6 TAD Arborescentes

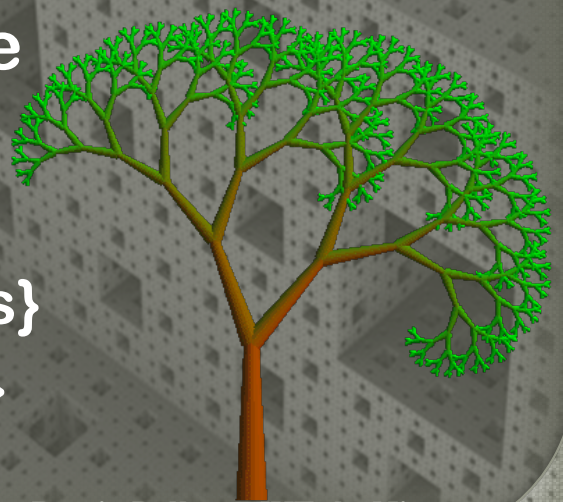
Arbre, Arbre Binaire, AVL…

Définition

Manipulation (Insertion, Suppression, Parcours)

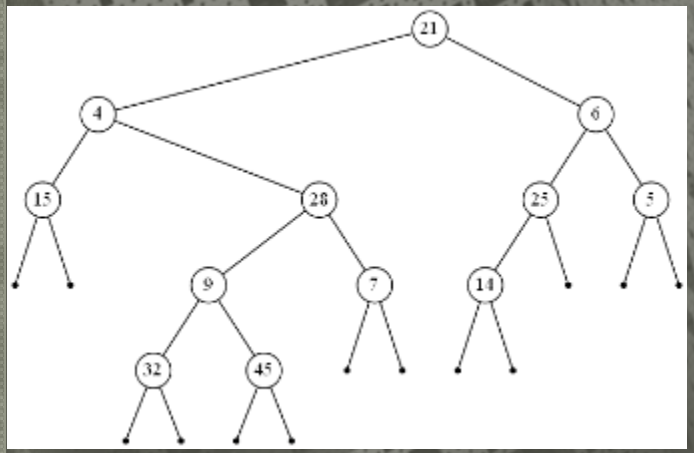
Définitions

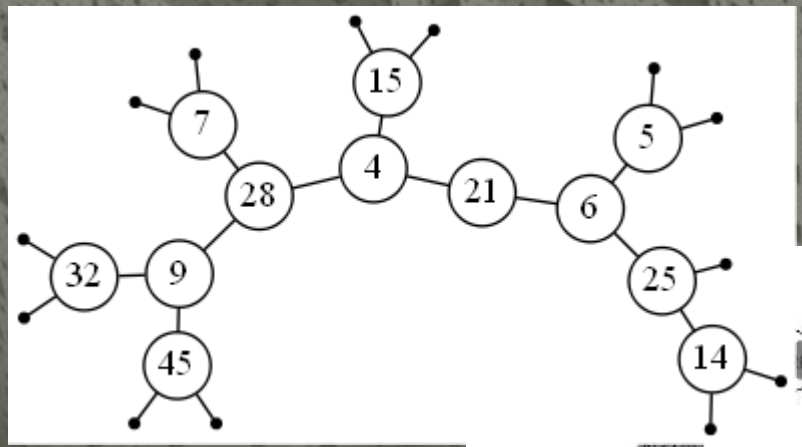
- **Arbre**
  - Ensemble de nœuds organisés de manière hiérarchique à partir du nœud racine
  - Structure fondamentale en informatique
    - Fichiers, expressions arithmétiques, exploration d'un ensemble de solutions (Tic-Tac-Toe)
  - Définition naturellement récursive
    - Manipulation récursive
    - Liste = {1 valeur & 1 suivant}
    - Arbre binaire = {1 valeur & 2 suivants}
    - Arbre n-aire = {1 valeur & n suivants}

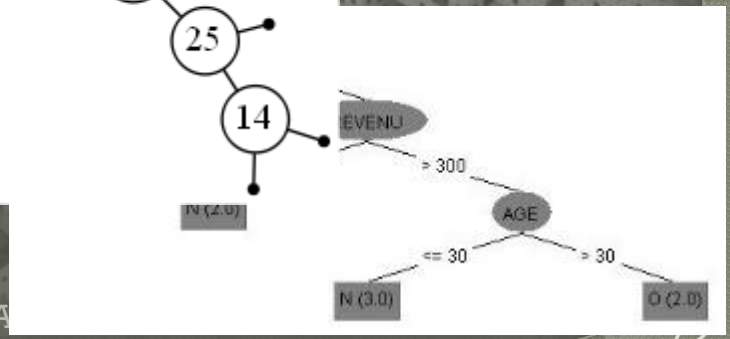


Arbre Binaire

- Composé de
  - une et une seule racine
  - un ensemble de nœuds ayant chacun 0, 1 ou 2 branches
- À chaque nœud est associé une donnée

