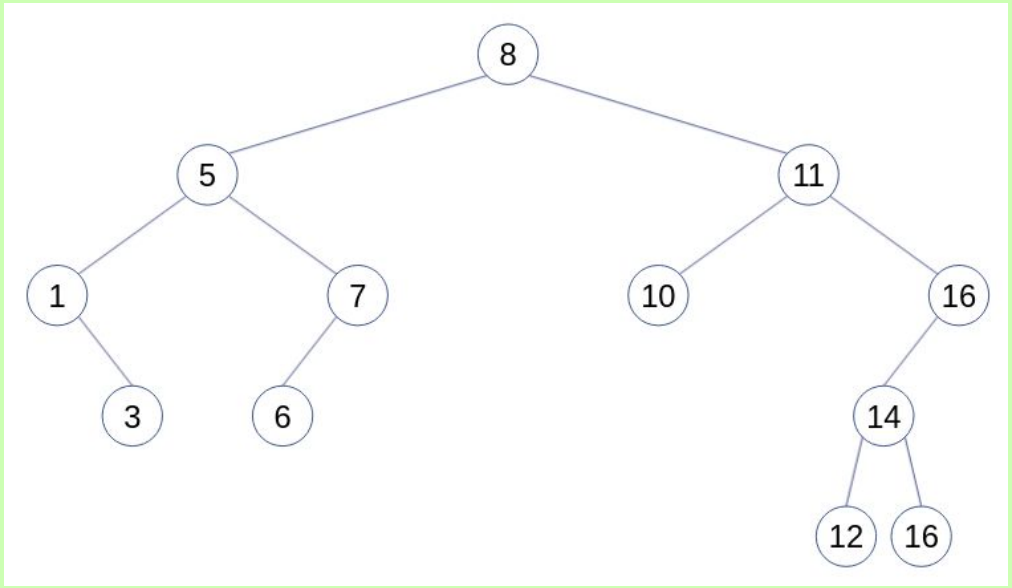


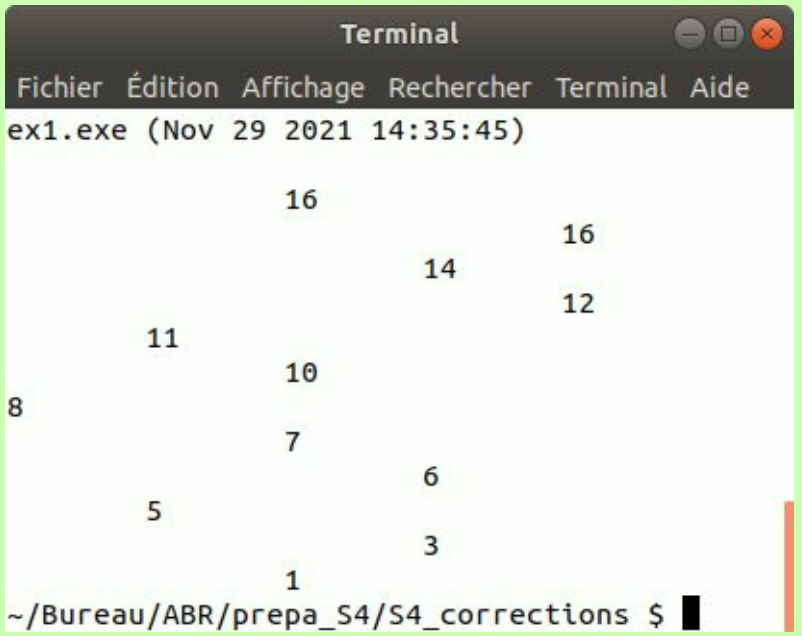
Affichage d'un ABR Format "pseudo-graphique"

2h15

15 min

- void printBST(T_bst root, int indent)
- Afficher les éléments d'un ABR de telle sorte que chaque élément s'affiche sur une ligne distincte, avec une indentation proportionnelle à la profondeur du nœud qui le contient





Exercice
Recherches dans un ABR

2h00

15 min

- T_bstNode * searchBST_rec(T_bst root, T_elt e)
  - Recherche récursive dans un ABR
- T_bstNode * searchBST_it(T_bst root, T_elt e)
  - Recherche itérative dans un ABR

Exercice : développer en récursif
Tester au fur et à mesure

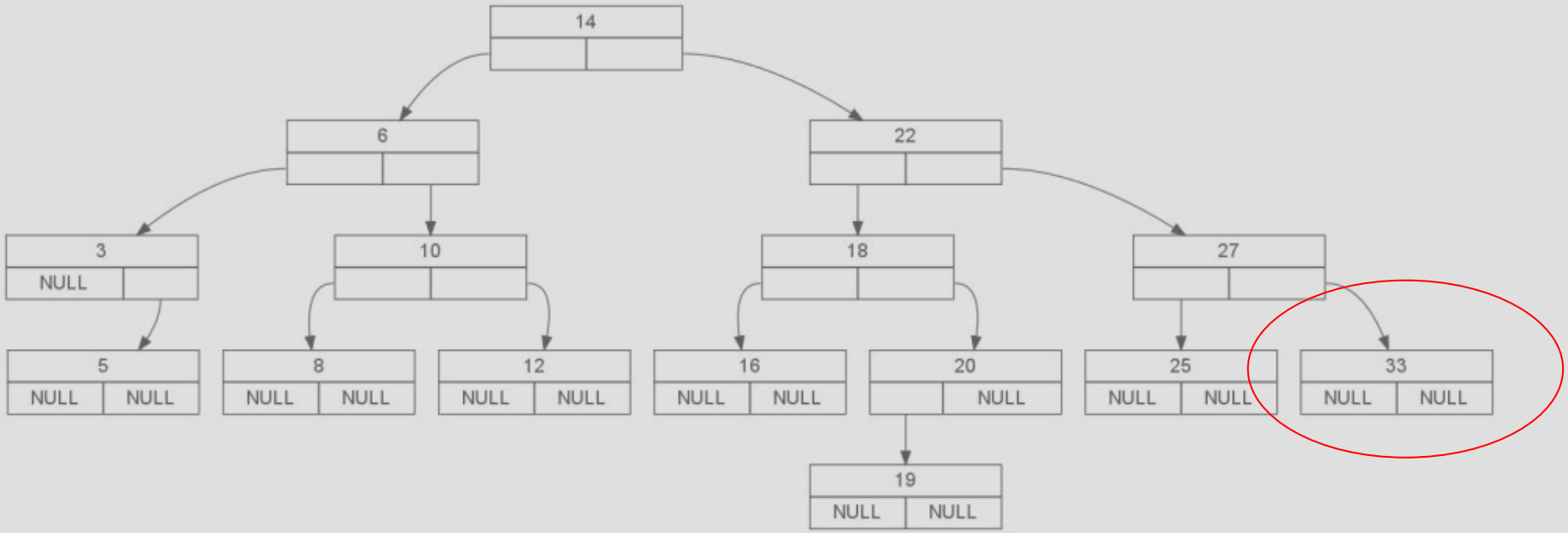
- int heightBST(T_bst)
- int nbNodesBST(T_bst)
- int nbLeavesBST(T_bst)
- T_bstNode * minBST(T_bst)
  - Développer T_bstNode * maxBST(T_bst) en itératif
- int areEqualsBST(T_bst, T_bst)
- int areIsomorphicBST(T_bst ,T_bst)
- T_bst copyBST(T_bst)

Suppression dans un ABR

- Plusieurs cas de figure doivent être considérés selon que le nœud à supprimer :
  - est une feuille ;
  - n'a pas de sous-arbre gauche ;
  - n'a pas de sous-arbre droit ;
  - a un sous-arbre gauche ET un sous-arbre droit.
- Il est nécessaire, dans tous les cas, de rechercher le nœud à supprimer et d'avoir accès à son père
  - Le cas de la suppression de la racine doit être traité de façon spécifique

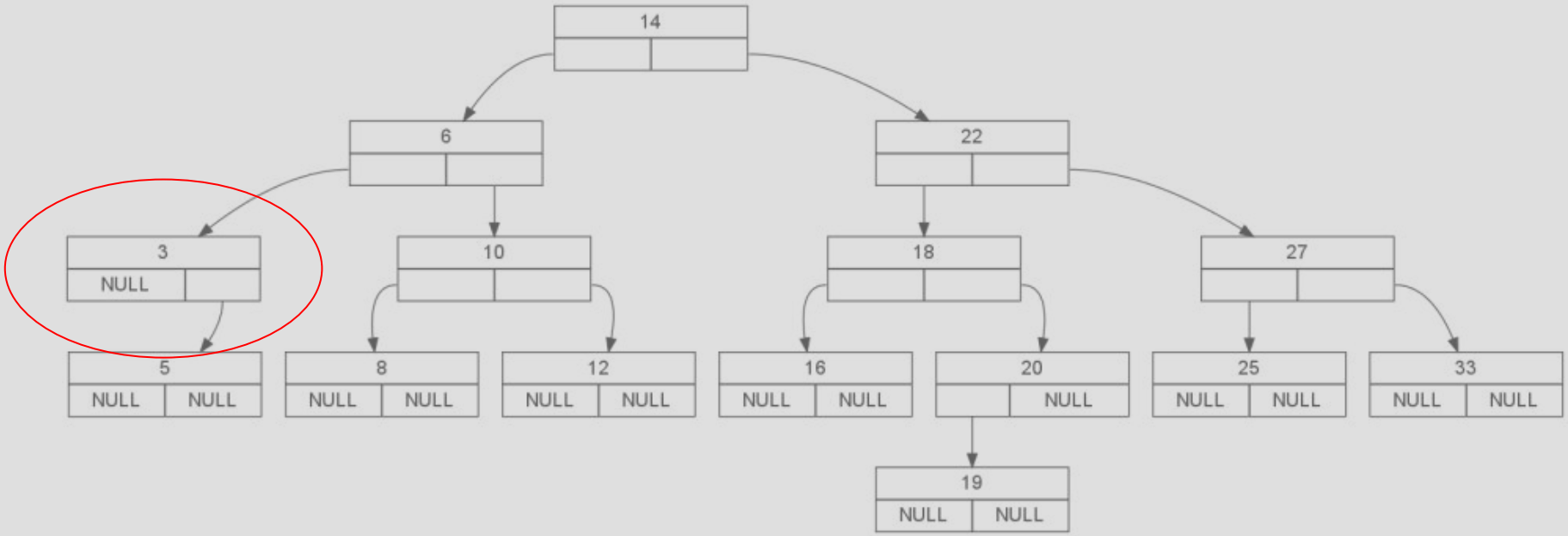
Suppression dans un ABR

- Suppression d'une feuille : 33
  - On supprime simplement le nœud correspondant



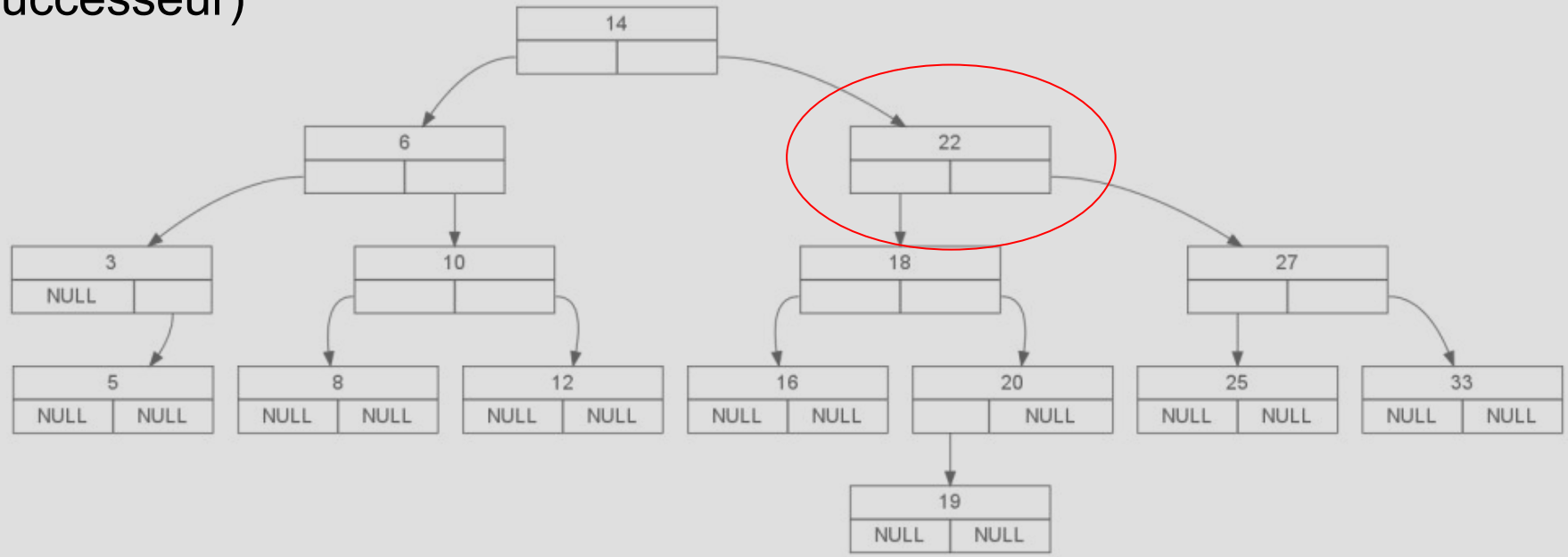
Suppression dans un ABR

- Suppression d'un nœud n'ayant pas de sous-arbre gauche : 3
  - On lui substitue simplement la racine de son sous-arbre droit



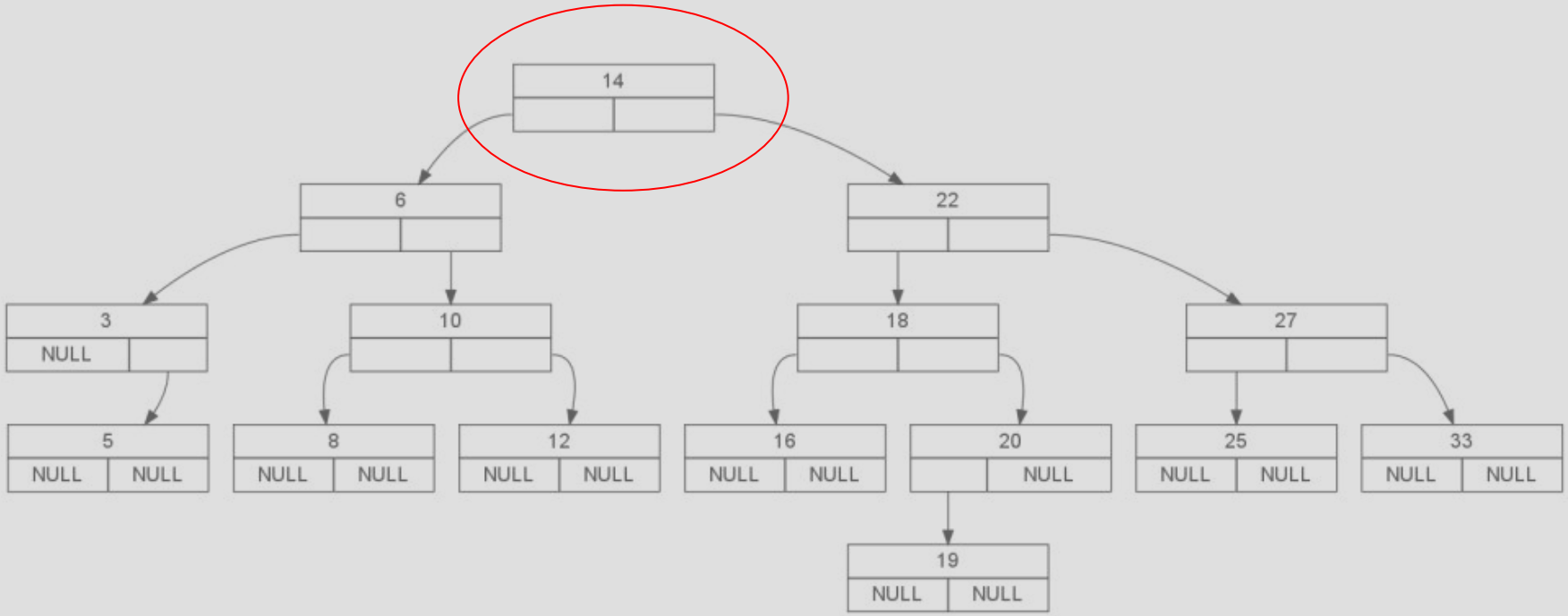
Suppression dans un ABR

- Suppression d'un nœud ayant un ss-arbre gauche et un ss-arbre droit : 22
- On remplace la valeur du noeud par celle du plus grand élément du ss-arbre gauche (soit son prédécesseur). On supprime celui-ci et on « reconnecte »
  - On peut aussi utiliser le plus petit élément du ss-arbre droit (son successeur)



Suppression dans un ABR

- Suppression du nœud racine : 14
- Processus similaire au précédent



centralelille

AUDACE • EXIGENCE • RESPECT

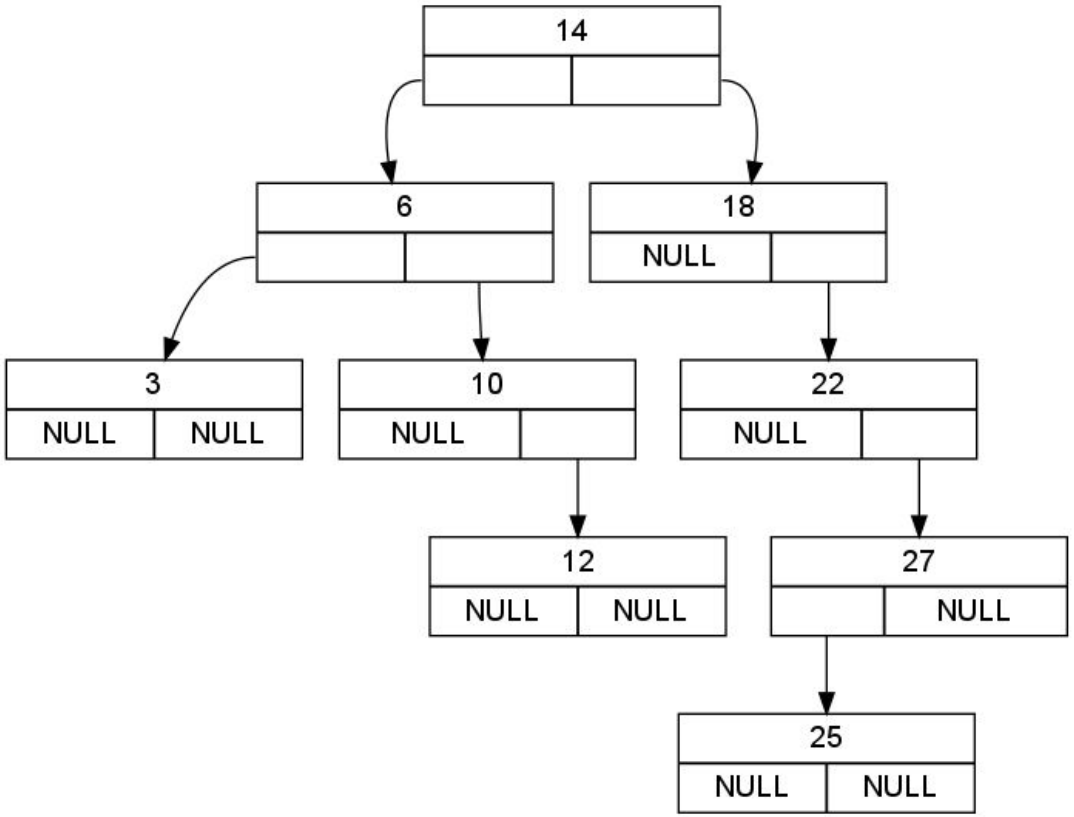
1h30

Parcours d'arbres et affichages

Parcours d'un ABR
Parcours en profondeur (DFS)

- Exploration en profondeur d'abord
  - DFS : Depth First Search
- 3 types de parcours
  - Parcours infixe ou GRD (*inorder* traversal, LNR)
  - Parcours préfixe ou RGD (*preorder* traversal, NLR)
  - Parcours postfixe ou GDR (*postorder* traversal, LRN)
- Réalisation simple des fonctions de parcours en récursif (sinon, besoin d'une pile)

Parcours d'un ABR
Parcours en profondeur (DFS)



- GRD :
  - 3 6 10 12 14 18 22 25 27
- RGD :
  - 14 6 3 10 12 18 22 27 25
- GDR :
  - 3 12 10 6 25 27 22 18 14

Exercice : Parcours récursifs en profondeur

1h25

15 min

- void printInOrderBST(T_bst root)
- void printPostOrderBST(T_bst root)
- void printPreOrderBST(T_bst root)

- Eliminer les appels récursifs terminaux

Dérécursivation d'algorithmes
Récursivité terminale : facile !