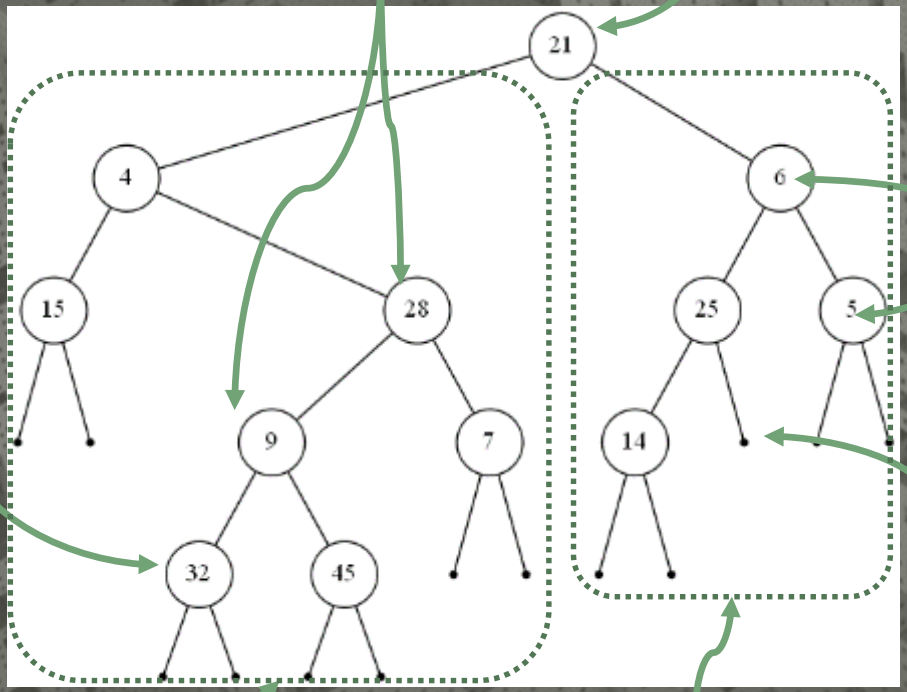






Arbre Binaire

- **Vocabulaire**



Arbre Binaire

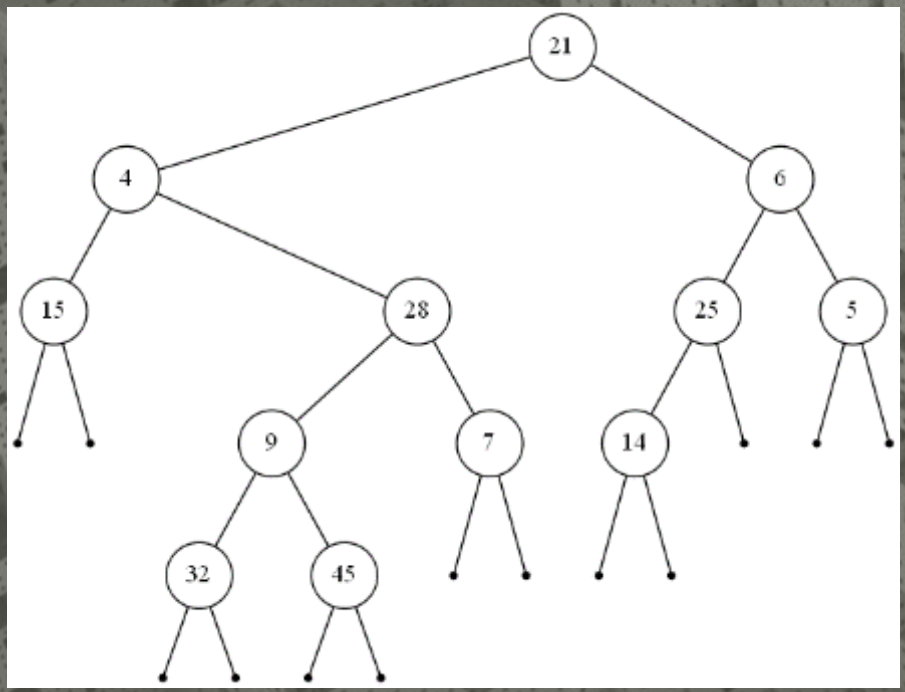
- ABin =
  - Soit ∅ (ABinVide)
  - Soit
    - Valeur
    - Sous-arbre gauche (SAG) qui est un ABin
    - Sous-arbre droit (SAD) qui est un ABin

TAD ArbreBinaire

- **TAD ABin**
  - Arbre binaire
  - Méthodes
    - ABin() : $\varnothing \rightarrow$ ABin
    - ABin() : T $\times$ ABin $\times$ ABin $\rightarrow$ ABin
    - SAG() : ABin $\rightarrow$ Abin
    - Taille() : ABin $\rightarrow$ Entier
    - Racine() : ABin $\rightarrow$ T
    - EstVide() : ABin $\rightarrow$ Bool
    - SAD() : ABin $\rightarrow$ Abin
    - Hauteur() : ABin $\rightarrow$ Entier
  - Pré-conditions (a $\in$ ABin)
    - SAG(a), SAD(a) & Racine(a) défini ssi $\neg$ EstVide(a)
  - Axiomes (v $\in$ T; g,d $\in$ ABin)
    - EstVide(ABin()) = Vrai
    - EstVide(ABin(v,g,d)) = Faux
    - SAG(ABin(v,g,d))=g
    - SAD(ABin(v,g,d))=d
    - Taille(ABin())=0
    - Taille(ABin(v,g,d))= 1+Taille(g)+Taille(d)
    - Hauteur(ABin())=0
      Hauteur(ABin(v,g,d))=1+Max{Hauteur(g),Hauteur(d)}
    - Si Taille(a)=n & Hauteur(a)=h alors h $\leq$ n $\leq 2^h - 1$
- https://en.wikipedia.org/wiki/Tree_(data_structure)