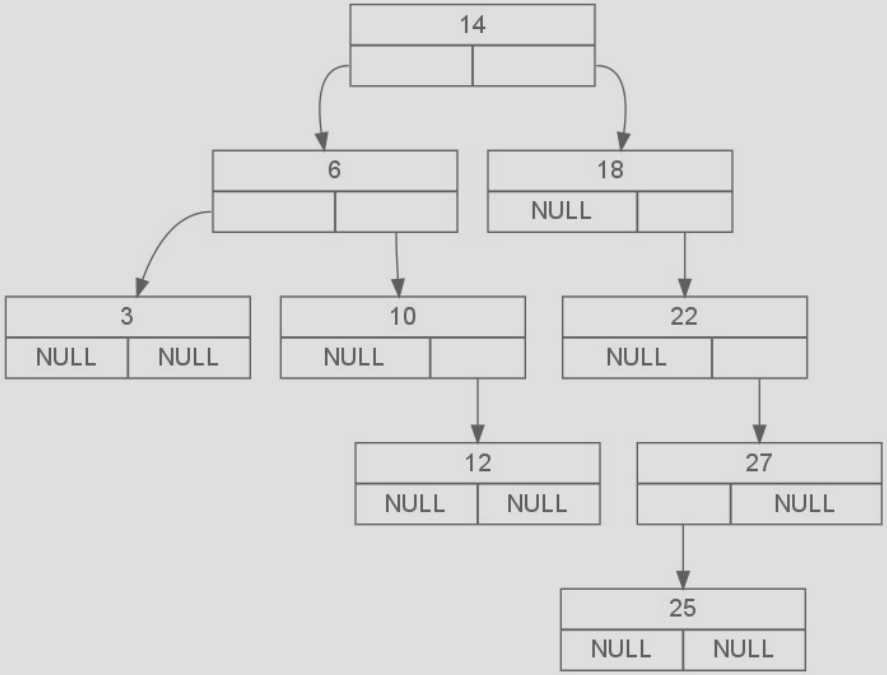
```
Foo(cas)
    Si  (IS_BASE_CASE(cas))
        renvoyer BASE_RESULT(cas)
    Sinon
        next_cas= OPERATIONS(cas)
        renvoyer Foo(next_cas)
```

// appel à Foo() terminal : on renvoie directement le résultat de l'appel récursif

```
Foo(cas)
    aux = cas
    TANT QUE ( ! IS_BASE_CASE(aux))
    FAIRE
        aux= OPERATIONS(aux)
        // aux récupère à chaque itération
        // ce qu'aurait dû traiter l'appel récursif
    FIN FAIRE

renvoyer BASE_RESULT(aux)
```

Parcours itératif en profondeur

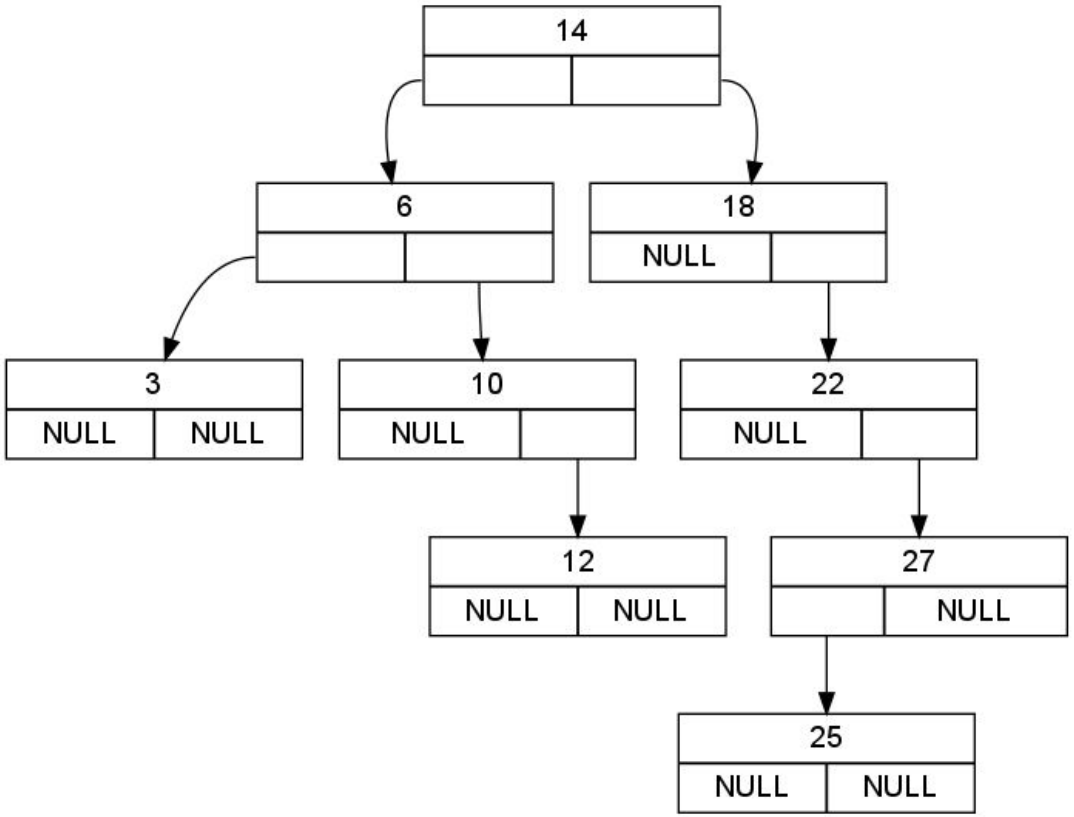


3 6 10 14 18 22 25 27

```
void AfficheGRD(T_Abr a) {
    T_Pile pile = creerPile(0);
    int Fini = 0;
    while (!Fini) {
        if (a != NULL) {
            Push(a, &pile);
            a = a->g;
        }
        else if (!estPileVide(&pile)) {
                a = Pop(&pile);
                printf("%5d\n", a->val);
                a = a->d;
        }
        else Fini = 1;
    }
    supprPile(&pile);
}
```

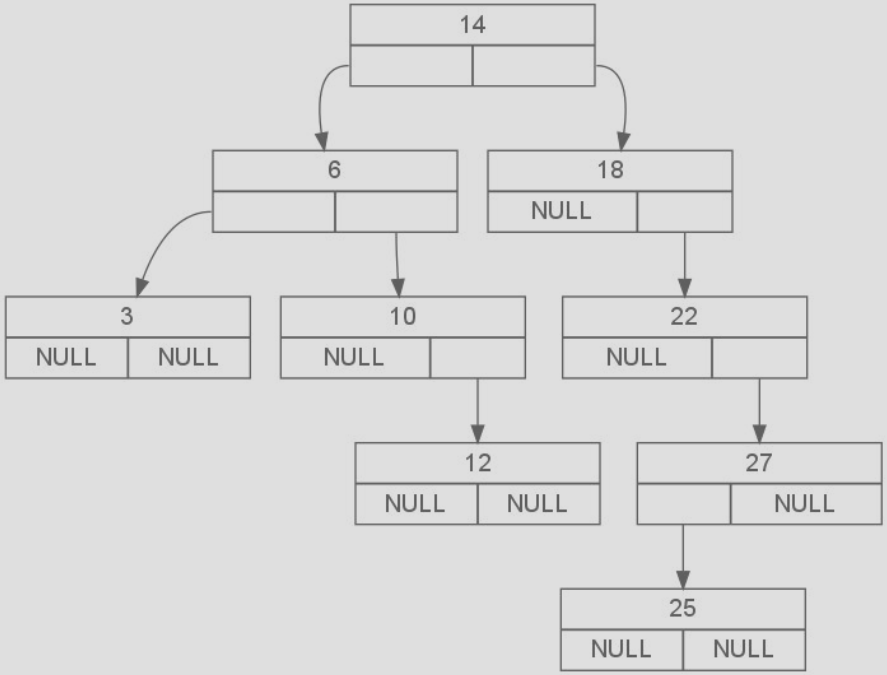
Nécessite l'utilisation d'une pile pour mémoriser les racines des sous-arbres traversés

Parcours en largeur



14 6 18 3 10 22 12 27 25

Parcours en largeur



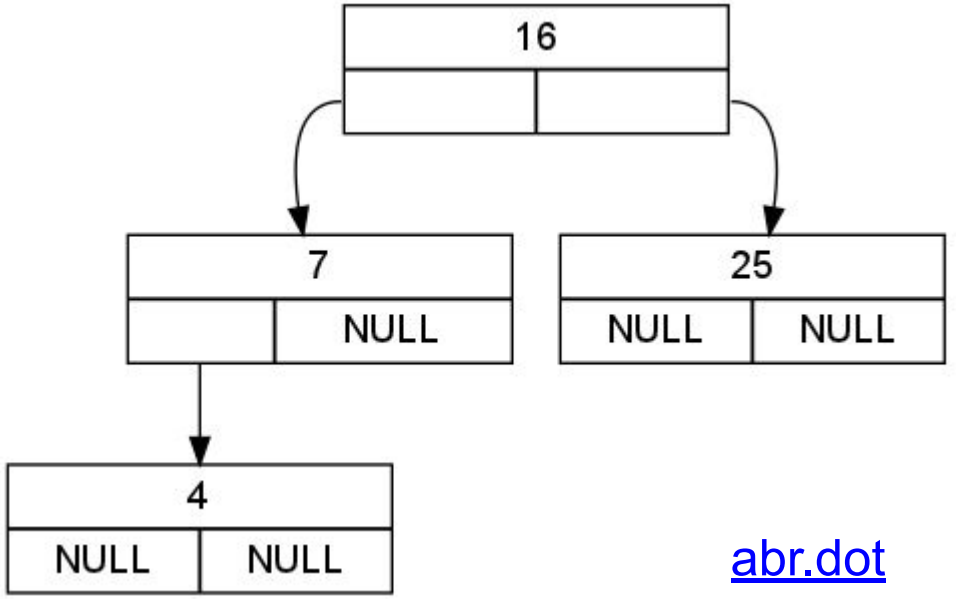
14 6 18 3 10 22 12 27 25

```
void afficherBFS(T_ABR a) {
   T_Fifo fifo;
   if (a == NULL)return ;
   fifo = creerFifo(10);
   ajouterFifo(a, &fifo);
   while (!estFifoVide(&fifo)) {
      a = extraireFifo(&fifo);
      printf("%5d ", a->val);
      if (a->g != NULL)
         ajouterFifo(a->g, &fifo);
      if (a->d != NULL)
         ajouterFifo(a->d, &fifo);
   }
   supprFifo(&fifo);
}
```

Nécessite l'utilisation d'une file (FIFO : First In, First Out) pour mémoriser les racines des sous-arbres restant à explorer

- *2 opérations de base : ajouterFifo, extraireFifo*

Affichage d'un ABR Format dot



NB : les noeuds doivent contenir des valeurs toutes distinctes

abr.dot

```
digraph BST_test {
   node [fontname="Arial", shape=record, width=2];
   16 [label = "{<c> 16 | { <g> | <d>}}"];
   16:g -> 7;
   7 [label = "{<c> 7 | { <g> | <d> NULL}}"];
   7:g -> 4;
   4 [label = "{<c> 4 | { <g> NULL | <d> NULL}}"];
   16:d -> 25;
   25 [label = "{<c> 25 | { <g> NULL | <d> NULL}}"];
}
```

Exemple
Affichage d'un ABR au format dot

1h10

10 min

- Tester sur votre machine Linux ou en ligne :
  - dot -T png abr.dot -o abr.png
  - https://dreampuf.github.io/GraphvizOnline
- void genDot(T_bst root, FILE *fp)
  - Écrit la structure dot correspondant à l'ABR passé en paramètre dans le flux fp
- void createDotBST(const T_bst root, const char *basename)
  - Produit le fichier dot correspondant à l'ABR passé en paramètre
  - Appelle genDot

centralelille

AUDACE • EXIGENCE • RESPECT

1h00

Sujet du fil rouge
2021-2022

Arbres AVL

Adelson Velski et Landis (1962, Russie)

Arbres équilibrés en hauteur

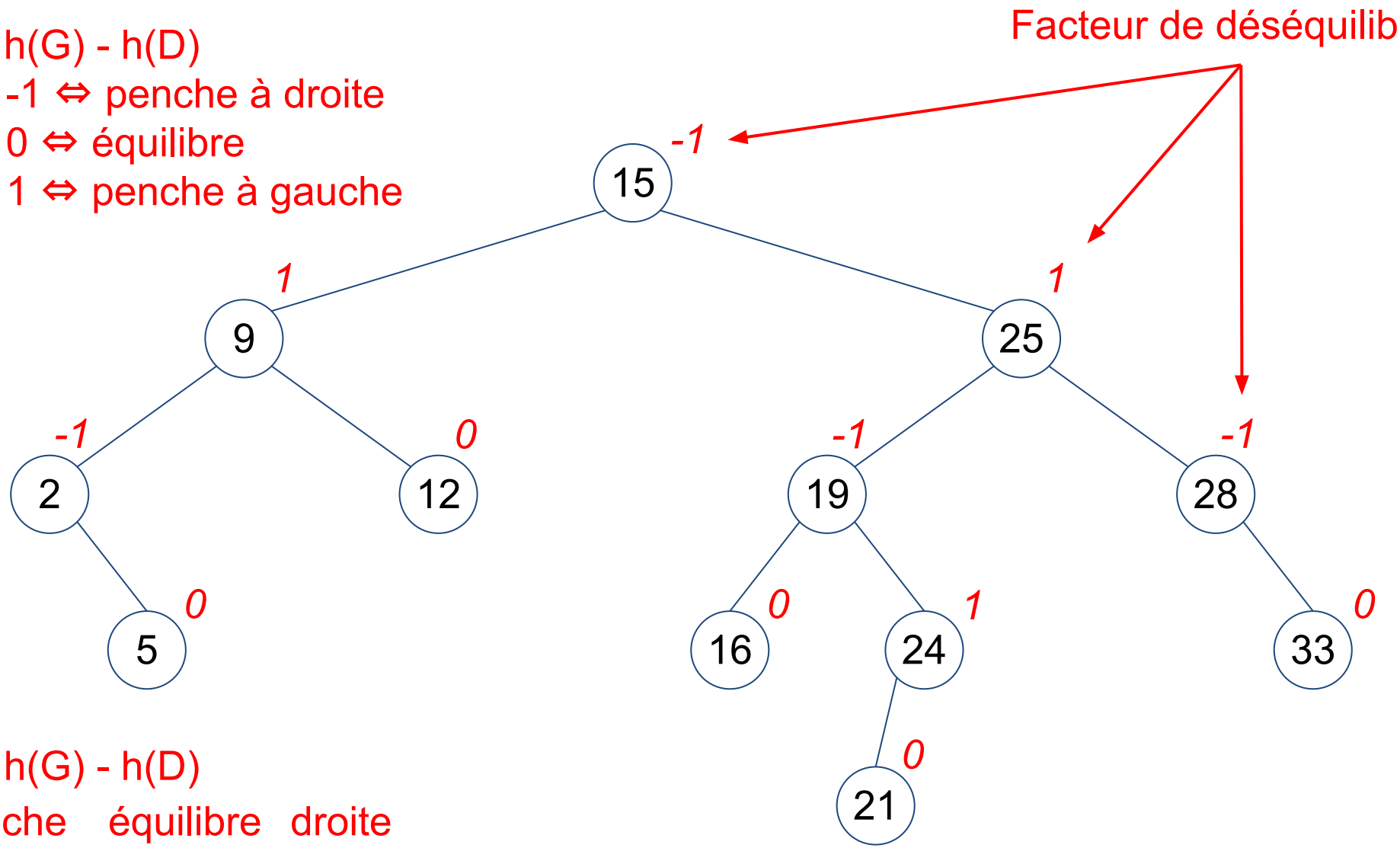
Arbres AVL : définition

- Un arbre binaire AVL vérifie la propriété suivante :
  - La différence de hauteur du sous-arbre gauche et du sous-arbre droit de la racine ne peut excéder 1 en valeur absolue
  - Cette propriété s'applique récursivement à chaque sous-arbre
- On appelle déséquilibre d'un arbre enraciné sur un nœud donné, la différence de hauteur de ses deux-sous arbres, s'ils existent
  - Dans un arbre AVL, pour chaque nœud le facteur de déséquilibre peut valoir uniquement -1, 0 ou 1

Arbres AVL : exemple

Δ = h(G) - h(D)

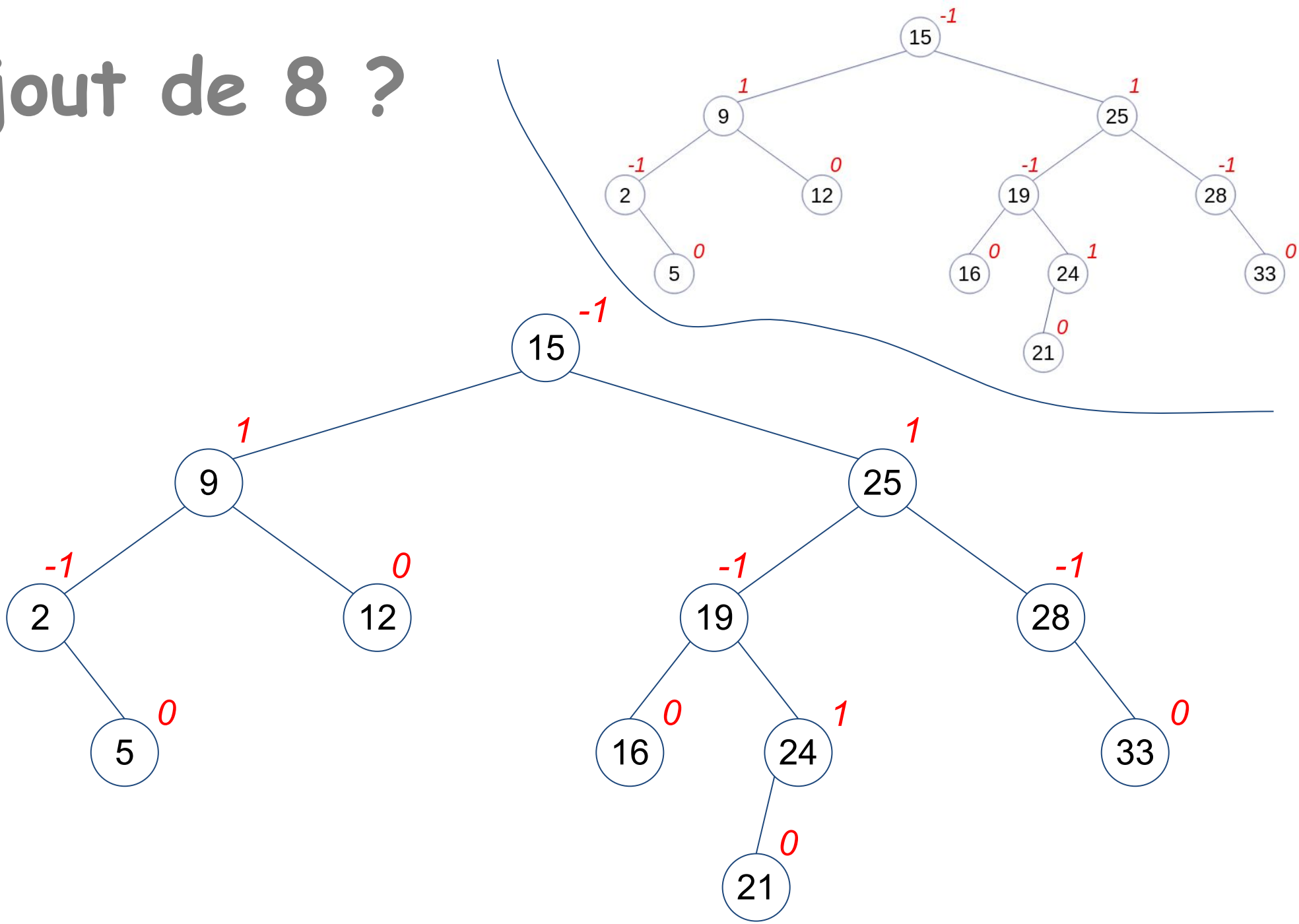
- -1 ⇔ penche à droite
- 0 ⇔ équilibre
- 1 ⇔ penche à gauche



Δ = h(G) - h(D)