Arbre Binaire

- **Vocabulaire**



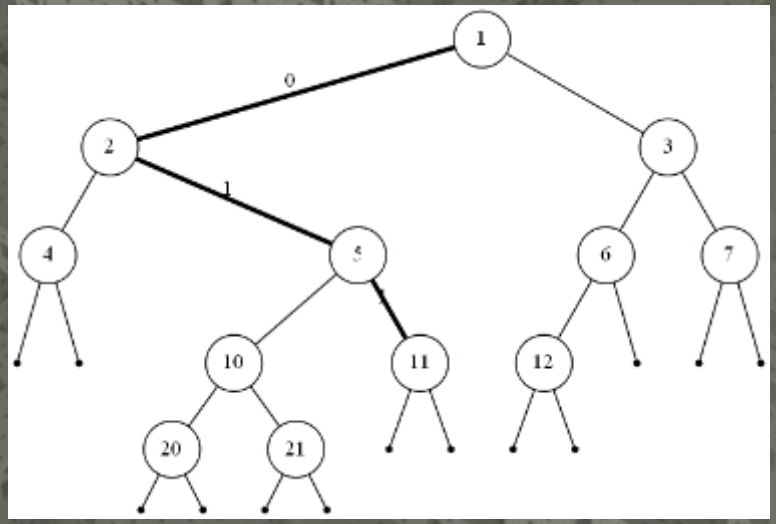
Numérotation hiérarchique

- Soit $\langle b_1, \ldots, b_k \rangle$ un chemin menant à a' avec

$$b_i = \begin{cases} 0 \; si \; sag \\ 1 \; si \; sad' \end{cases}, \forall \, 1 \leq i \leq k$$

- Le numéro de a' est $\langle 1 b_1 b_2 \ldots b_k \rangle_2$

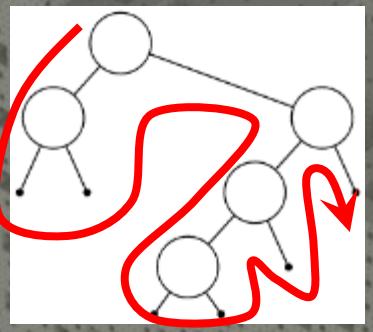
- Exemple
  - Racine = 1
  - Si *n* = numéro d'1 nœud
    Alors fils gauche = nx2
    fils droit = nx2 + 1

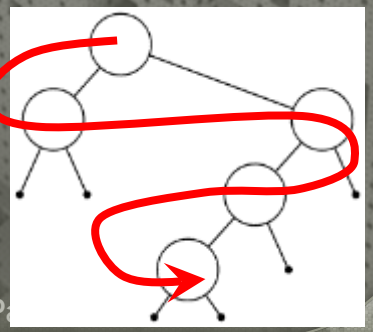


$<1\ 011>_2 = <11>_{10}$

Parcours

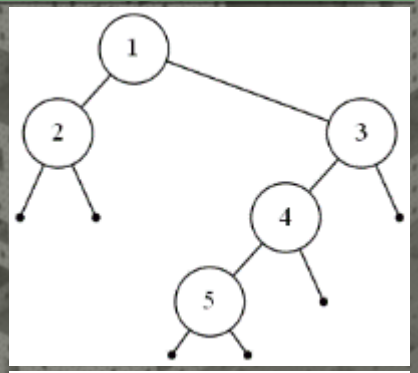
- Algorithme qui appelle une méthode M (affichage par ex.) sur tous les nœuds d'un arbre dans un ordre bien précis défini à priori
- Parcours en profondeur
  - Le parcours d'un sous-arbre est terminé avant d'entamer le parcours de l'autre sous-arbre
- Parcours en largeur
  - Se fait en parcourant les nœuds selon leur niveaux croissant

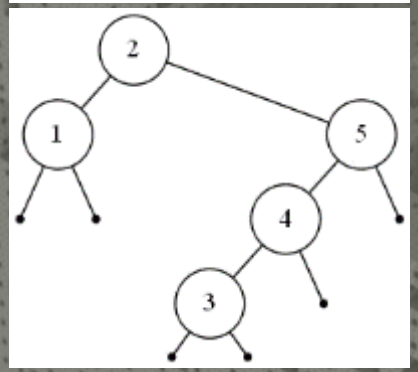


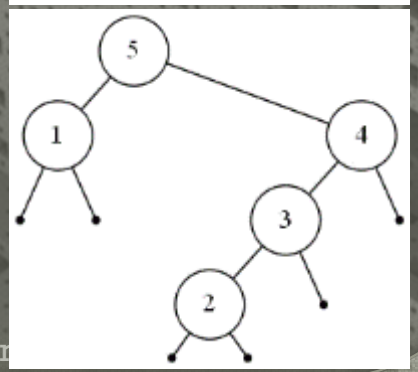


Parcours en profondeur

- **Préfixé**
  1. Appliquer M à la racine
  2. Préfixé de SAG
  3. Préfixé de SAD
- **Infixé (ou symétrique)**
  1. Infixé de SAG
  2. Appliquer M à la racine
  3. Infixé de SAD
- **Postfixé**
  1. Postfixé de SAG
  2. Postfixé de SAD
  3. Appliquer M à la racine