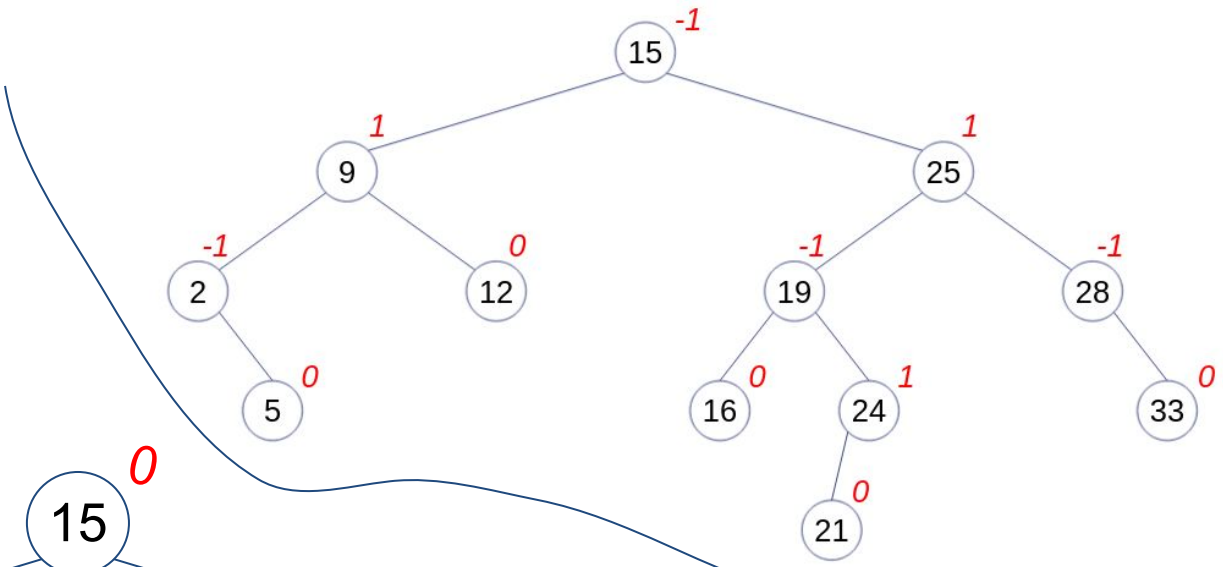
gauche	équilibre	droite
1	0	-1

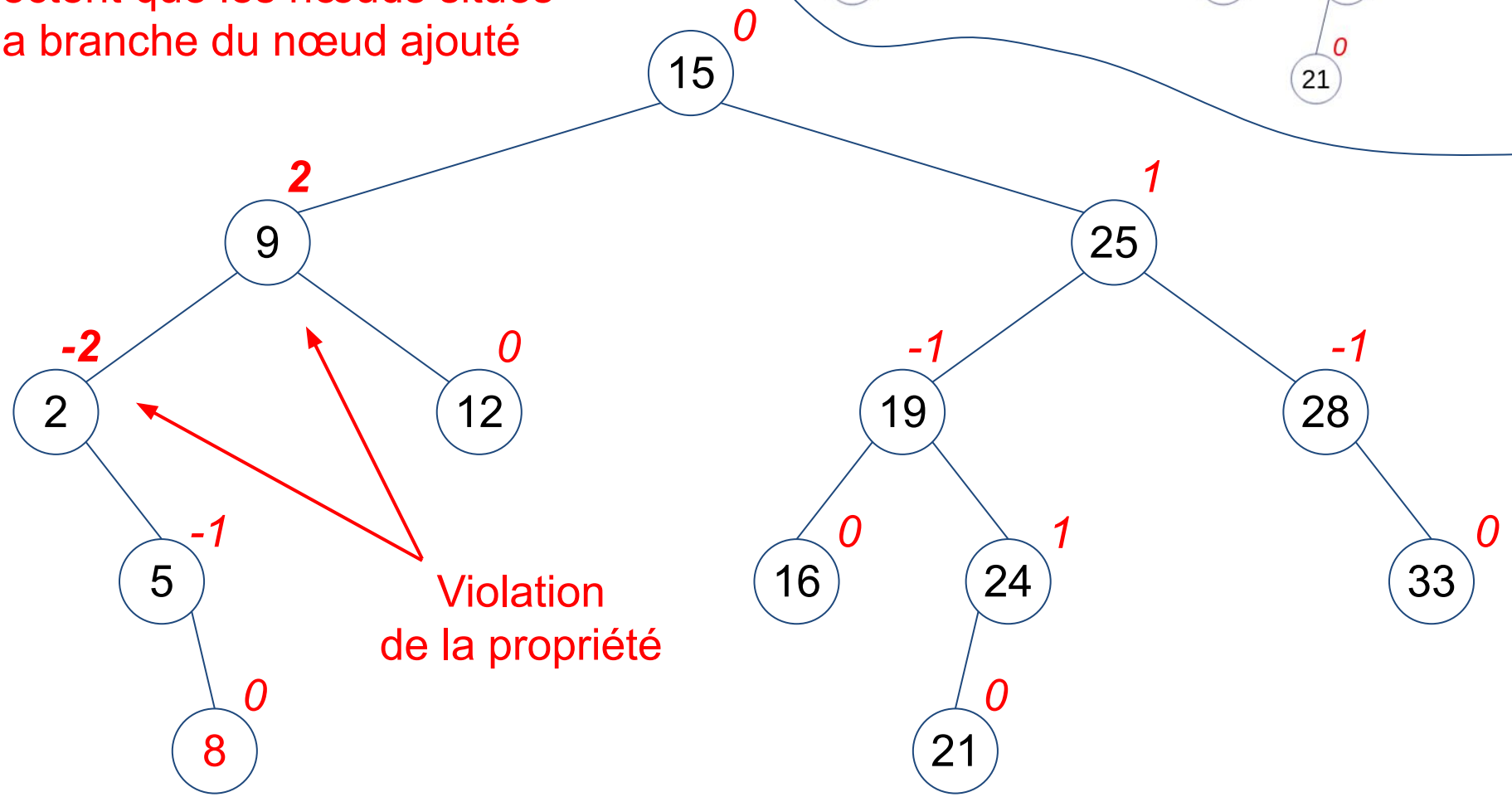
Ajout de 8 ?



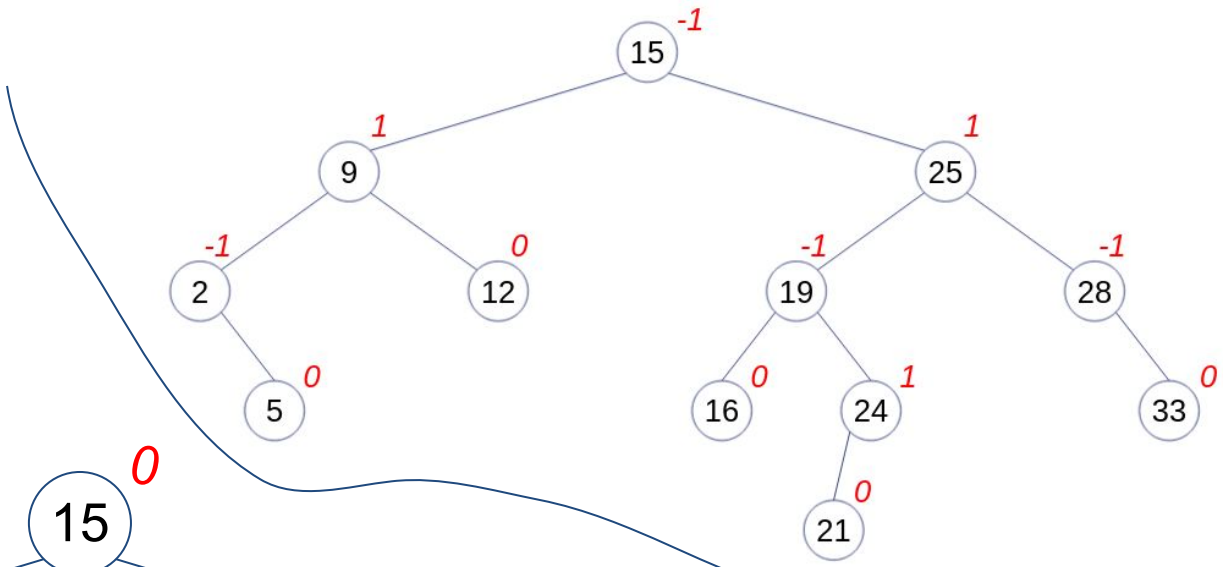
Ajout de 8

NB : Les modifications du facteur de déséquilibre n'affectent que les nœuds situés sur la branche du nœud ajouté

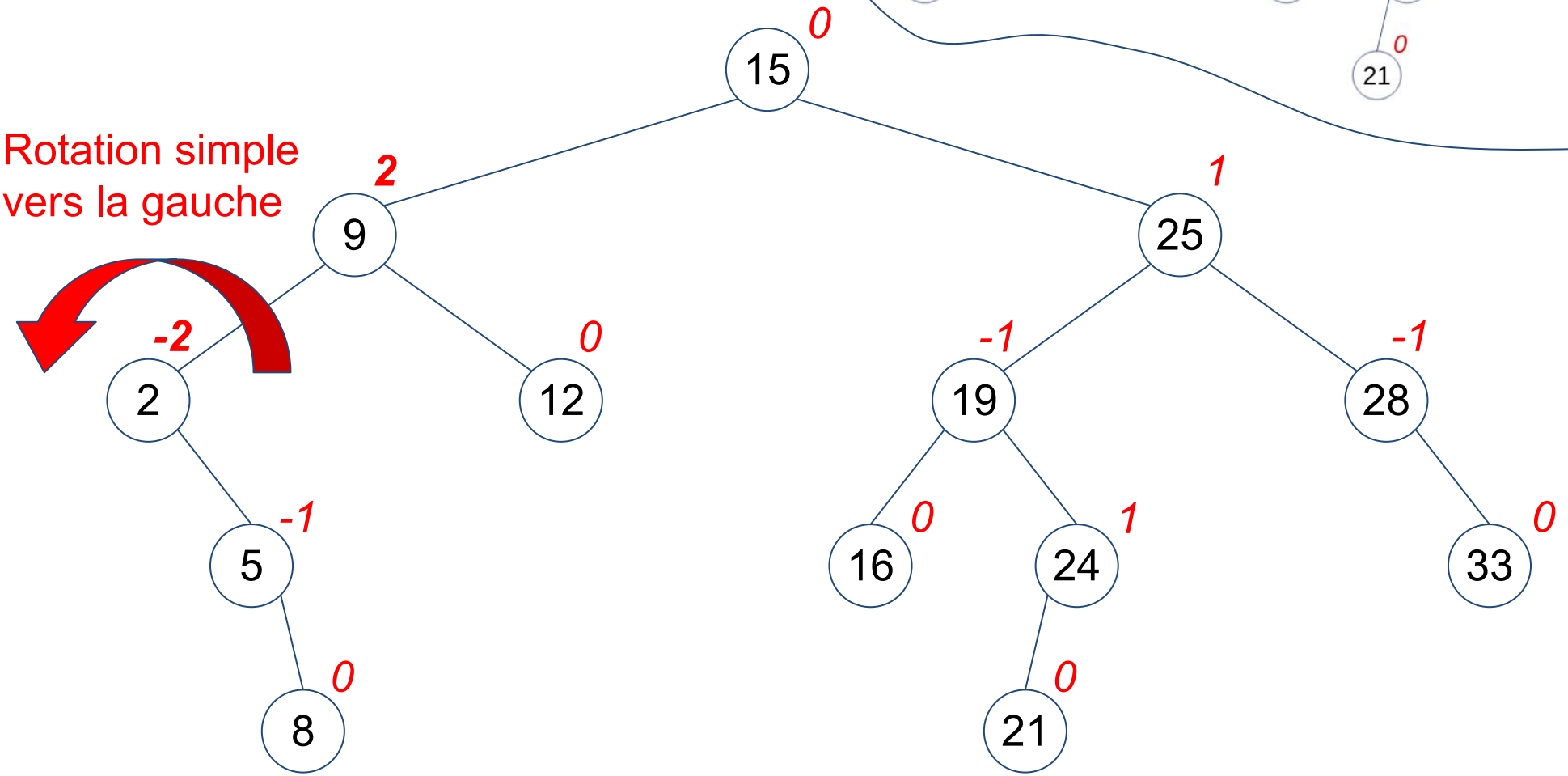




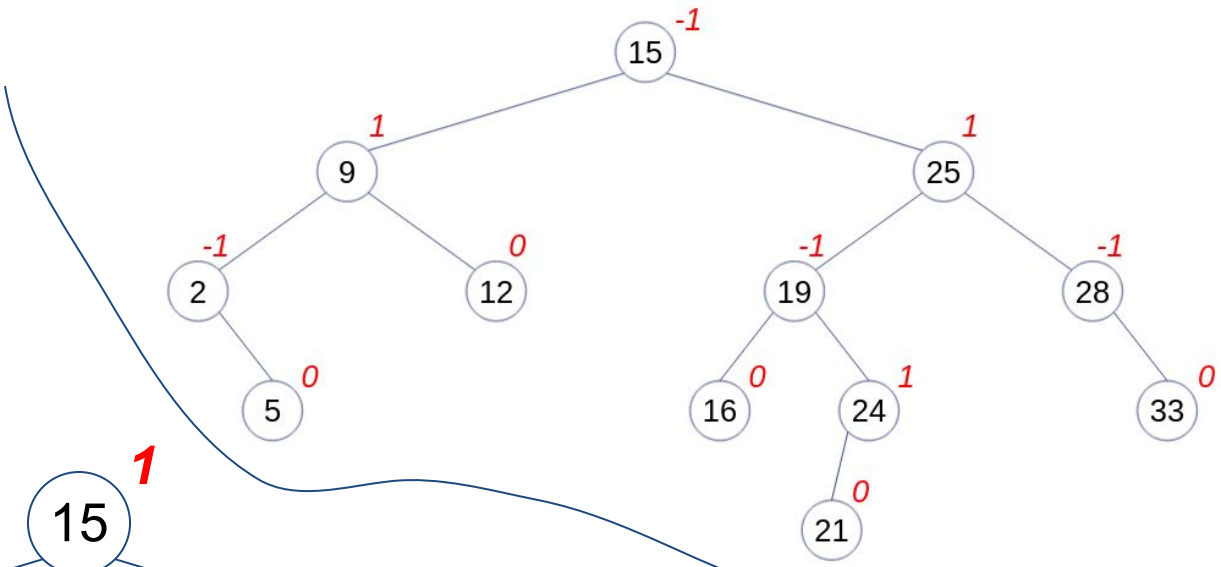
Ajout de 8
Rotation simple



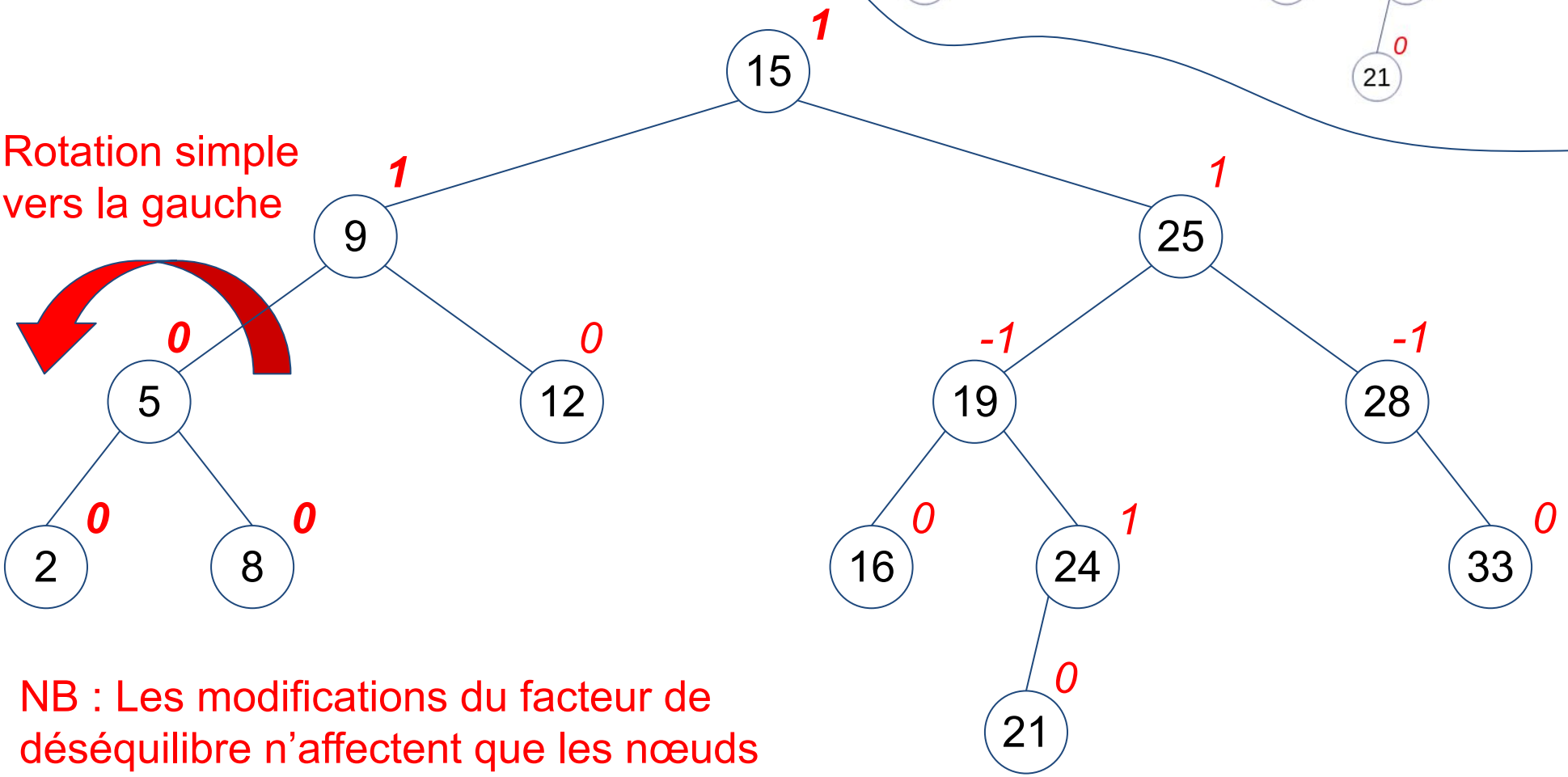
Rotation simple
vers la gauche



Ajout de 8
Rotation simple

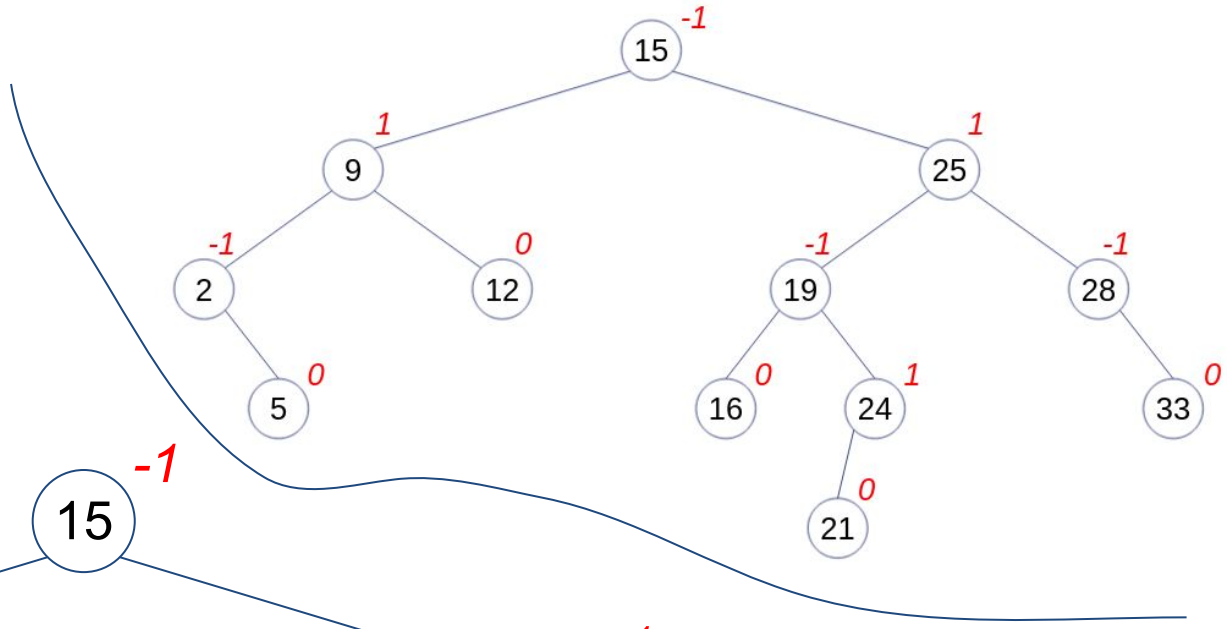


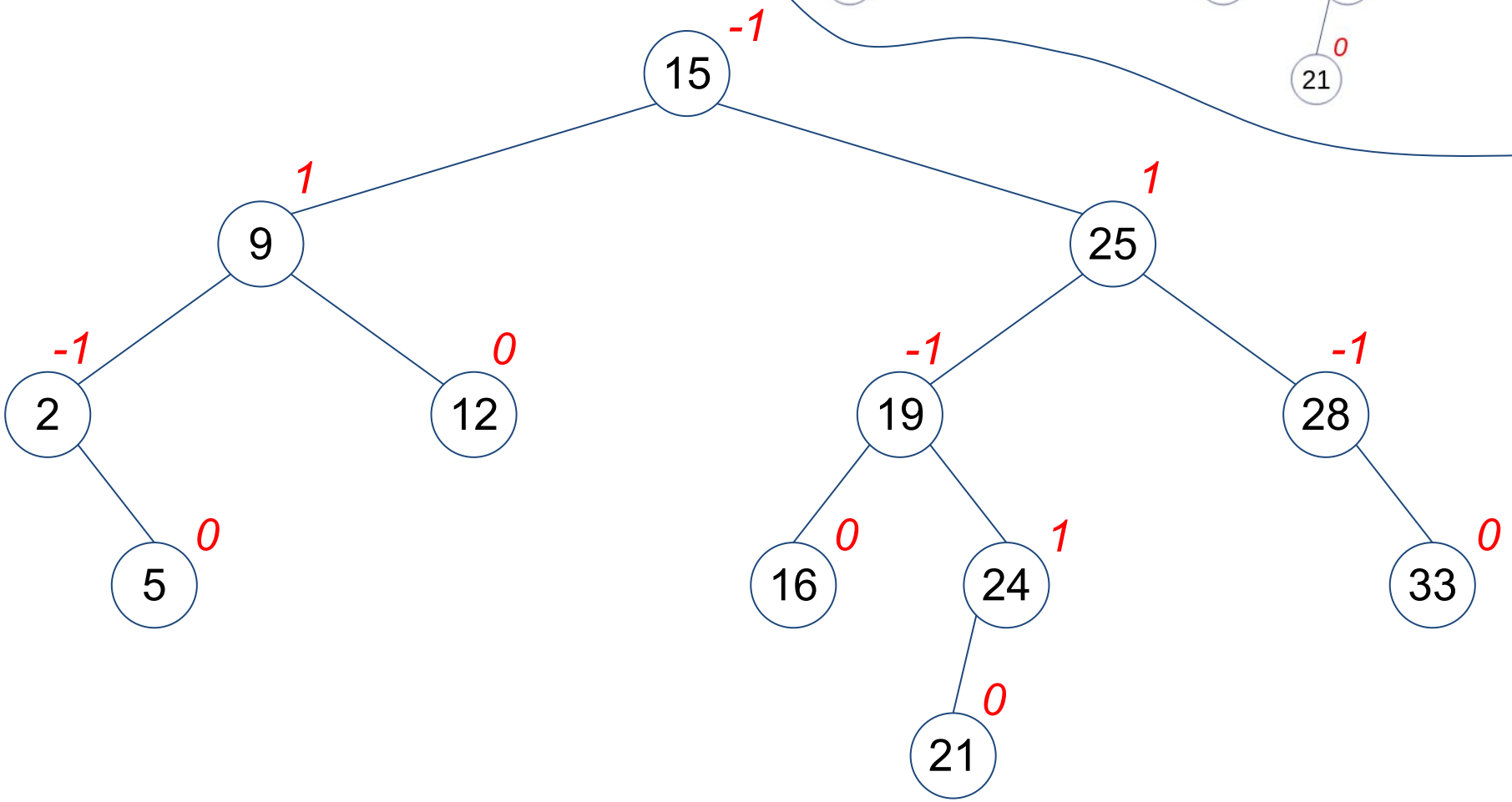
Rotation simple vers la gauche



NB : Les modifications du facteur de déséquilibre n'affectent que les nœuds situés sur la branche du nœud ajouté

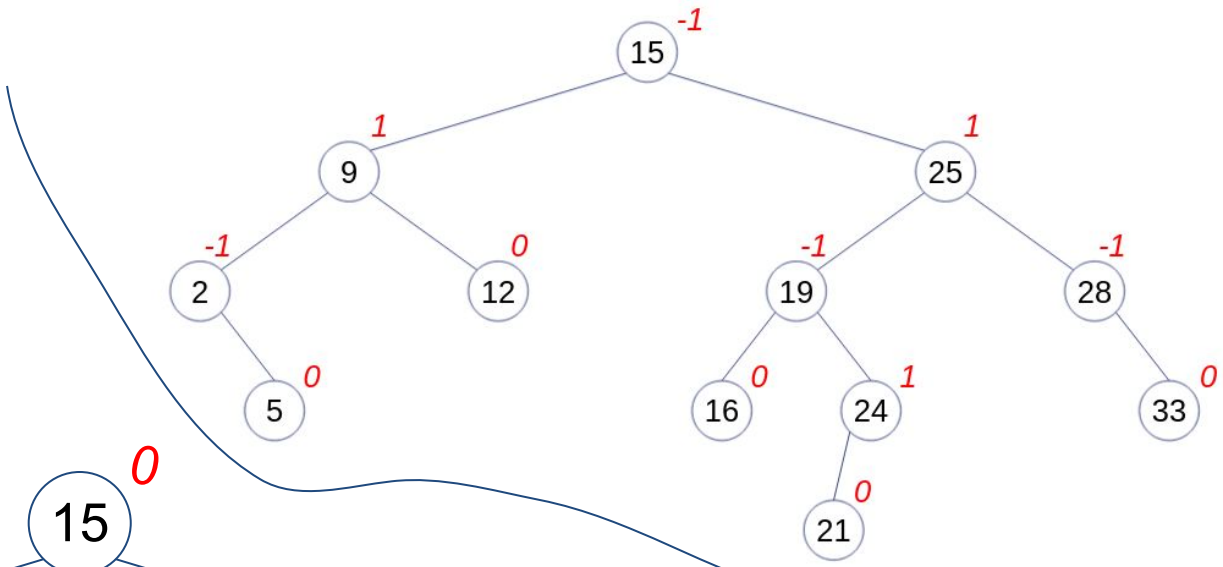
Ajout de 4 ?

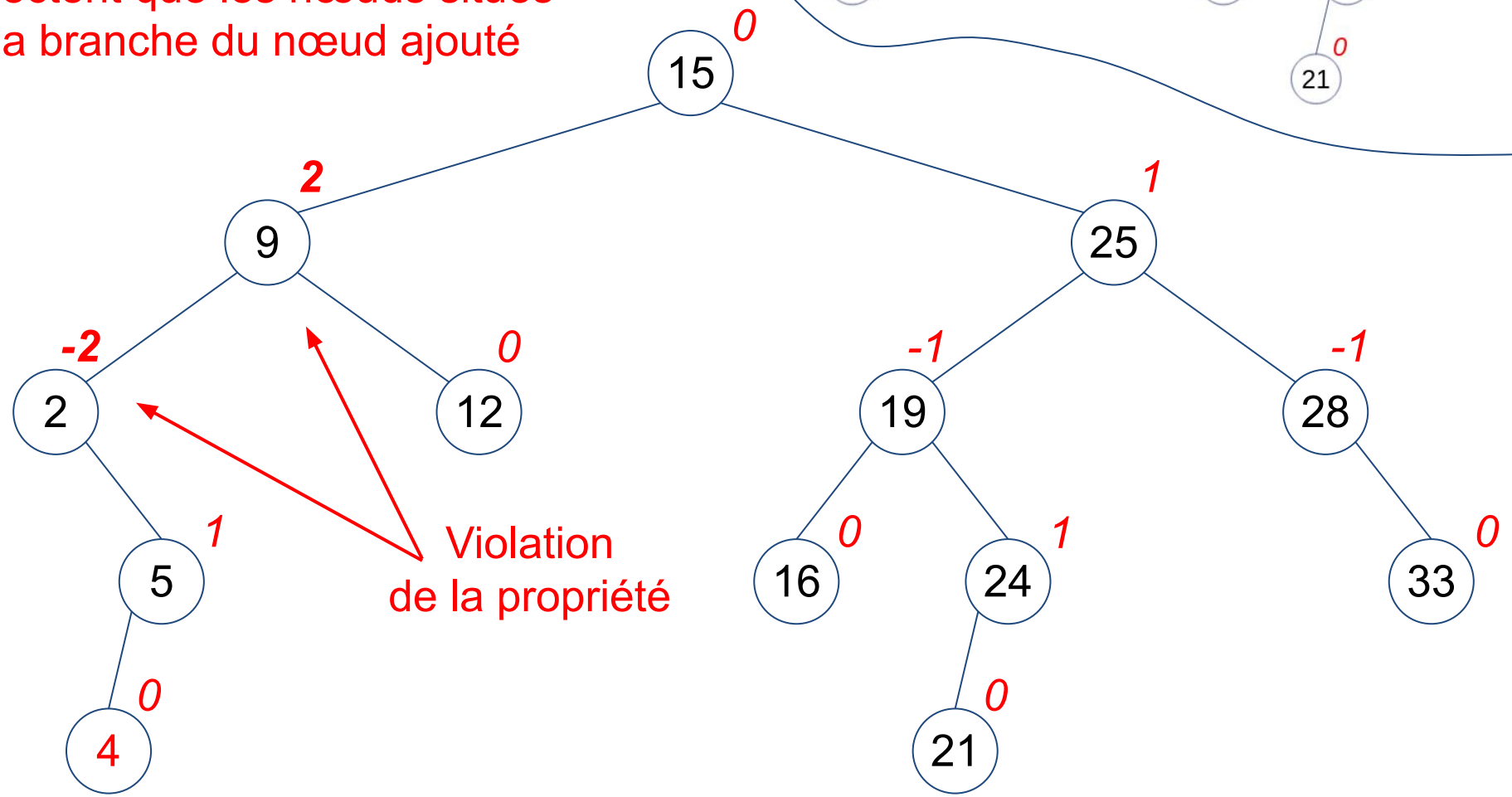

15
-1
9
1
25
1
2
-1
12
0
19
-1
28
-1
5
0
16
0
24
1
33
0
21
0


15
-1
9
1
25
1
2
-1
12
0
19
-1
28
-1
5
0
16
0
24
1
33
0
21
0

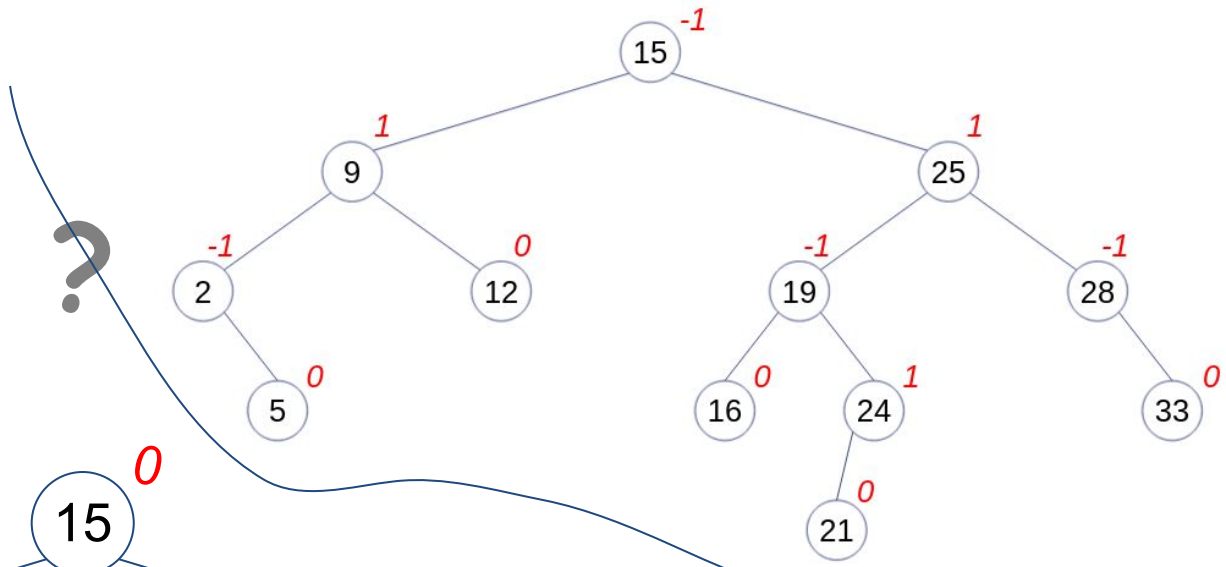
Ajout de 4

NB : Les modifications du facteur de déséquilibre n'affectent que les nœuds situés sur la branche du nœud ajouté

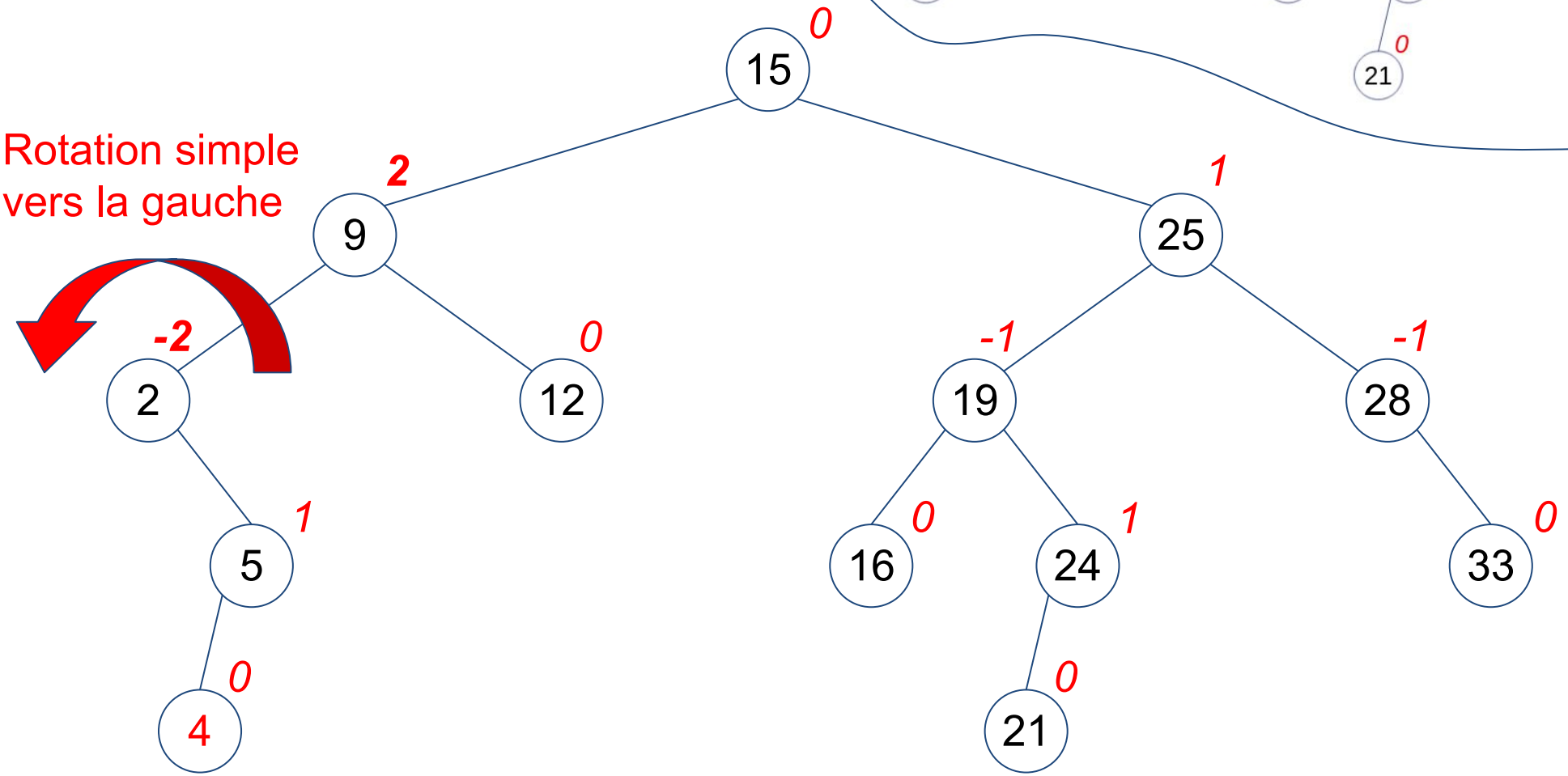




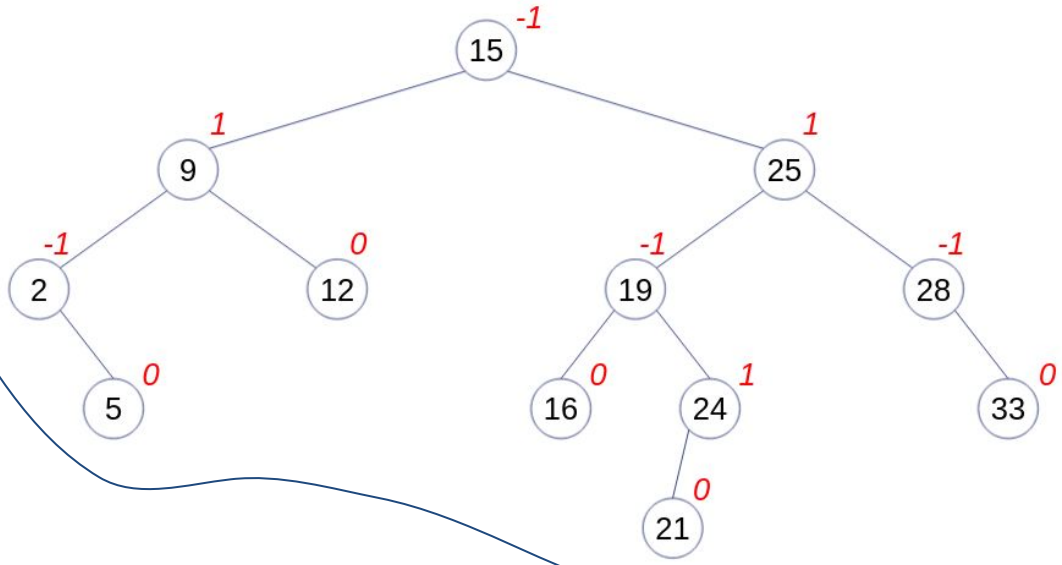
Ajout de 4
Rotation simple ?



Rotation simple
vers la gauche



Ajout de 4
Rotation simple ?

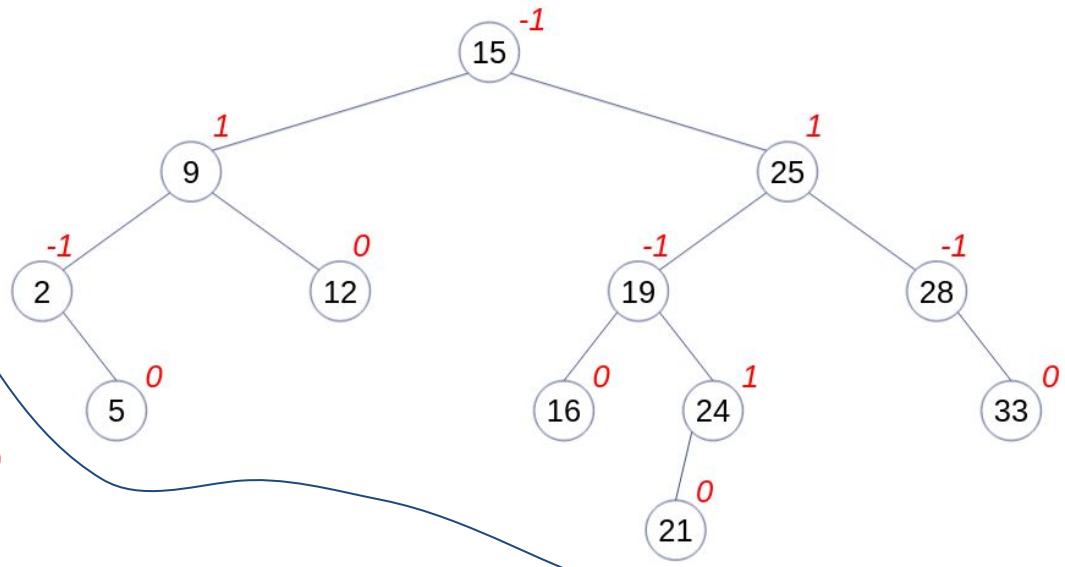


Rotation simple
vers la gauche

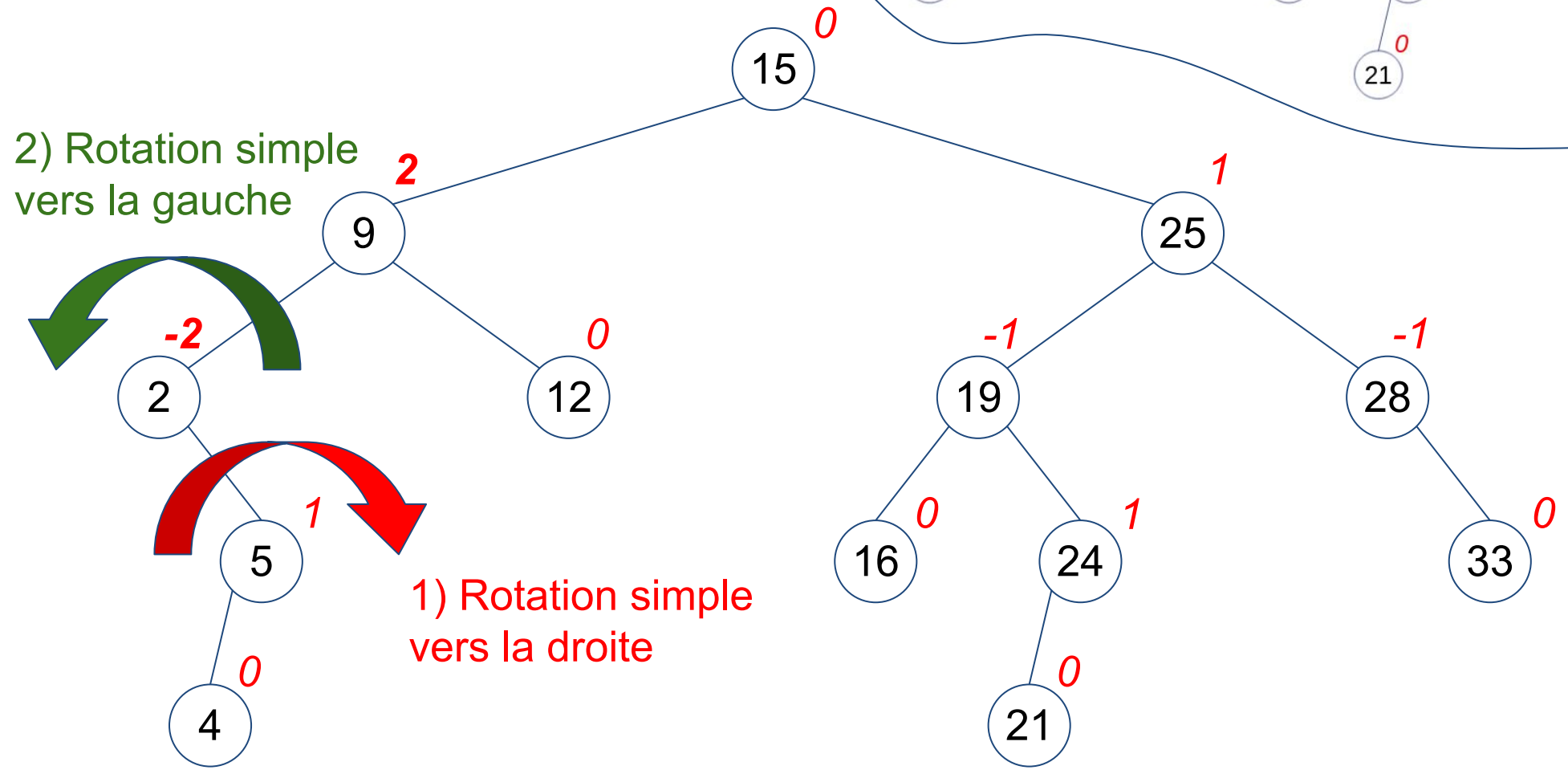
Violation
de la propriété

Rotation simple insuffisante !