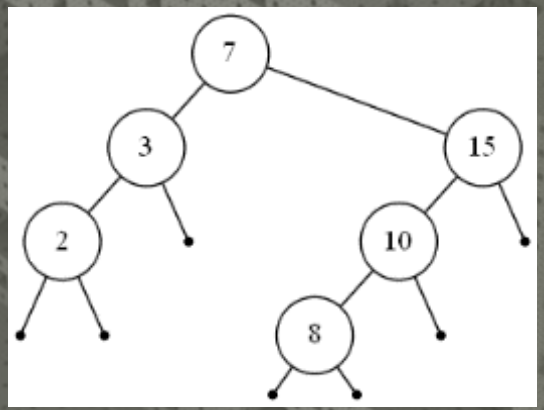


Parcours en largeur

```
void parcours_largeur(ABin a) {
        f = FileVide()
        f.Enfile(a)
        while (! f.EstVide()) {
                ssa = f.Defile()
                M(ssa)
                f.Enfile(SAG(ssa))
                f.Enfile(SAD(ssa))
        }
}
```

ABR

- Arbre Binaire de Recherche (ABin ordonné)
  - Les valeurs des nœuds du sag sont *strictement* inférieures à la valeur de la racine
  - Les valeurs des nœuds du sad sont *strictement* supérieures à la valeur de la racine
- Théorème
  - 1 ABin est 1 ABR ssi les valeurs des nœuds parcourus de manière *infixé* est *strictement* croissante
  - Ex : <2,3,7,8,10,15>
- Intérêts
  - Recherche, insertion plus rapide si arbre pas trop déséquilibré

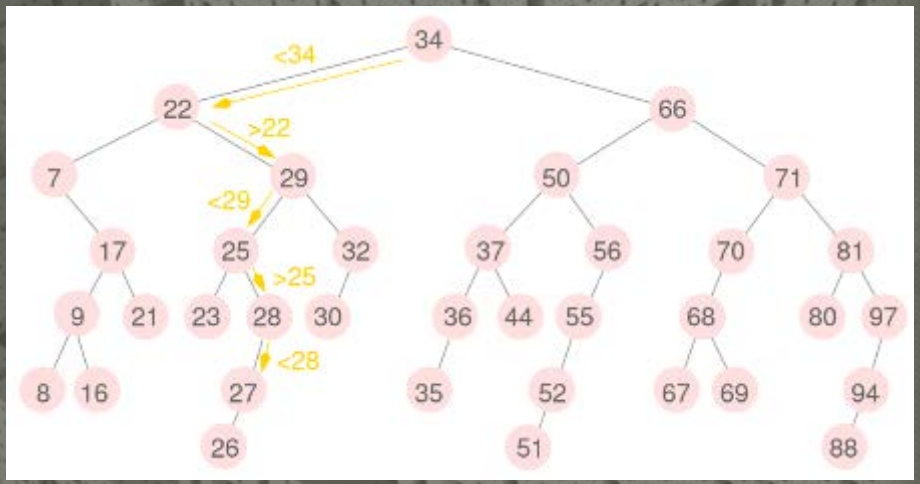


ABR

- Recherche

```
boolean estDansABR(ABR a, Element e) {
        if (estVide(a))
                return False;
        else if (racine(a)==e)
                return True ;
        else if (e<racine(a))
                return estDansABR(sag(a),e)
        else
                return estDansABR(sad(a),e)
        }
}
```

- Recherche de 27

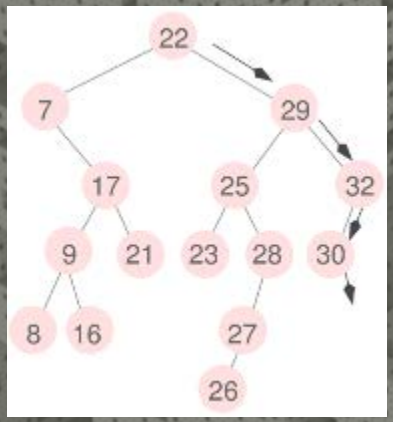


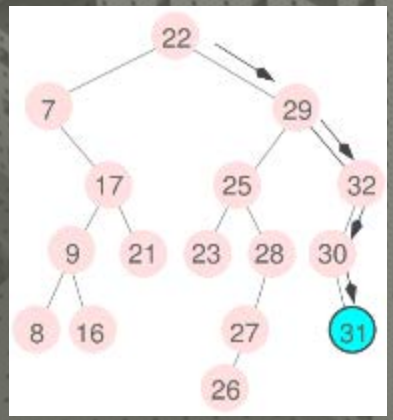
ABR

- Insertion

```
ABR insertionABR(ABR a, Element e) {
        if (estVide(a))
                return ABR(e,ABRVide, ABRVide);
        else if (racine(a)==e)
                return a ;
        else if (e<racine(a))
                return ABR(racine(a),insertionABR(sag(a),e),sad(a)) ;
        else
                return ABR(racine(a),sag(a), insertionABR(sad(a),e)) ;
        }
}
```

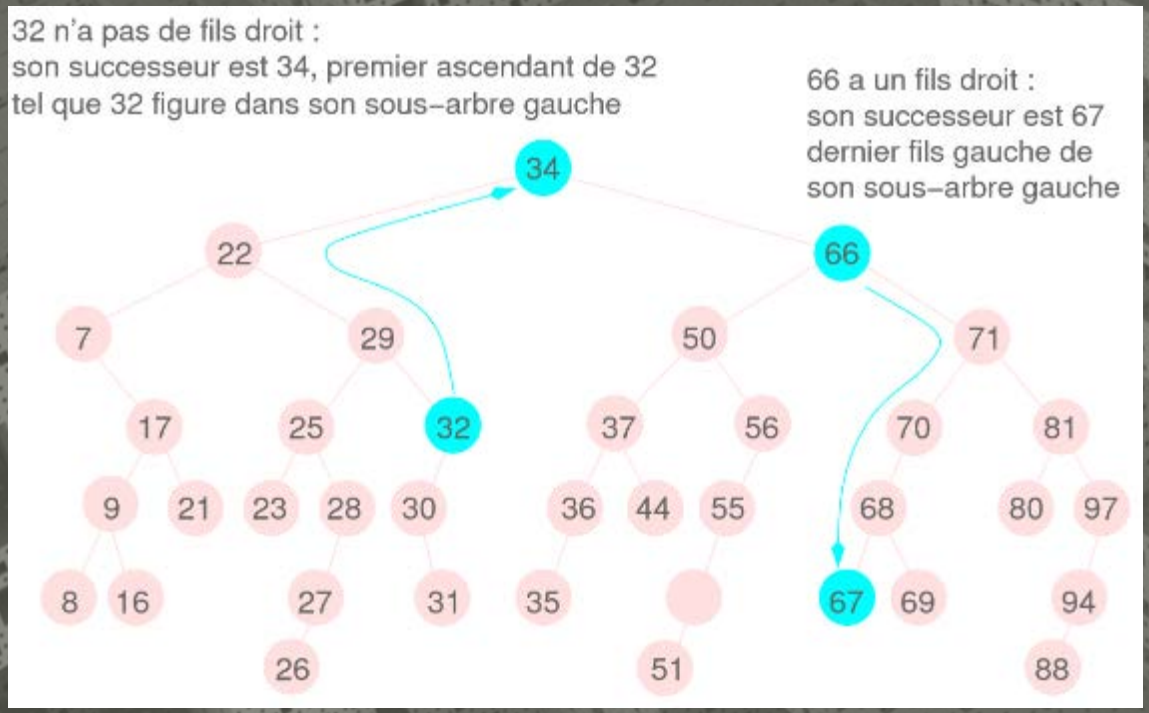
- Insertion de 31





ABR

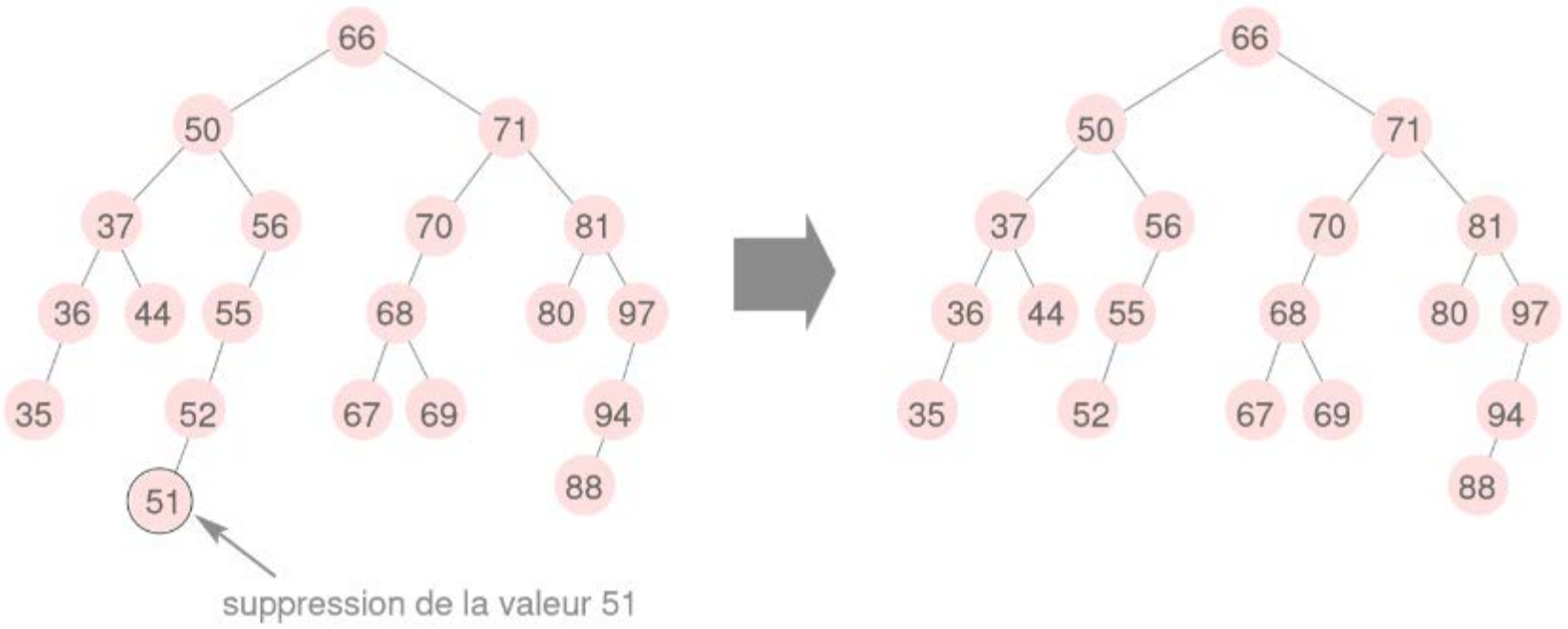
Successeur



- Problème
  - Nécessité de pouvoir accéder au père !