Ajout de 4
Rotation **double**
droite-gauche

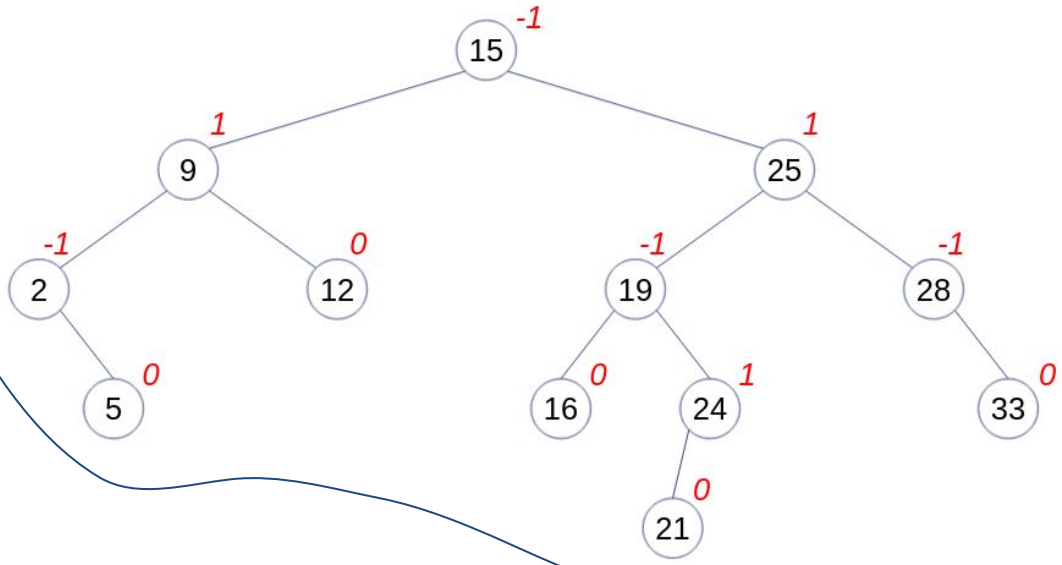


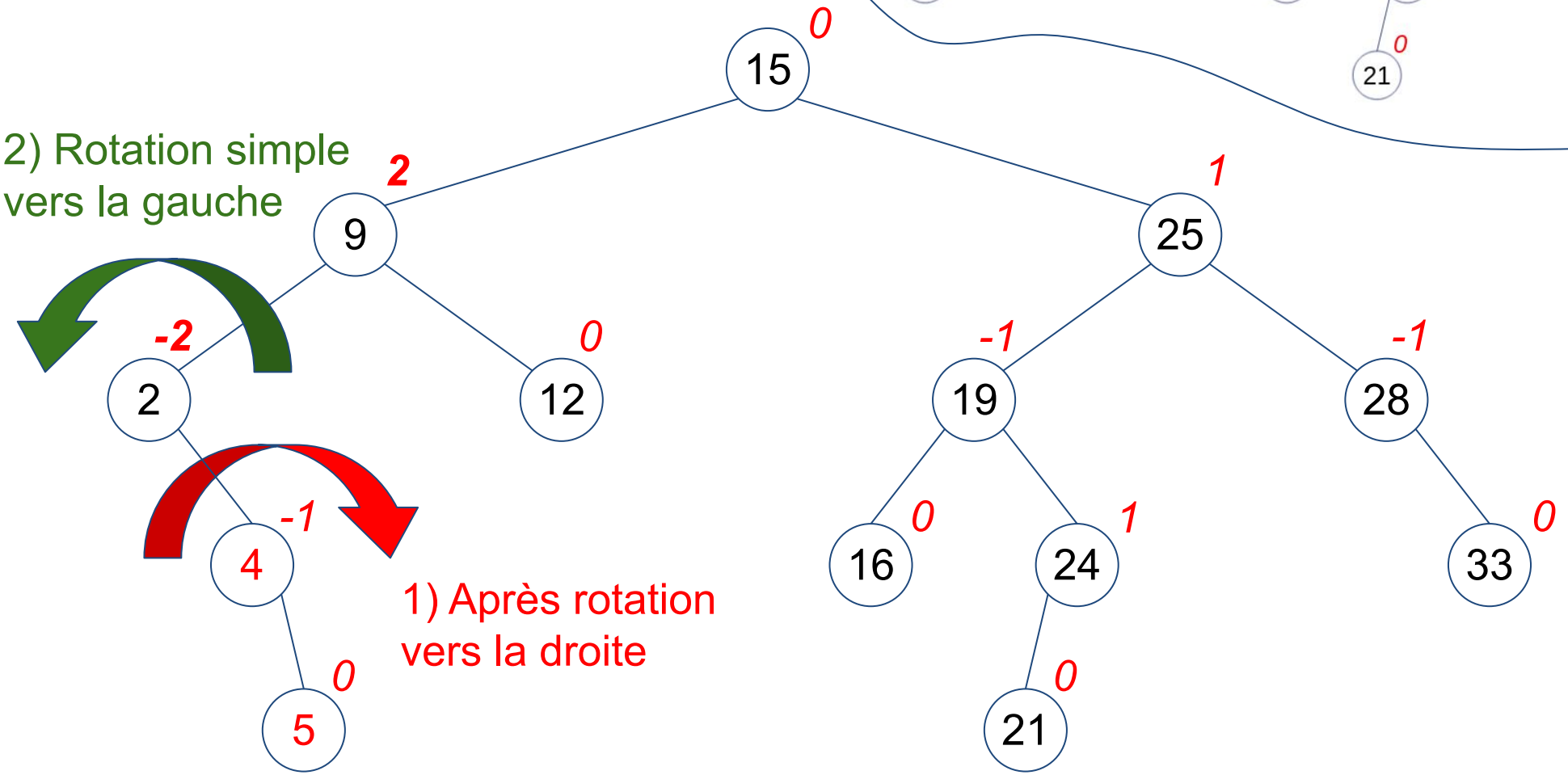
2) Rotation simple vers la gauche



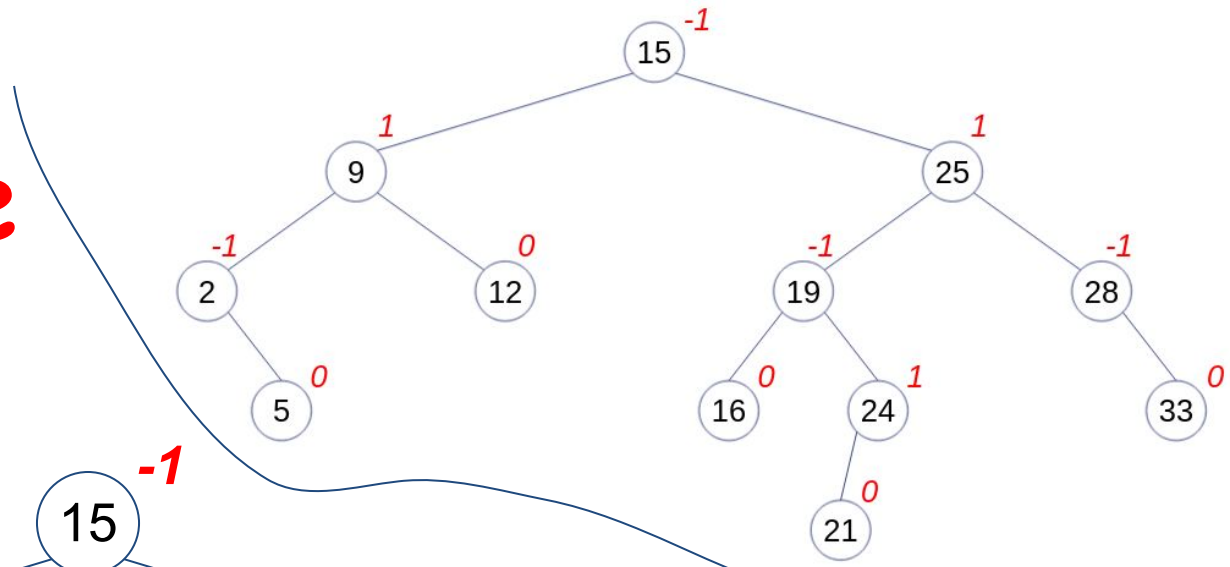
1) Rotation simple vers la droite

Ajout de 4
Rotation **double droite-gauche**

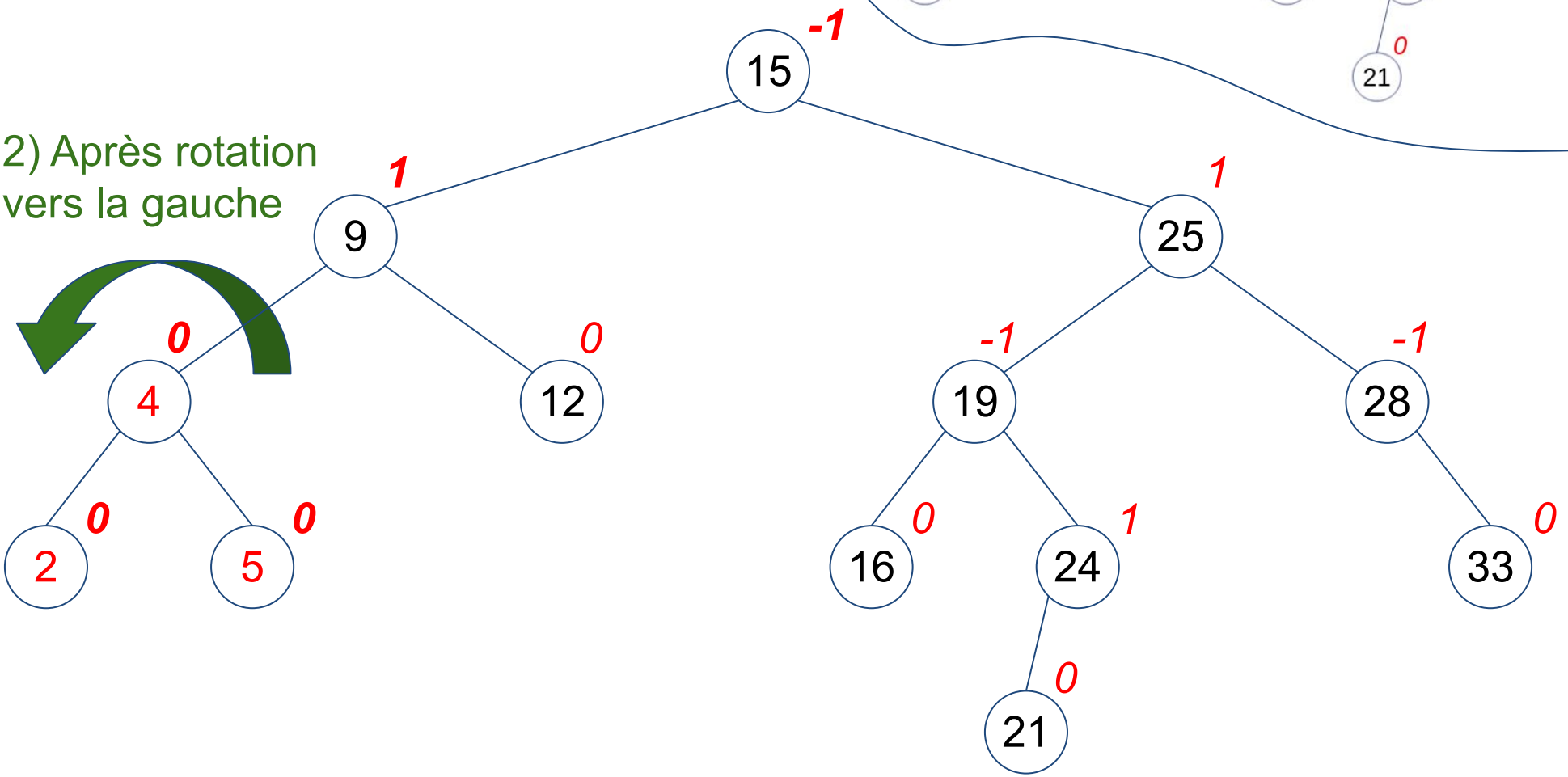




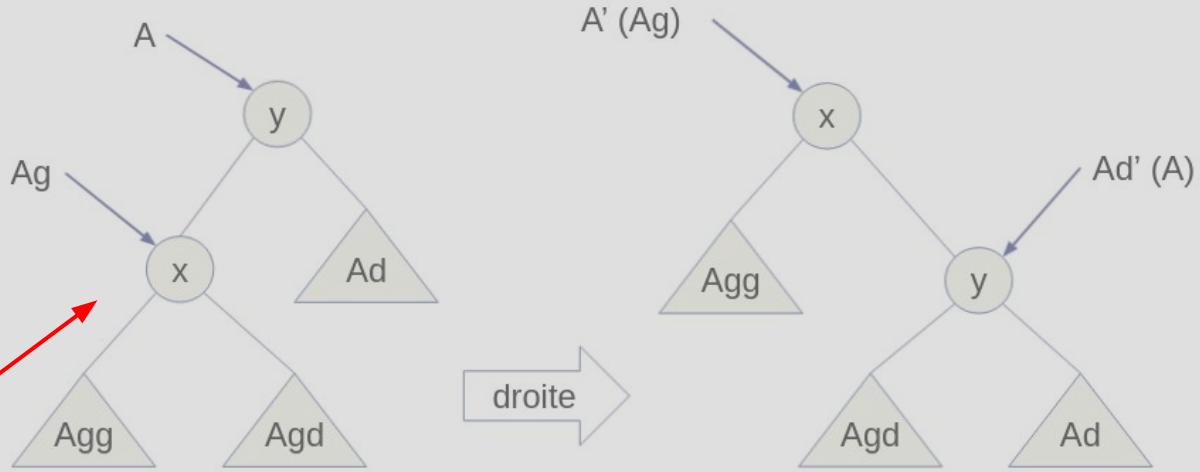
Ajout de 4
Rotation double droite-gauche



2) Après rotation vers la gauche



Formellement :
Rotation droite



- Arbre déséquilibré à gauche
  - h(Ag) > 1 + h(Ad)
- Supposons que h(Agg) > h(Agd) (donc h(Ag) = 1+h(Agg))
  - Ainsi : h(Agg) ≥ 1 + max(h(Agd) , h(Ad))
- On a :
  - h(Ag) = 1 + max(h(Agg), h(Agd)) = 1 + h(Agg)
  - h(A) = 1 + max(h(Ag) , h(Ad)) = 1 + h(Ag) = 2 + h(Agg)
- Après rotation droite :
  - h(Ad') = 1 + max(h(Agd) , h(Ad))
  - h(A') = 1 + max(h(Agg), 1 + max(h(Agd) , h(Ad)) = 1 + h(Agg)

⇒ Le déséquilibre est réduit d'une unité

Arbres AVL : insertion et rééquilibrage

- Après une insertion, seuls les nœuds qui sont sur la branche du point d'insertion à la racine sont susceptibles d'être déséquilibrés
  - Il n'y a pas systématiquement déséquilibre
  - En cas de déséquilibre, deux cas se présentent
- Si insertion dans le sous-arbre droit du fils droit ⇒ simple rotation gauche
  - Cf. ajout de 8
  - Resp. si insertion dans le sous-arbre gauche du fils gauche ⇒ simple rotation droite
- Si insertion dans le sous-arbre gauche du fils droit ⇒ double rotation droite-gauche (droite, puis gauche)
  - Cf. ajout de 4
  - Resp. si insertion dans le sous-arbre droit du fils gauche ⇒ rotation gauche-droite

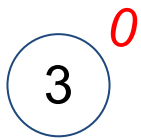
Exercice
Construction d'un AVL

40 min

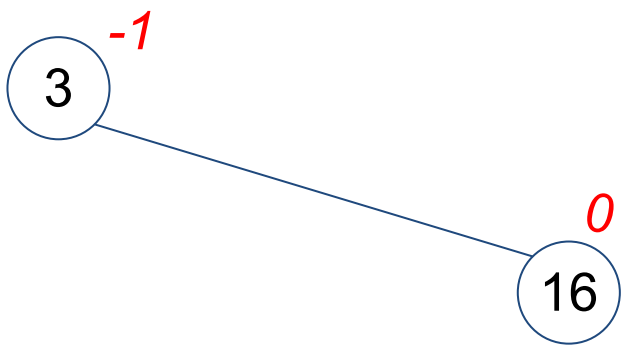
10 min

- En partant d'un arbre binaire vide, construire les arbres AVL obtenus successivement après ajout des éléments suivants : 3, 16, 8, 11, 5, 7, 10
- Après chaque ajout :
  - Vous représenterez le facteur de déséquilibre de chacun des nœuds de l'arbre après accrochage du nouvel élément,
  - Vous préciserez également la transformation éventuellement appliquée et le nœud du sous-arbre sur lequel elle s'applique pour le rééquilibrer,
  - Vous présenterez l'arbre AVL ainsi obtenu