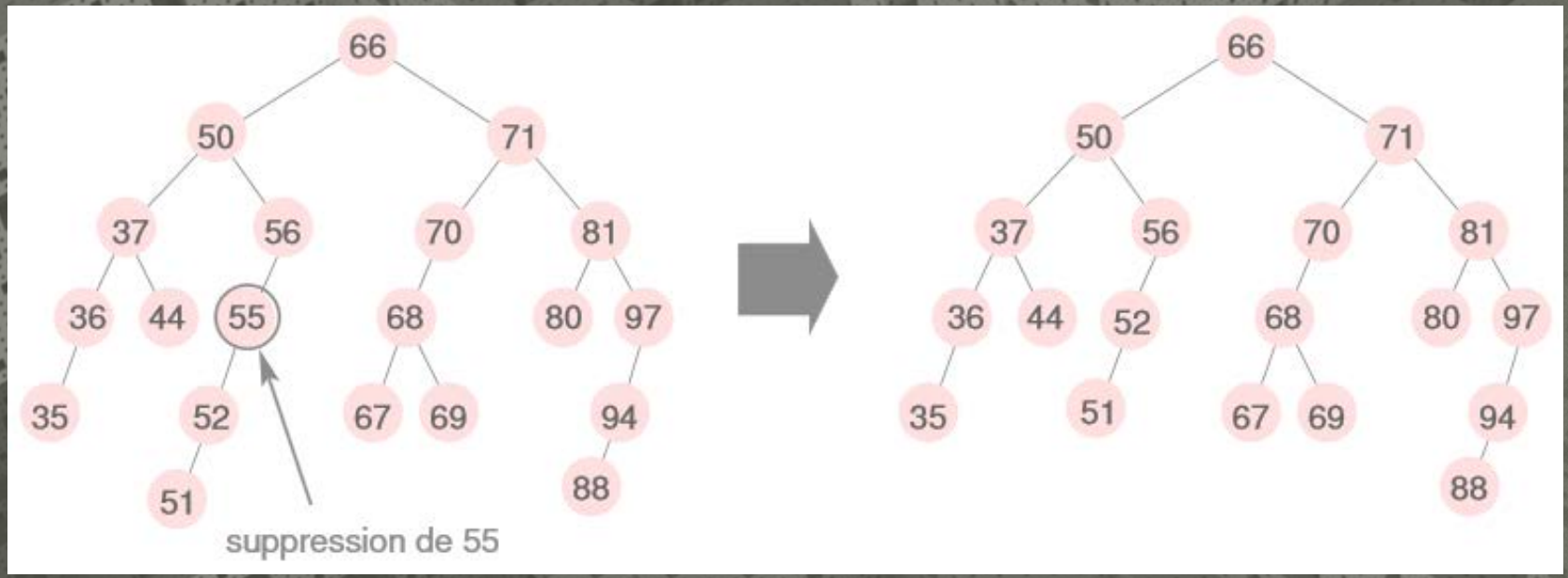
ABR

- Suppression
- Cas 1
  - Le nœud est une feuille



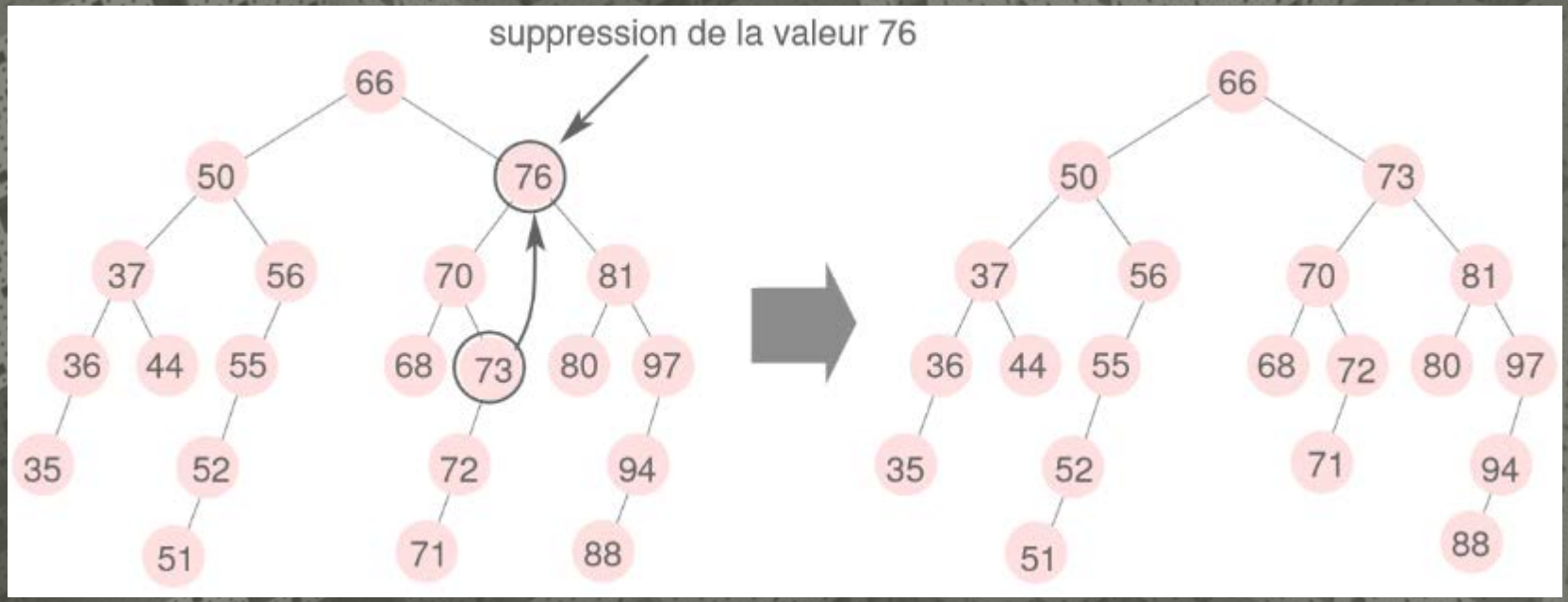
ABR

- Suppression
- Cas 2
  - Le nœud a un seul fils



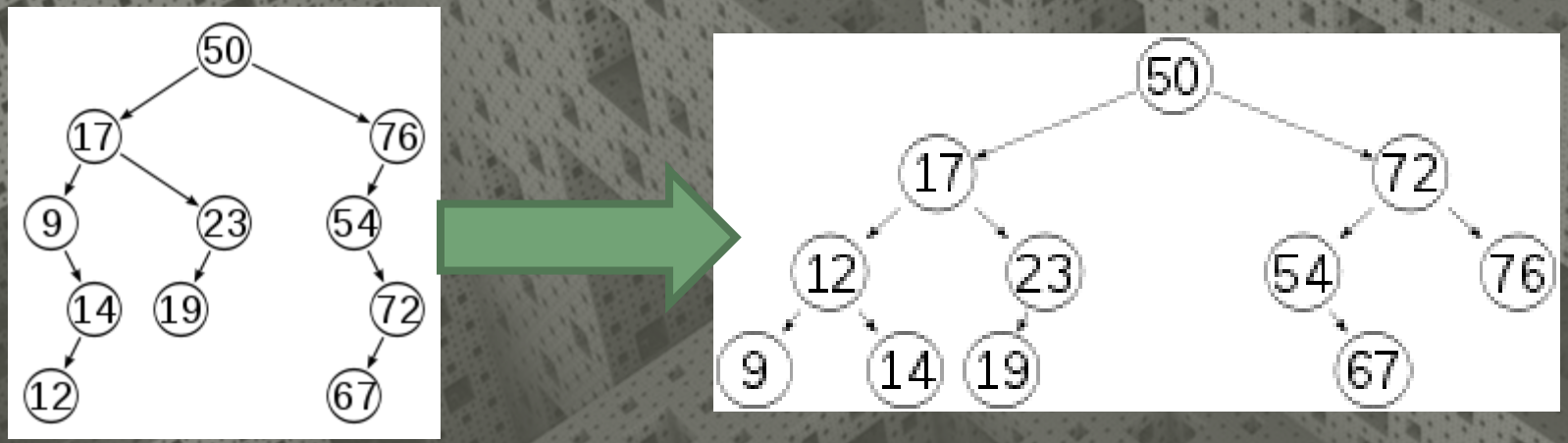
ABR

- Suppression
- Cas 3
  - Le nœud a 2 fils



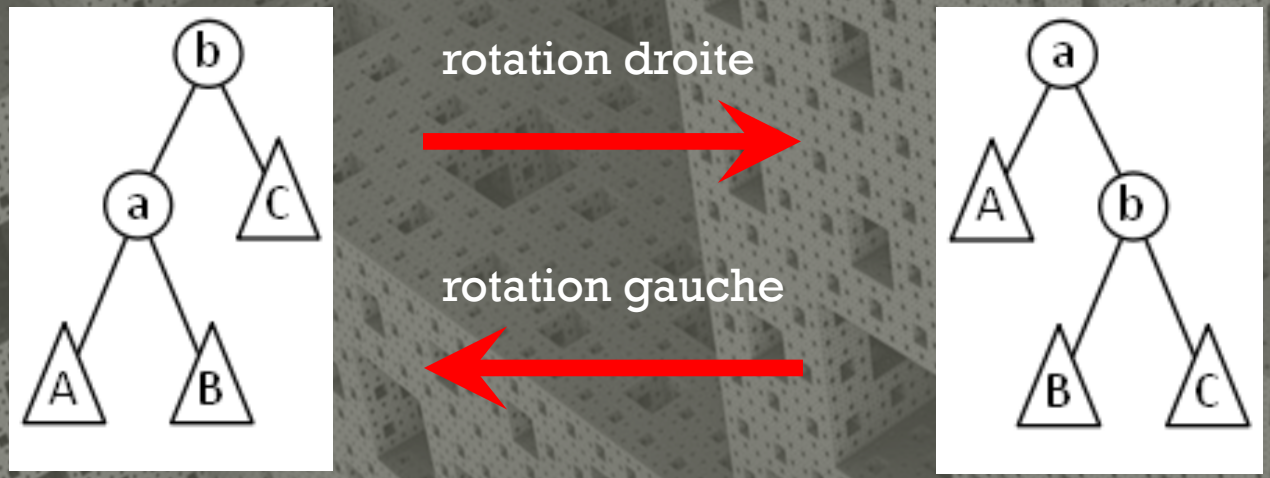
AVL (Adelson-Velskii & Landis)

- ABR pour lequel, en tout nœud, les hauteurs des sous-arbres gauche et droit diffèrent au plus de 1
- Arbre AVL = ABR tel que
  - $\forall a \in ABin, |haut(sag(a)) - haut(sad(a))| \leq 1$