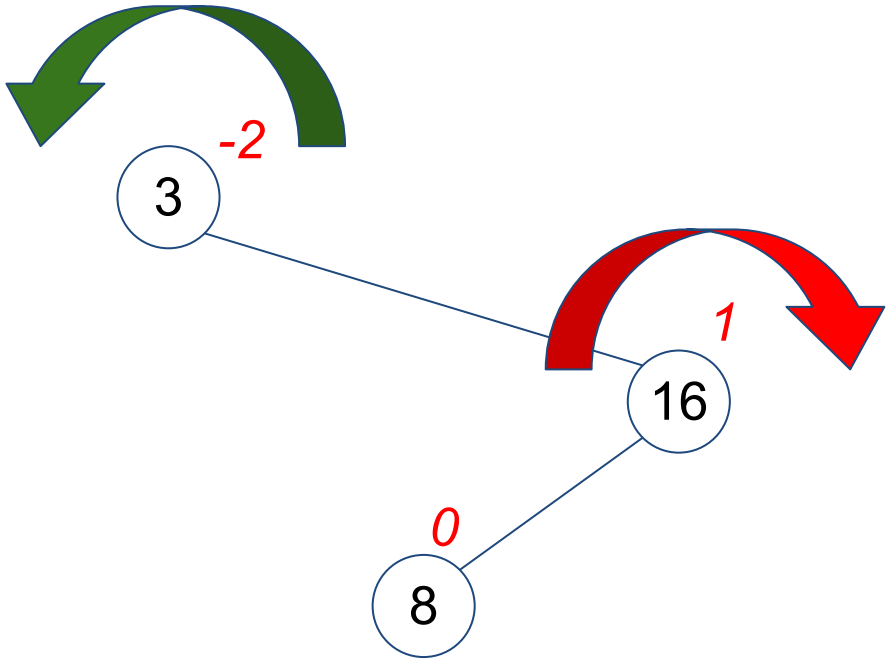
3, 16, 8, 11, 5, 7, 10



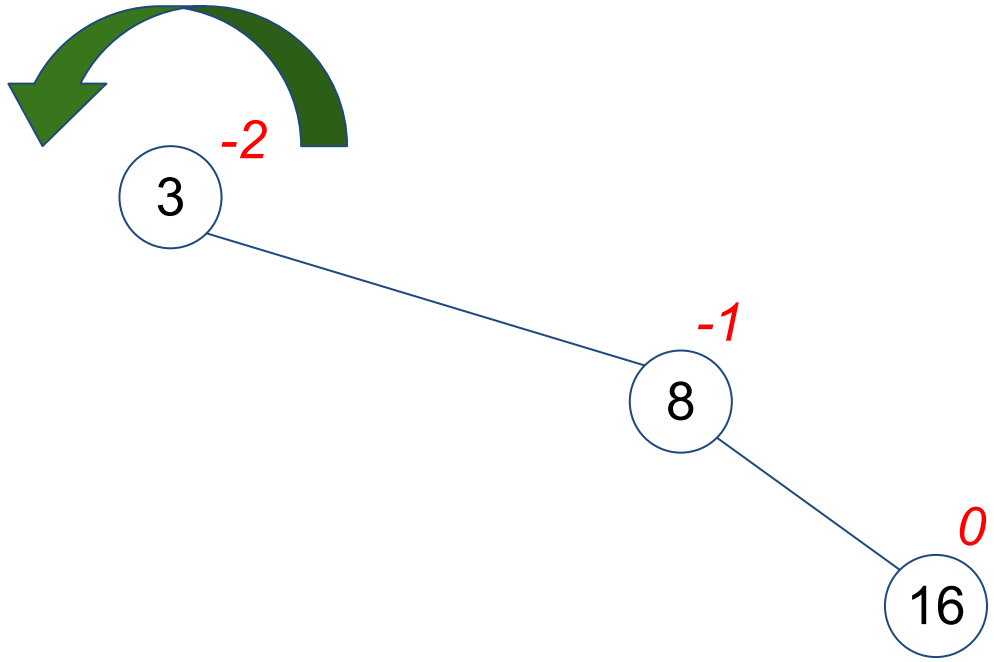
3, 16, 8, 11, 5, 7, 10



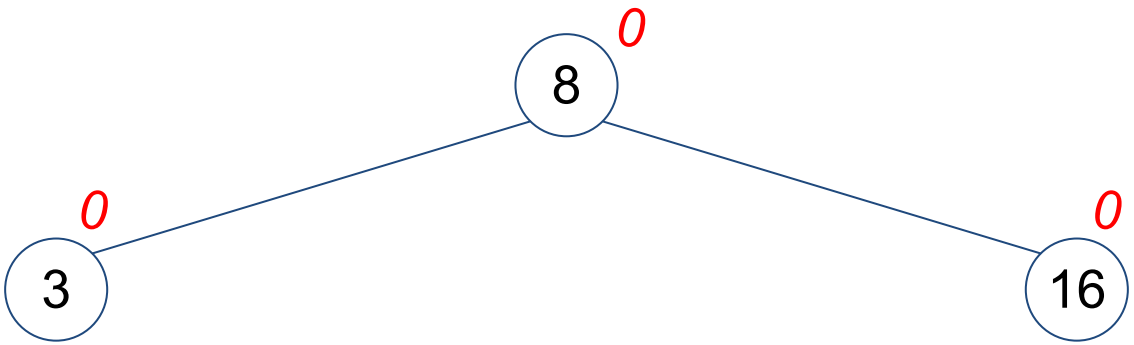
3, 16, 8, 11, 5, 7, 10



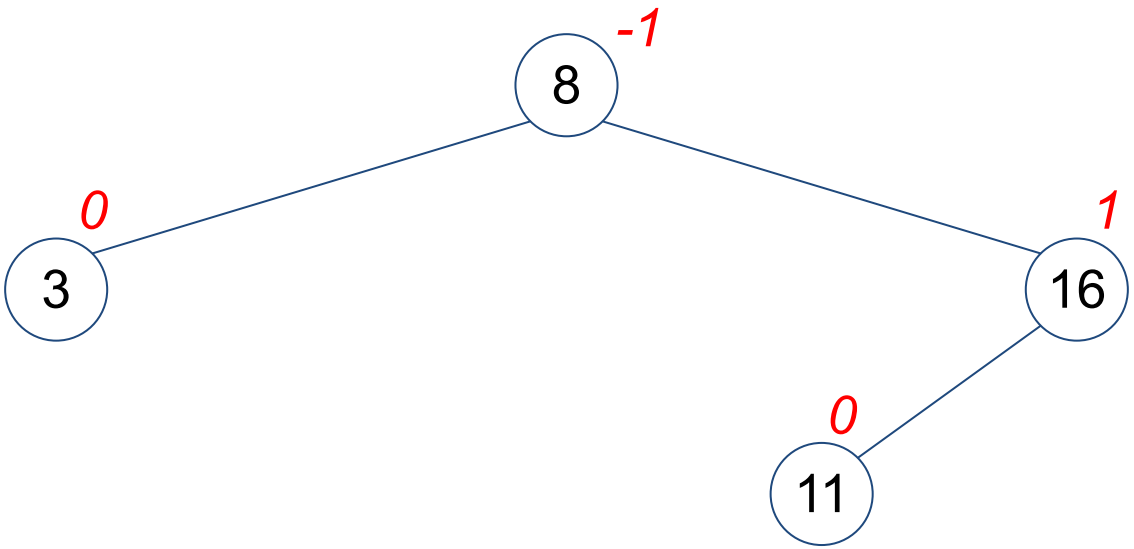
3, 16, 8, 11, 5, 7, 10



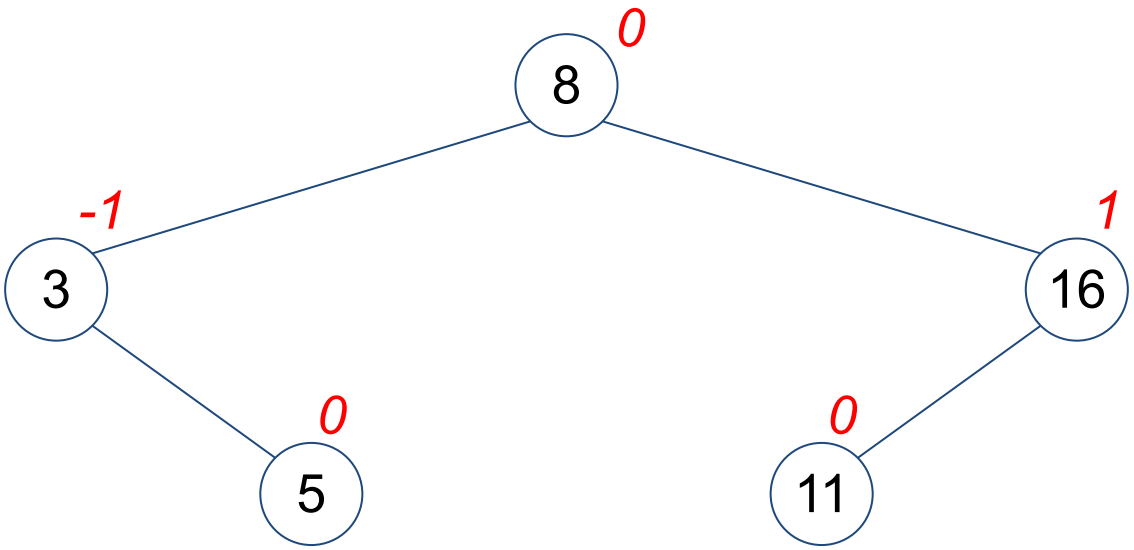
3, 16, 8, 11, 5, 7, 10



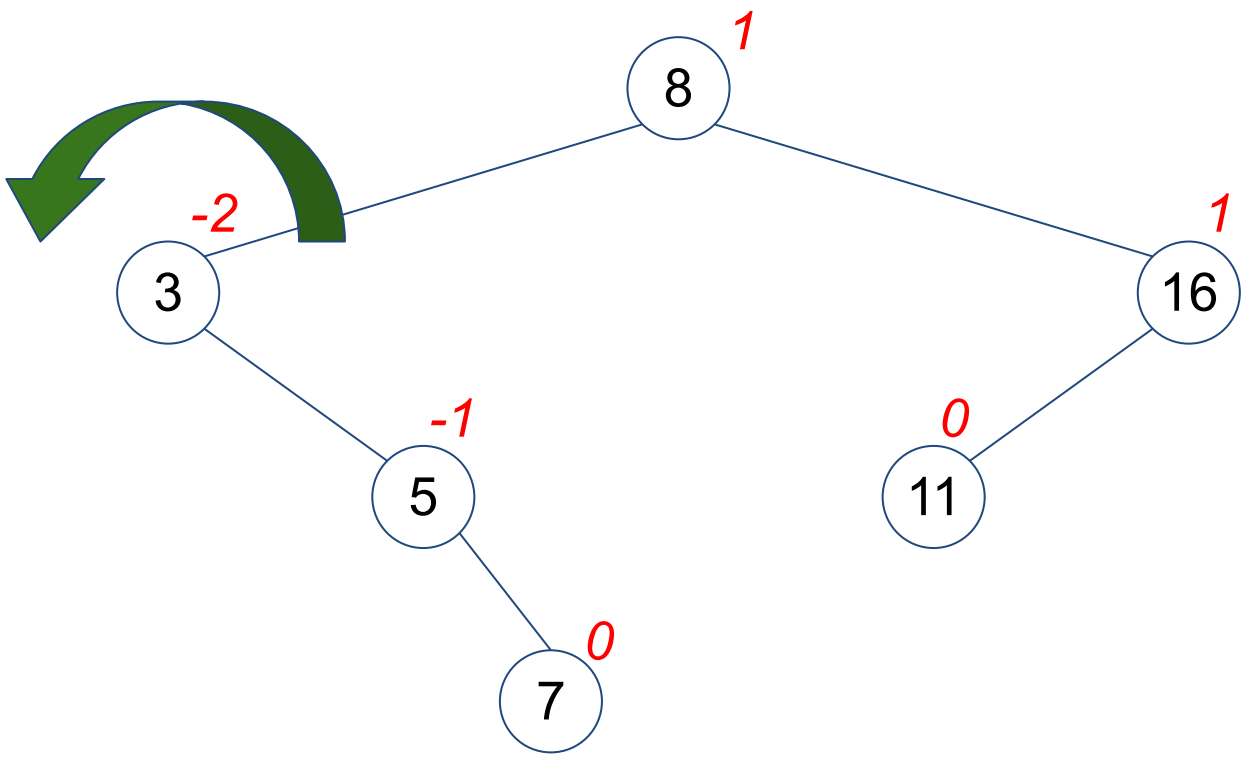
3, 16, 8, 11, 5, 7, 10



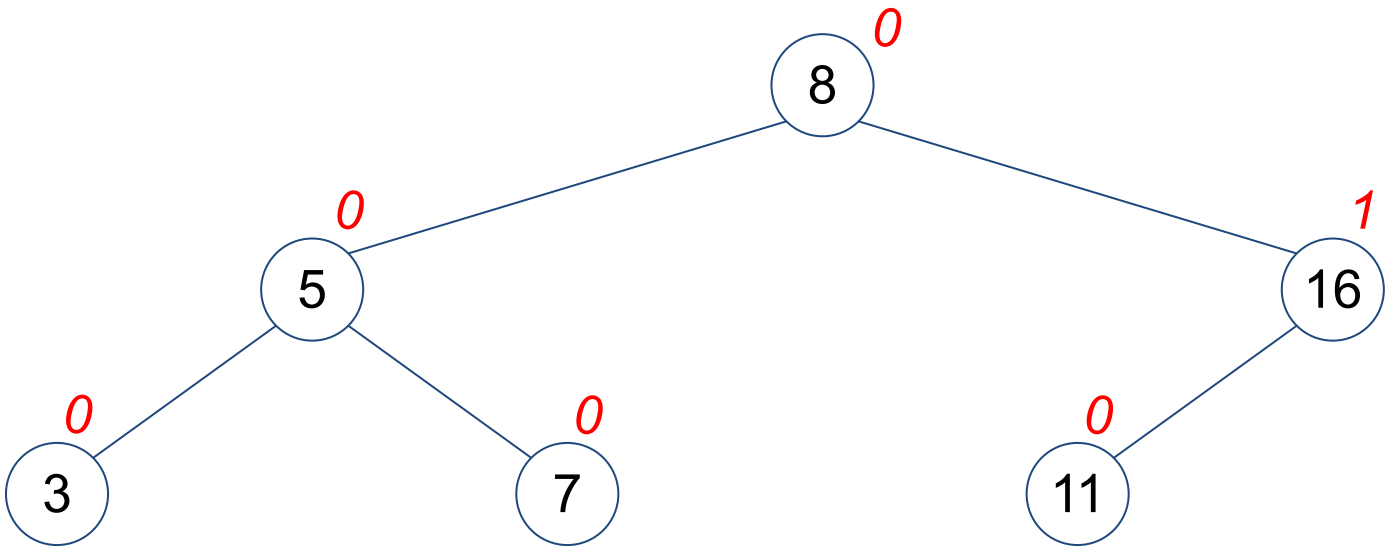
3, 16, 8, 11, 5, 7, 10



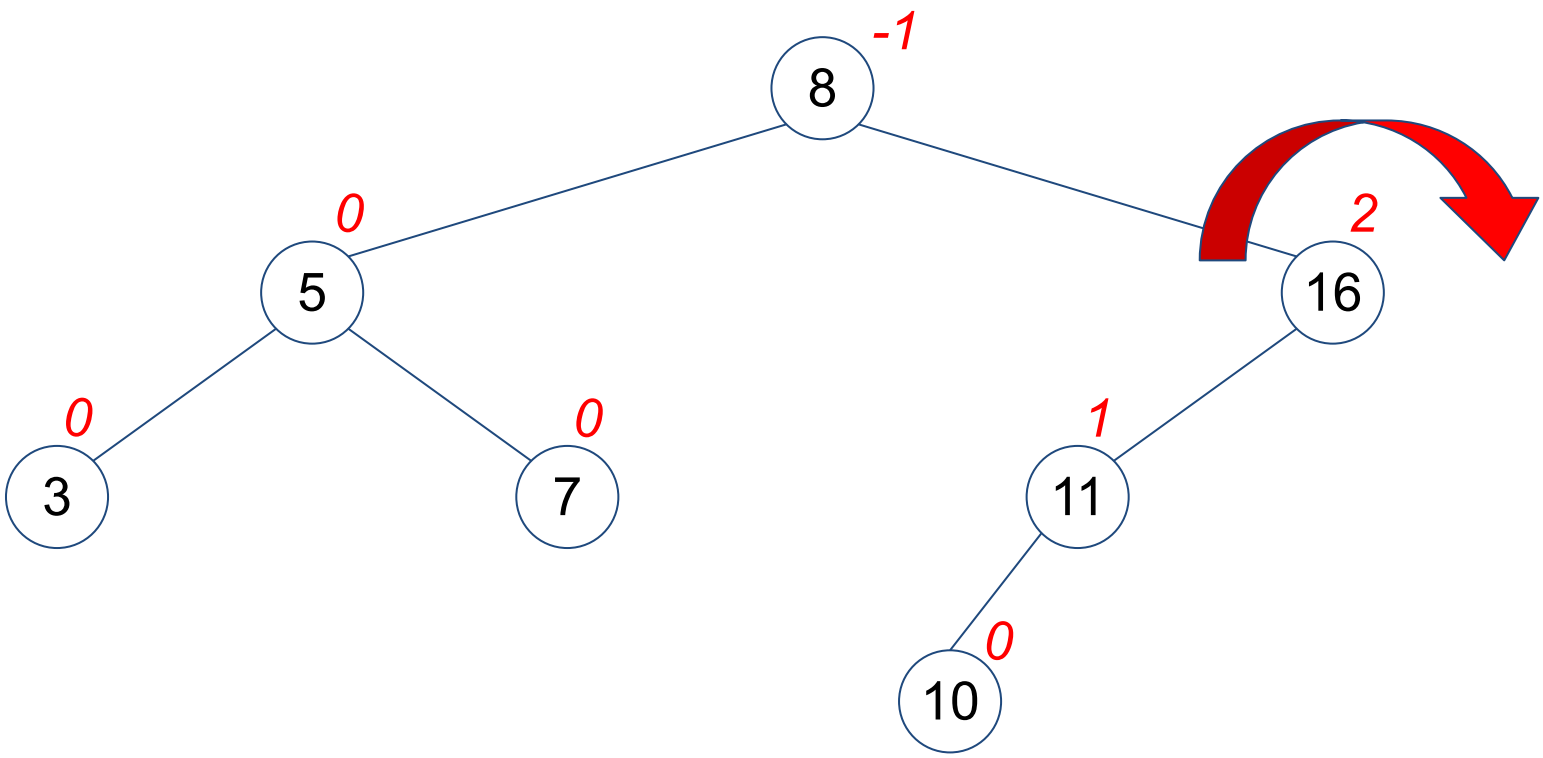
3, 16, 8, 11, 5, 7, 10



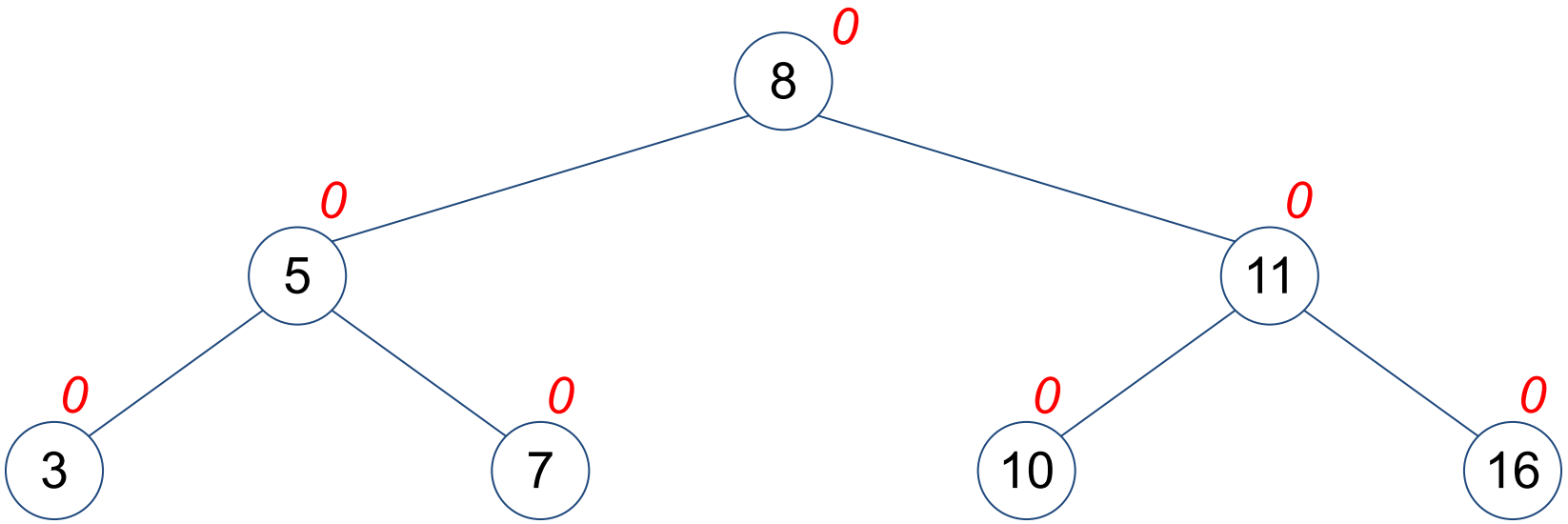
3, 16, 8, 11, 5, 7, 10



3, 16, 8, 11, 5, 7, 10



3, 16, 8, 11, 5, 7, 10



centralelille

AUDACE • EXIGENCE • RESPECT

30 min

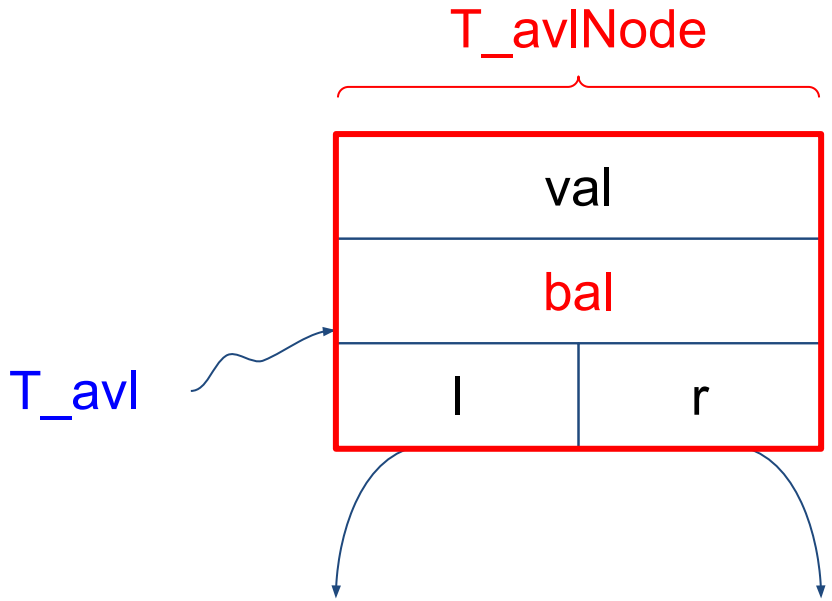
Arbres AVL : Implémentation

Arbres AVL : Implémentation

```
typedef enum { DOUBLE_RIGHT = -2,
               RIGHT = -1,
               BALANCED,
               LEFT,
               DOUBLE_LEFT } T_bal;
```

- Chaque nœud est annoté par un champ bal qui dénote la différence de hauteur entre le fils gauche et le fils droit

```
typedef struct aNode{
   T_elt val;
   T_bal bal;
   struct aNode *l;
   struct aNode *r;
} T_avlNode, *T_avl;
```



Arbres AVL : implémentation

- Développement d'une fonction balanceAVL qui rééquilibre le sous-arbre passé en paramètre et renvoie sa nouvelle racine
  - Cette fonction utilise les rotations élémentaires : rotateLeftAVL, rotateRightAVL
- Développement d'une fonction insertAVL qui insère un élément en maintenant la propriété AVL
  - Cette fonction renvoie 0 ou 1 selon que la hauteur de l'arbre est maintenue ou augmentée d'une unité ⇒ il faut lui passer l'adresse de la variable qui sert à dénoter la racine de l'arbre à utiliser
  - int insertAVL (T_avlNode ** root, T_elt e)
  - Pour insérer : root = insertBST(root,e) devient deltaH = insertAVL(&root,e)

Fonctions à développer

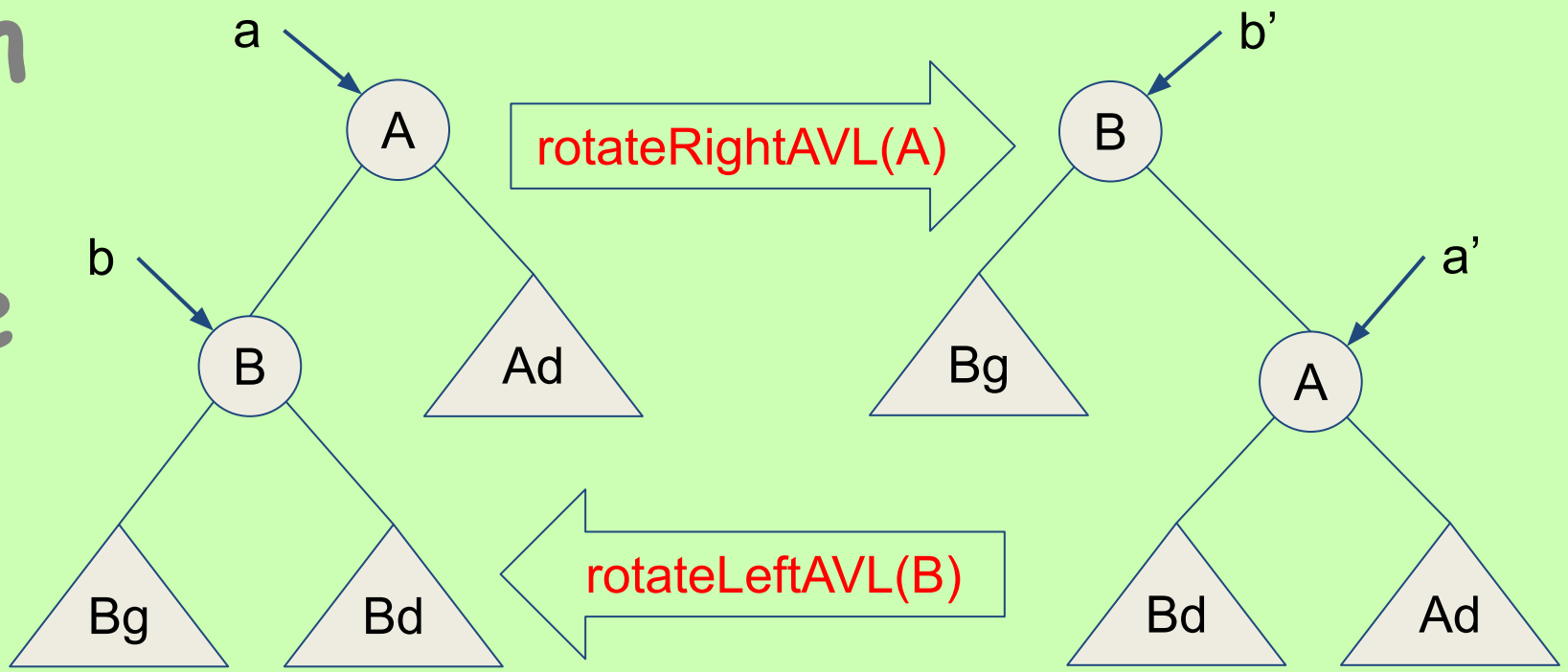
- static T_avl newNodeAVL(T_elt e);
- static T_avlNode * rotateLeftAVL (T_avlNode * A);
- static T_avlNode * rotateRightAVL (T_avlNode * B);
  - NB : rotation double = enchaînement de deux rotations simples...
- static T_avlNode * balanceAVL(T_avlNode * A);
- int insertAVL (T_avlNode ** root, T_elt e);

Fonctions à développer

Sujet du fil rouge
2021-2022

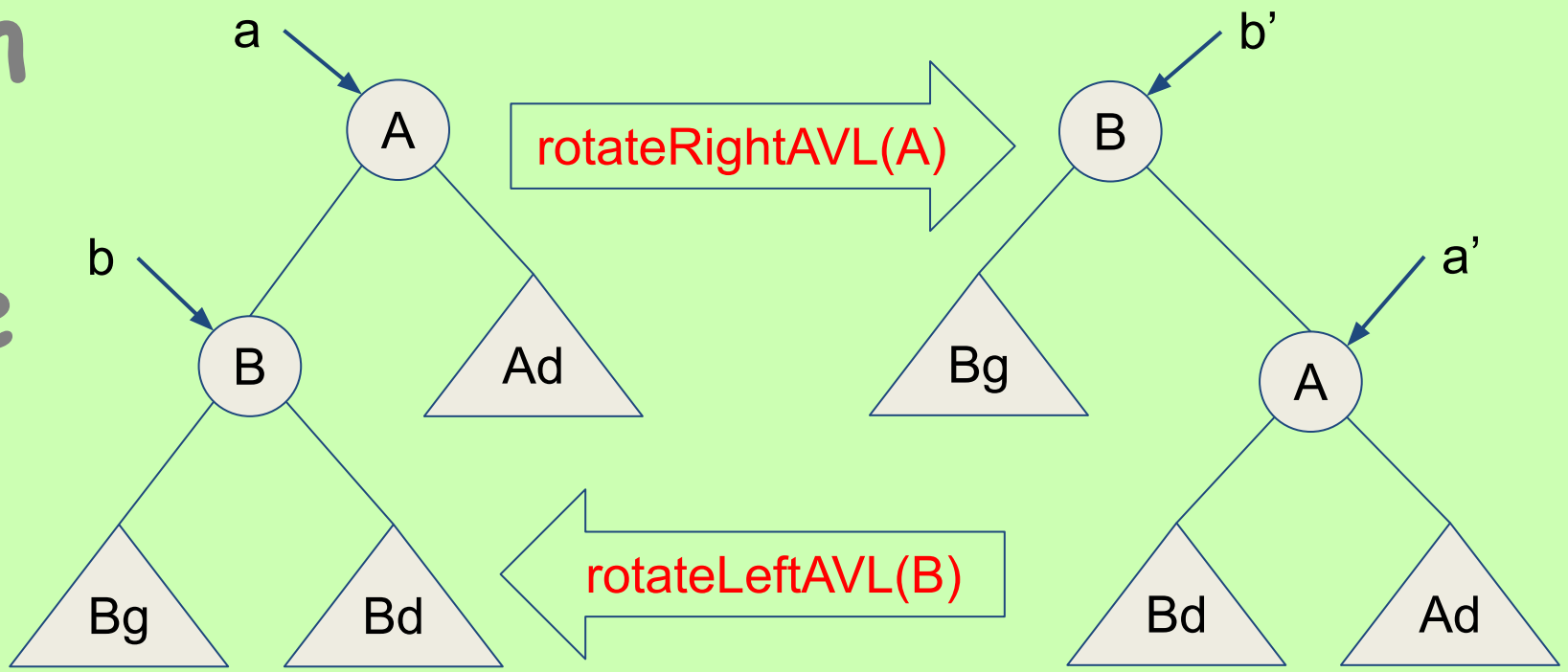
- void printAVL(T_avl root, int indent);
- int heightAVL(T_avl);
- int nbNodesAVL(T_avl);
- T_avlNode * searchAVL_rec(T_avl root, T_elt e);
- T_avlNode * searchAVL_it(T_avl root, T_elt e);

Rotation simple à droite



- Donner le pseudo-code permettant la rotation droite de l'arbre enraciné en A
- Déterminer l'expression du facteur de déséquilibre a' du nœud A après une rotation droite, en fonction des facteurs de déséquilibre a et b des nœuds A et B avant la rotation
- Déterminer l'expression du facteur de déséquilibre b' du nœud B après une rotation droite, en fonction du facteur de déséquilibre b du nœud B avant la rotation et, au choix, du facteur de déséquilibre du nœud A avant rotation, ou a' son facteur de déséquilibre après la rotation droite

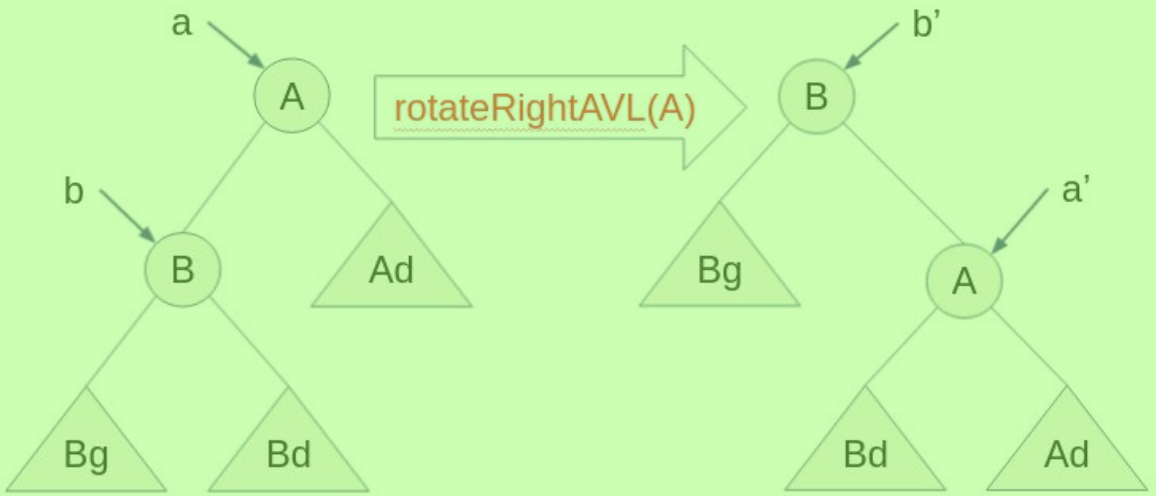
Rotation simple à droite




```
rotateRightAVL(A) {
    B = A->l
    A->l = B->r
    B->r = A
    return B
}
```

- Les facteurs de déséquilibre de Bg, Bd et Ad n'ont pas changé
- Il faut mettre à jour :
  - A->bal = a'
  - B->bal = b'

Rotation simple à droite



- a = h(B) - h(Ad)
- b = h(Bg) - h(Bd)
- h(B) = 1 + max(h(Bg), h(Bd))

- a' = h(Bd) - h(Ad)

= h(Bd) + a - h(B)

= a + h(Bd) - 1 - max(h(Bg), h(Bd))

= a - 1 + min(-h(Bg), -h(Bd)) + h(Bd)

= a - 1 + min (h(Bd)-h(Bg), 0)

= a - 1 + min (-b, 0)

- a' = a - 1 - max (0,b)

- b' = h(Bg) - h(A)
- h(A) = 1 + max(h(Bd), h(Ad))
- b' = h(Bg) - 1 - max (h(Bd), h(Ad))

= -1 + min (-h(Bd), -h(Ad)) + h(Bg)

= -1 + min(h(Bg)-h(Bd), h(Bg)-h(Ad))

= -1 + min(b,h(Bg) -h(Bd)+h(Bd) - h(Ad))

= -1 + min(b, b+a')

- b' = b -1 + min(0,a')

Equilibrage

static T_avlNode * balanceAVL(T_avlNode * A)

- Hypothèse : les facteurs de déséquilibre de tous les nœuds de A sont corrects : Ils ont été mis à jour lors des rotations…

```
Si (A penche à gauche)
    Si (A->l penche à droite)
        A->l = rotateLeftAVL(A->l)
        Renvoyer rotateRightAVL(A)
    Sinon
        Renvoyer rotateRightAVL(A)
Si (A penche à droite)
    … (à terminer)
```

Insertion et Rééquilibrage

int insertAVL (T_avlNode ** pA, T_elt e)

- *Insère un élément en maintenant la propriété AVL*
- *Renvoie 0 ou 1 selon que la hauteur de l'arbre est maintenue ou augmentée d'une unité*
- *On lui passe l'adresse de la variable contenant l'adresse de A*

Si (e est inférieur ou égal à l'élément dans la racine)

    deltaH = insertAVL(...) // insertion dans sous-arbre gauche

    (*pA)->bal += +deltaH // mise à jour du facteur de déséquilibre

Sinon

    deltaH = insertAVL(...) // insertion dans sous-arbre droit

    (*pA)->bal += -deltaH // mise à jour du facteur de déséquilibre

Si (deltaH == 0) Renvoyer 0 // pas de modification de hauteur : on renvoie 0

Sinon // le sous-arbre renvoyé par l'appel récursif a grandi

    *pA = balanceAVL(*pA) // on rééquilibre

Si (le facteur de déséquilibre de A n'est pas redevenu nul) Renvoyer 1

Sinon Renvoyer 0

centralelille

AUDACE • EXIGENCE • RESPECT

20 min

Arbres AVL : Complexité

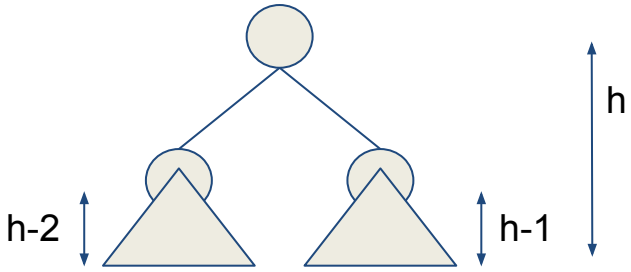
Complexité

- La complexité des opérations d'insertion et de suppression, incluant le rééquilibrage éventuel, est fonction de la hauteur de l'arbre AVL
- Exemple de l'insertion d'un élément :
  - L'accrochage du nouvel élément nécessite un nombre de comparaisons inférieur ou égal à la hauteur de l'arbre + 1
  - S'il y a rééquilibrage, alors la réorganisation de l'arbre ne concerne que les nœuds sur la branche du nouvel élément : leur nombre est inférieur ou égal à la hauteur de l'arbre + 1

Complexité = f(hauteur)
Hauteur minimale d'un AVL de n nœuds ?

- Hauteur minimale d'un arbre AVL de n nœuds ?
  - Cas d'un arbre binaire équilibré complet
  - Cf. calculs précédents : $h_{min} = \lceil \log_2(n+1) \rceil - 1$

Complexité = f(hauteur)
Hauteur maximale d'un AVL de n nœuds ?



- Construire un AVL de hauteur maximale ?
  - Il faut le déséquilibrer
  - On ne peut pas le dégénérer complètement car c'est un AVL…

    ⇒ Seule forme possible : deux sous-arbres de hauteur h-1 et h-2
- Nombre de nœuds dans un arbre de hauteur h ayant cette structure ?
  - n(0) = 1 ; n(1) = 2
  - n(h) = 1 + n(h-1) + n(h-2)

Complexité = f(hauteur)
Hauteur maximale d'un arbre de n nœuds ?

- $n(0) = 1 ; n(1) = 2 ; n(h) = 1 + n(h-1) + n(h-2)$
- Changement de variable : $\beta(h)=n(h)+1$
- $\beta(h) = \beta(h-1) + \beta(h-2) ; \beta(0) = 2 ; \beta(1) = 3$
- $\beta(h) = F_{h+3}$ où F_k est le $k^{ième}$ terme de la suite de Fibonacci
- $F_{h+3} \lesssim \varphi^{h+3}/\sqrt{5}$
- $n(h) \lesssim \varphi^{h+3}/\sqrt{5} - 1$
- $\varphi^{h+3} \lesssim \sqrt{5}\,(n+1)$
- $h + 3 \lesssim \log_\varphi(\sqrt{5}) + \log_\varphi(n+1)$
- $h \lesssim \log_\varphi(n+1) - 1{,}33 < 1{,}44 \log_2(n+1)$

$$F_n = \frac{1}{\sqrt{5}}\left(\left(\frac{1+\sqrt{5}}{2}\right)^n - \left(\frac{1-\sqrt{5}}{2}\right)^n\right)$$

Nombre d'or : $\varphi = \frac{1+\sqrt{5}}{2}$

Hauteur d'un arbre AVL de n nœuds ?

- $h_{min} = \lceil \log2(n+1) \rceil - 1 > \log2(n+1) - 1$
- $h_{max} < 1{,}44 \log_2(n+1)$

$$\log2(n+1) - 1 < h < 1{,}44 \log_2(n+1)$$

⇒ La complexité des opérations de recherche, d'insertion et de suppression sur un arbre AVL est en **Θ(** log(n) **)**

Suite de Fibonacci

- $F_0 = 0; F_1 = 1; F_n = F_{n-1} + F_{n-2}$
- Soit v_n un vecteur de N2 défini par : $v_n = \begin{pmatrix} F_n \\ F_{n+1} \end{pmatrix}$
- Application f : $v_n \longrightarrow v_{n+1}$
- Matrice de f : $A = \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix}$ $v_{n+1} = \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix} v_n$
- Recherche des valeurs propres de A : polynôme caractéristique $P(x) = \det(x.I-A) = \det\begin{pmatrix} x & -1 \\ -1 & x-1 \end{pmatrix} = x^2-x-1$
- $\Delta = 5$
- Deux racines : $\lambda_1 = (1+\sqrt{5})/2$ $\lambda_2 = (1-\sqrt{5})/2$

Suite de Fibonacci

- A est semblable à une matrice diagonale $B = \begin{pmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{pmatrix}$
  - Avec $A = P\,B\,P^{-1}$
  - P matrice de passage vers une base de vecteurs propres

$$P = \begin{pmatrix} 1 & 1 \\ \lambda_1 & \lambda_2 \end{pmatrix} \quad P^{-1} = \begin{pmatrix} -\lambda_2 & 1 \\ \lambda_1 & -1 \end{pmatrix} . \, 1/\sqrt{5}$$

$$\begin{pmatrix} F_n \\ F_{n+1} \end{pmatrix} = A^n \begin{pmatrix} F_0 \\ F_1 \end{pmatrix} = P\,B^n\,P^{-1} \begin{pmatrix} 0 \\ 1 \end{pmatrix}$$

Nombre d'or : $\varphi = \dfrac{1+\sqrt{5}}{2}$

- Formule de Binet : (1786-1856)

$$F_n = \frac{1}{\sqrt{5}} \left(\left(\frac{1+\sqrt{5}}{2} \right)^n - \left(\frac{1-\sqrt{5}}{2} \right)^n \right)$$

centralelille

AUDACE • EXIGENCE • RESPECT

10 min

Projet Fil Rouge 2021

Cahier des charges

Organisation

Evaluation

Cahier des charges
Programme 1

- Développer un programme permettant d'afficher graphiquement les étapes de construction d'un arbre AVL contenant les <n> premiers mots d'un fichier dont le chemin est fourni en ligne de commande
  - Le paramètre <n> sera également fourni en ligne de commandes
  - Dans la représentation graphique, chaque nœud de l'arbre devra afficher le mot contenu dans le nœud ainsi que le facteur de déséquilibre de ce nœud

Cahier des charges Programme 2

**signature d'un mot : chaîne de caractères contenant les caractères du mot triés par ordre alphabétique croissant*

- Développer un programme permettant d'indexer les mots d'un dictionnaire, dont le chemin est fourni en ligne de commande, dans un arbre équilibré
- Chaque noeud contiendra :
  - Un champ représentant la signature* des mots enregistrés dans ce noeud
  - C'est ce champ qui servira de clé pour la relation d'ordre de l'arbre AVL
  - La liste des mots du dictionnaire présentant cette signature
- Le programme affichera :
  - La taille des mots du dictionnaire
  - Le nombre de mots du dictionnaire
  - La durée de construction de l'arbre en millisecondes
  - Le nombre de noeuds et la hauteur de l'arbre AVL construit
  - La hauteur minimale d'un arbre contenant le même nombre de noeuds

Cahier des charges
Programme 2

- Une fois l'arbre créé, le programme bouclera pour permettre à l'utilisateur de saisir un mot à rechercher dans l'arbre.
- Il affichera :
  - La liste des mots présentant la même signature que le mot saisi
  - La profondeur du noeud contenant ce mot dans l'arbre
  - Le temp nécessaire pour trouver ce mot dans l'arbre en millisecondes

Cahier des charges Programme 3

**anagramme d'un mot : un autre mot contenant les mêmes caractères dans un ordre différent*

- Développer un programme permettant de rechercher tous les anagrammes* présents dans le dictionnaire dont le chemin est fourni en ligne de commande
- Le programme commencera par afficher le nombre de mots du dictionnaire disposant d'anagrammes
- Le programme affichera ensuite ces mots et leurs anagrammes en les triant par nombre d'anagrammes décroissant

Ressources

- Fichiers dictionnaire
- Jeux d'essais à rendre pour chaque programme
- Cahier des charges détaillé
- https://drive.google.com/drive/folders/1ccVMgulG2IwdHjY5cT3h_eaoAxcuT5CZ?usp=sharing

Cadrage séance 5 :

- Pas de test en séance
- Retours et conseils individuels sur le travail des groupes
- Séance de développement avec possibilité de demander des conseils à l'intervenant présent

Organisation / Evaluation

- Équipes de 4 étudiants du même groupe de TP
  - Objectif : au maximum 4 groupes d'étudiants par groupe TP
- Remise du travail :
  - Rendre code + CR au plus tard 24h avant la dernière séance
  - Attention aux critères de qualité (livraison, CR, gestion de projet...) énoncés précédemment ! (capsule 6)
- Evaluation :
  - Qualité du code, du CR, de la livraison, de la gestion de projet
  - Comparaison de l'efficacité des programmes pour des graphes de taille et complexité croissantes
    - Un classement sera établi sur toute la promo
    - La note finale du fil rouge dépendra en partie de ce classement

Dernière séance

- Examen de 2h sur tout le contenu AAP
  - sur ordinateur
- Soutenances de 30 minutes/ groupe
  - 20 minutes de présentation, 10 minutes de questions

centralelille

AUDACE • EXIGENCE • RESPECT

Code Couleur

Légende des textes

- mot-clé important, variable, contenu d'un fichier, code source d'un programme
- chemin ou url, nom d'un paquet logiciel
- commande, raccourci
- commentaire, exercice, citation
- culturel, optionnel

Culturel / Approfondissement

- A ne pas connaître intégralement par coeur
  - Donc, le reste… est à maîtriser parfaitement !
- Pour anticiper les problématiques que vous rencontrerez en stage ou dans d'autres cours
- Pour avoir de la conversation à table ou en soirée...

Exemples ou Exercices

- Brancher le cerveau
- Participer
- Expérimenter en prenant le temps...

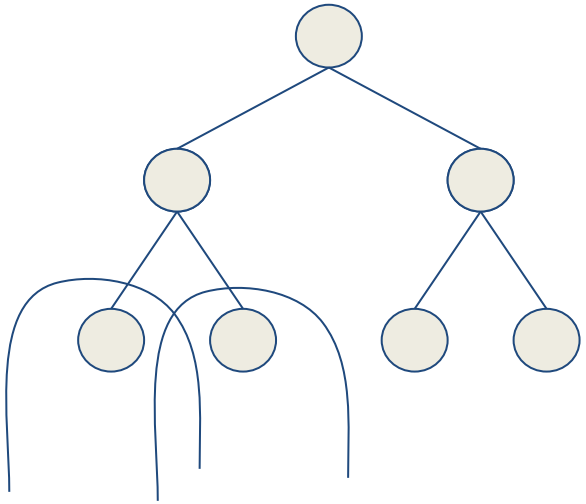
Bonnes pratiques, prérequis

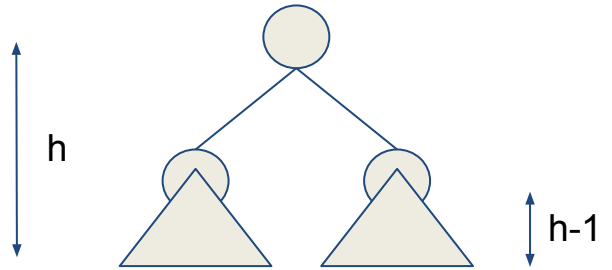
- Des éléments d'organisation indispensables pour un travail de qualité
- Des rappels de concepts déjà connus

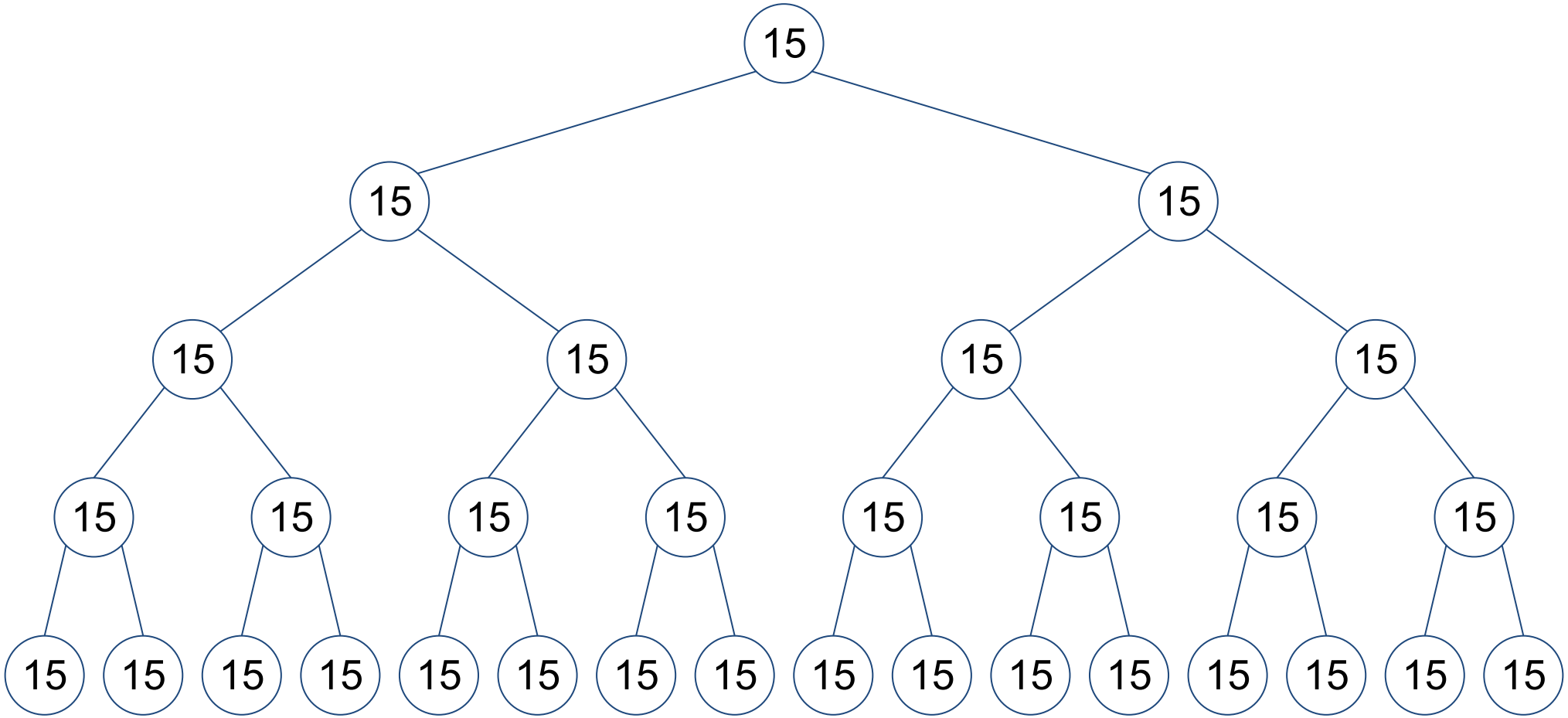
centralelille

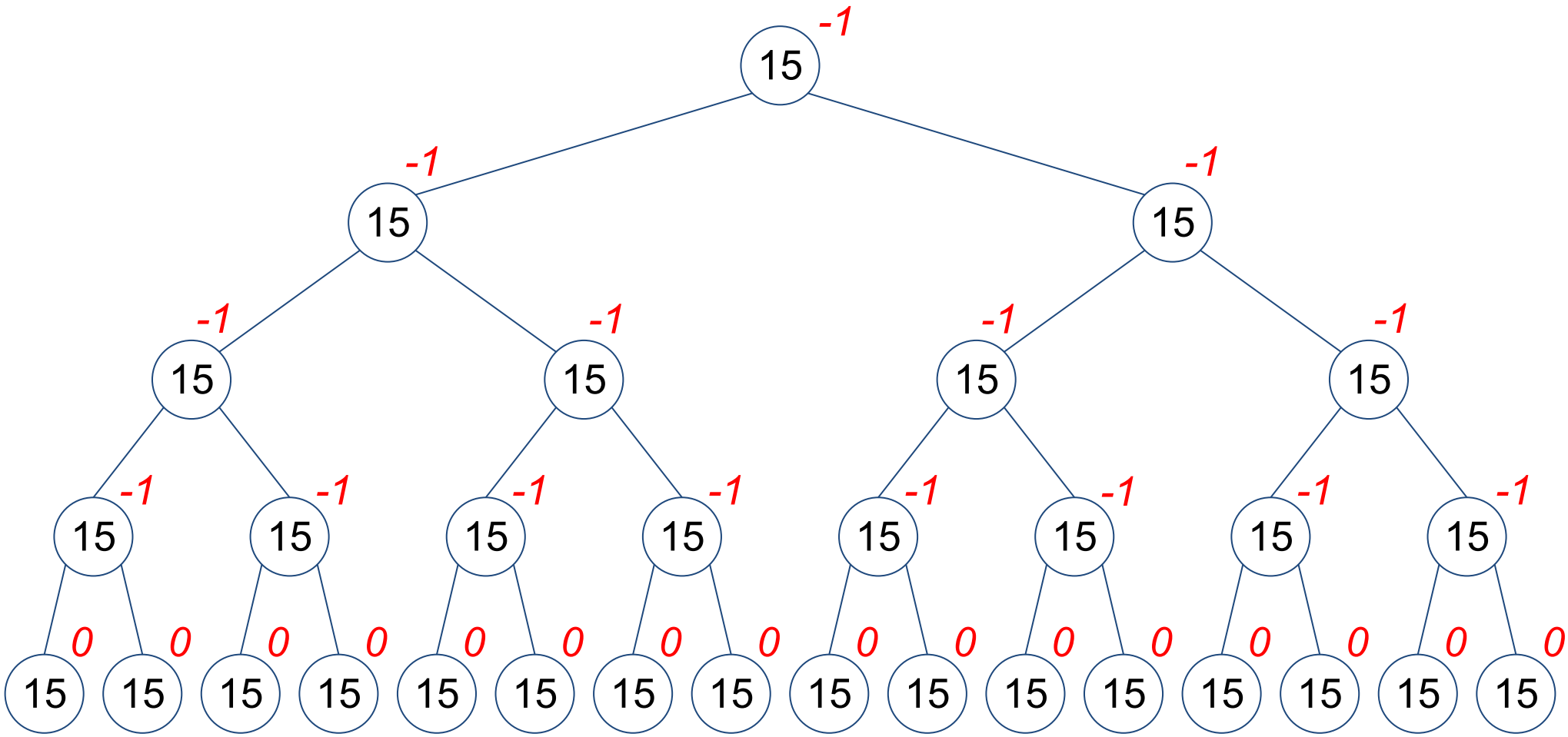
AUDACE • EXIGENCE • RESPECT

Annexes

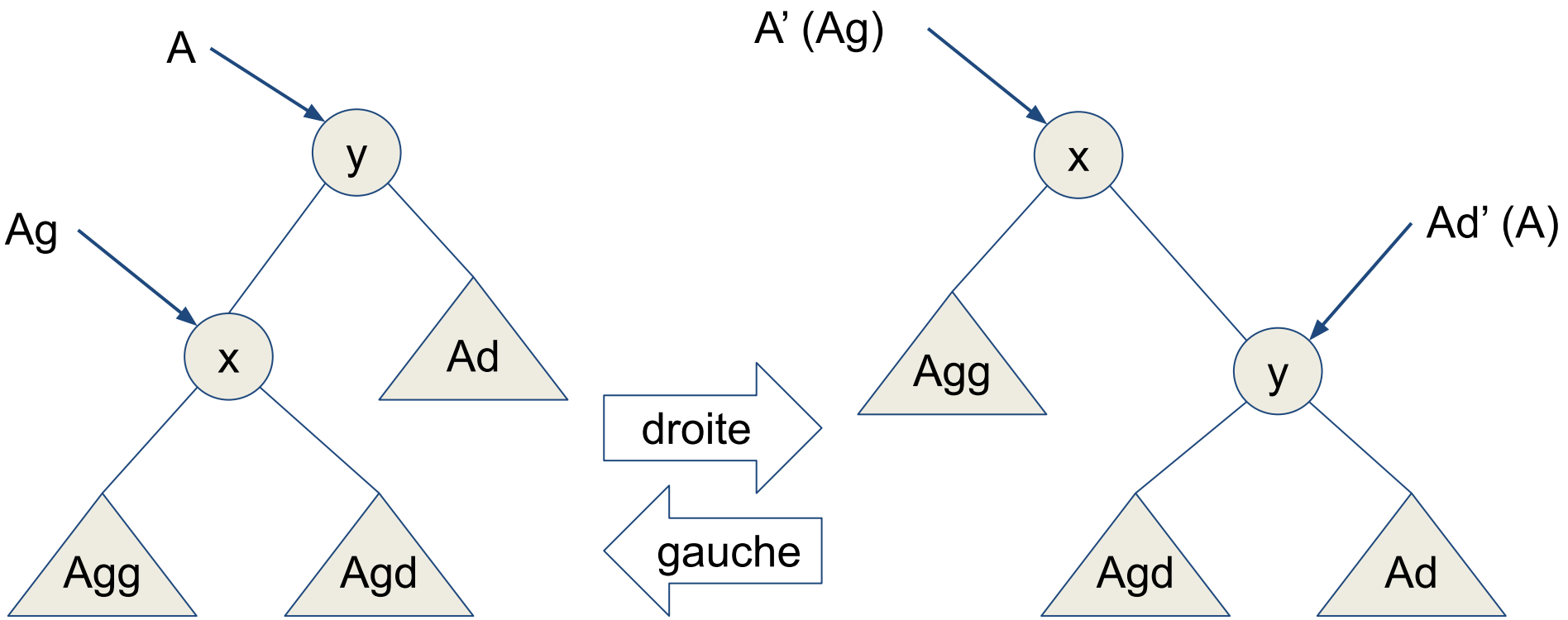



h
h-1

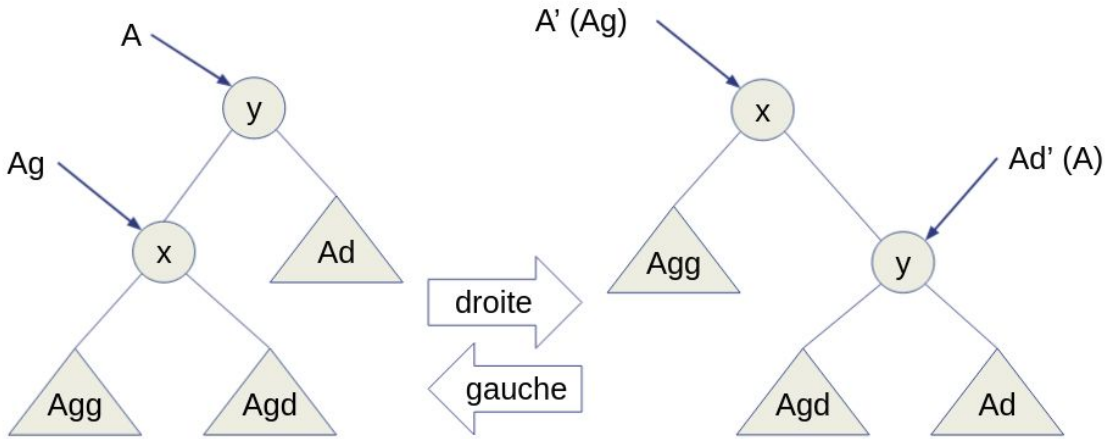

15
15
15
15
15
15
15
15
15
15
15
15
15
15
15
15
15
15
15
15
15
15
15
15
15
15
15
15


15 -1
15 -1
15 -1
15 -1 15 -1
15 -1 15 -1 15 -1 15 -1
15 -1 15 -1 15 -1 15 -1 15 -1 15 -1 15 -1 15 -1
15 0 15 0 15 0 15 0 15 0 15 0 15 0 15 0 15 0 15 0 15 0 15 0 15 0 15 0 15 0 15 0

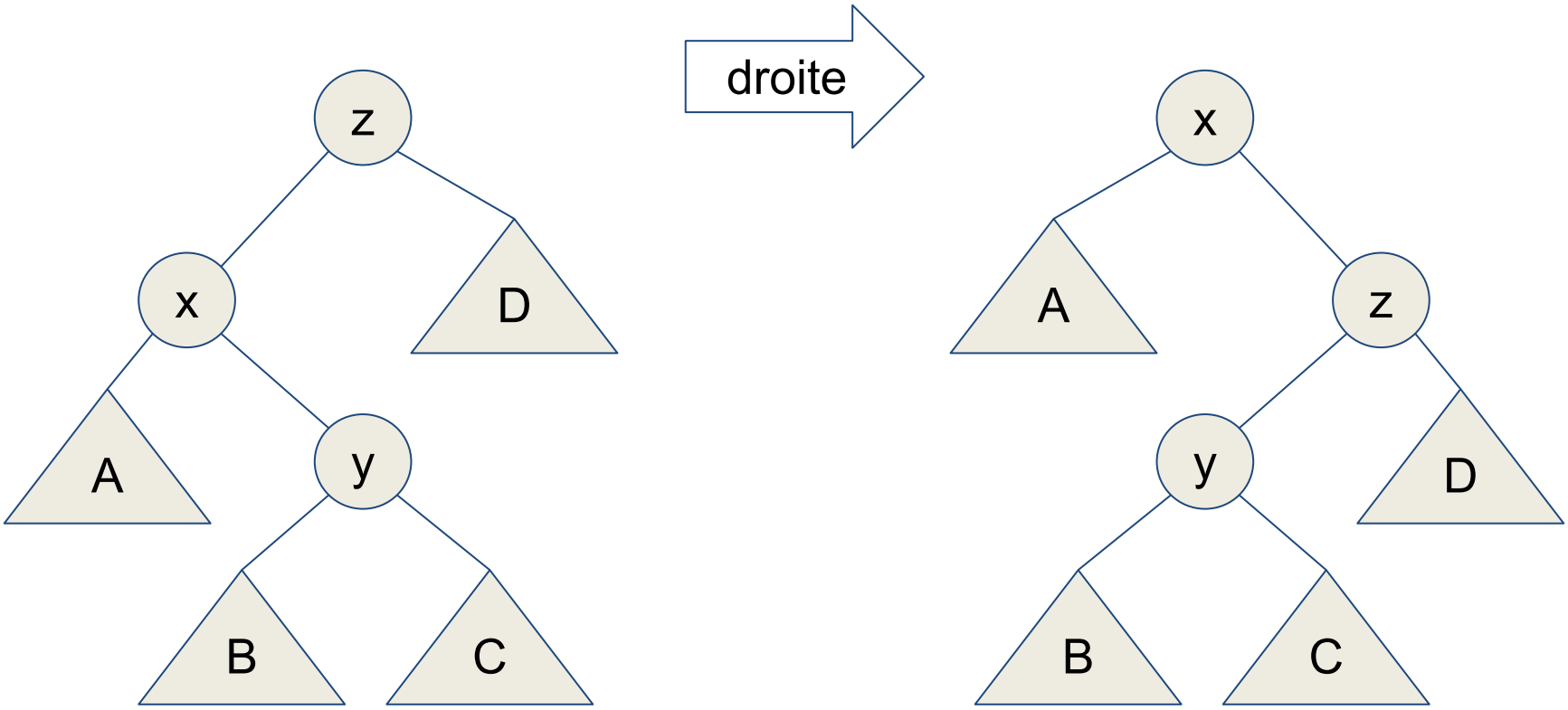
Rotations simples



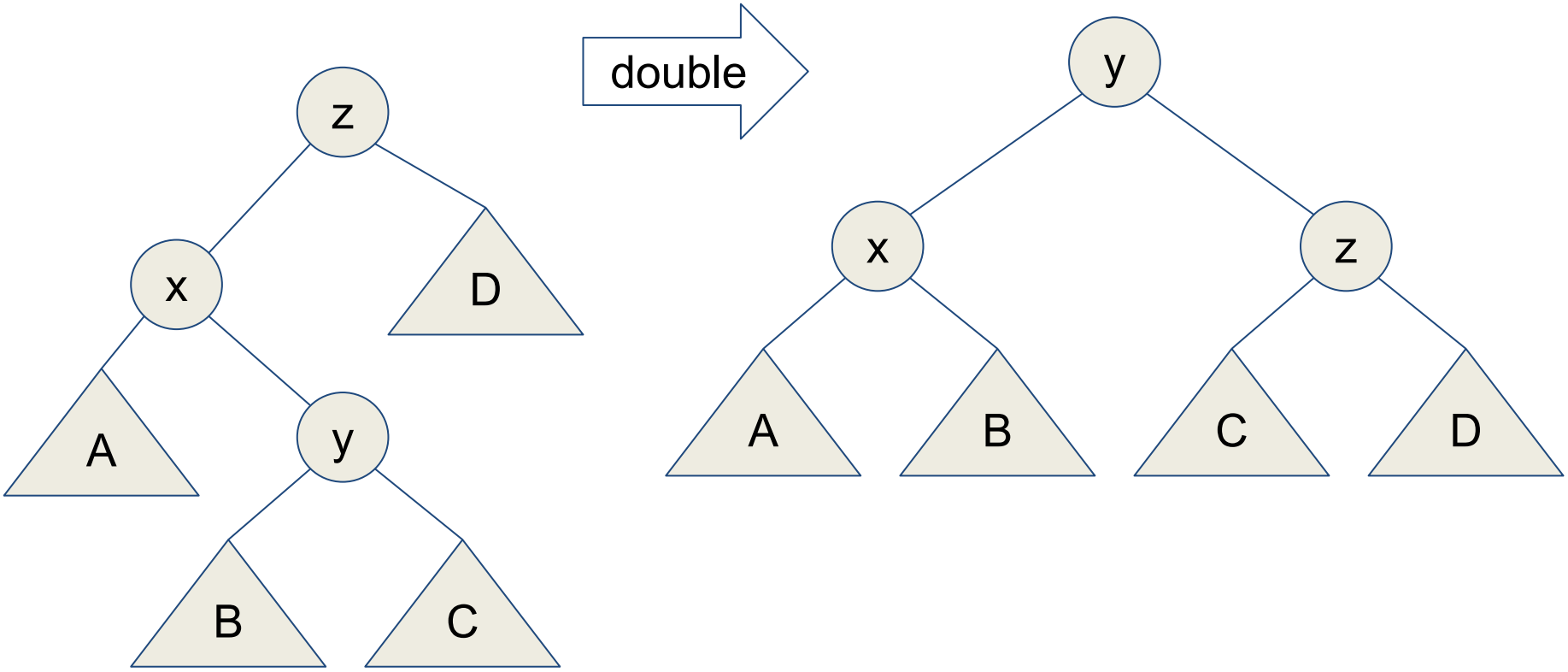
Rotation simple insuffisante



- Une rotation droite est insuffisante si h(Agd) > h(Agg)



Rotation double à droite
= Rotation gauche-droite



Rotation gauche-droite