AVL

- Équilibrage

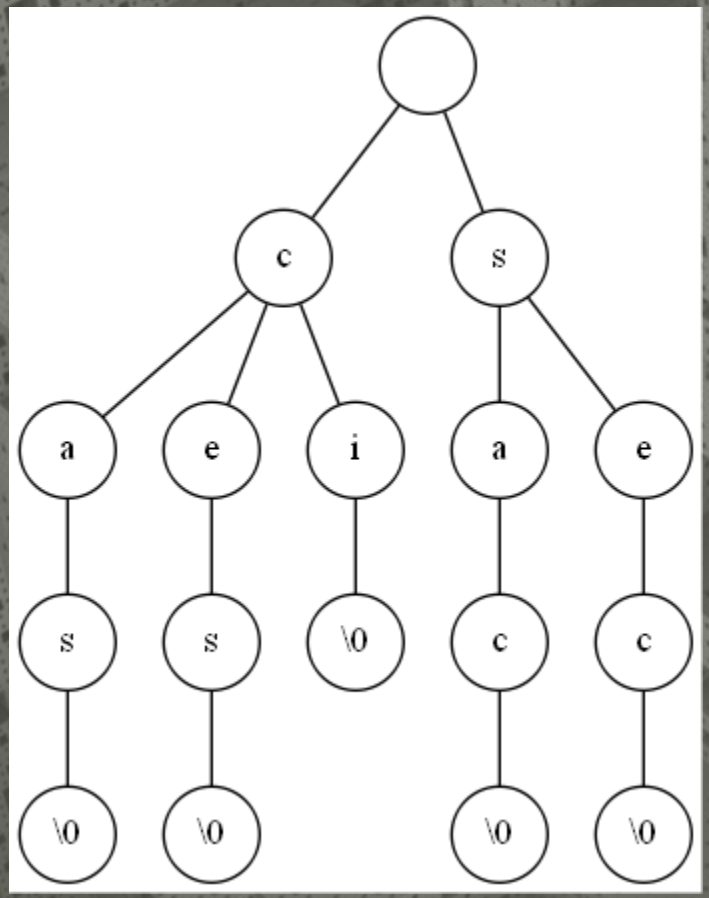


- Proposition
  - Après une insertion ou suppression, il suffit de 2 rotations pour ré-équilibrer un arbre

Arbres n-aire

- ANaire
  - Un nombre *n* de fils
- Implémentation Java
  - Jtree (Vue & Contrôleur)
  - DefaultMutableTreeNode (Modèle)

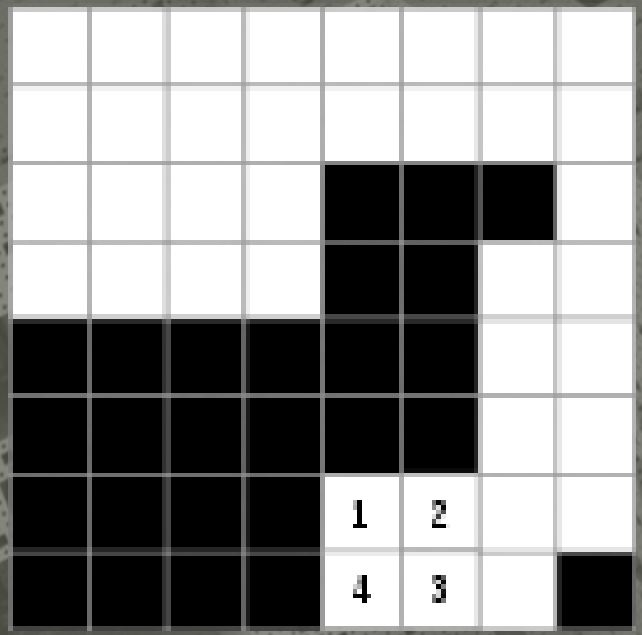
Dictionnaire

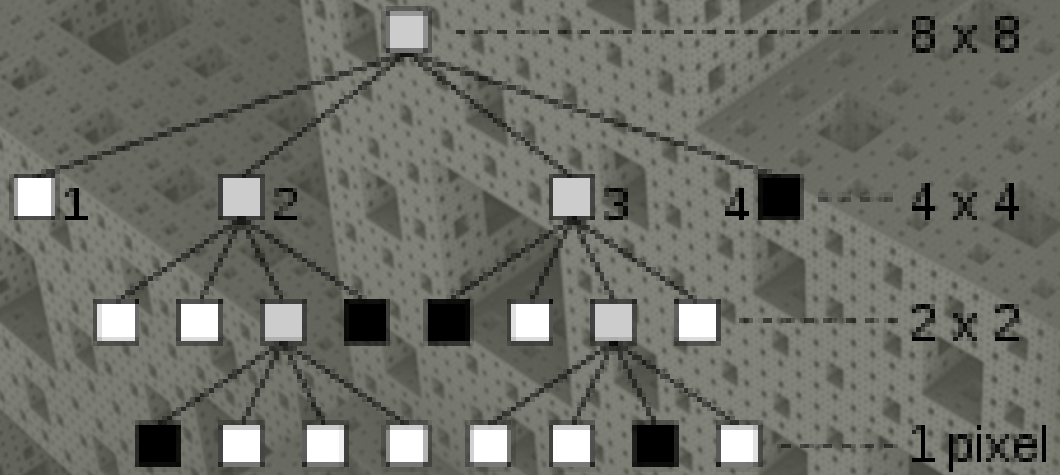


cas
ce
ces
ci
sa
sac

Quadtree (2D)

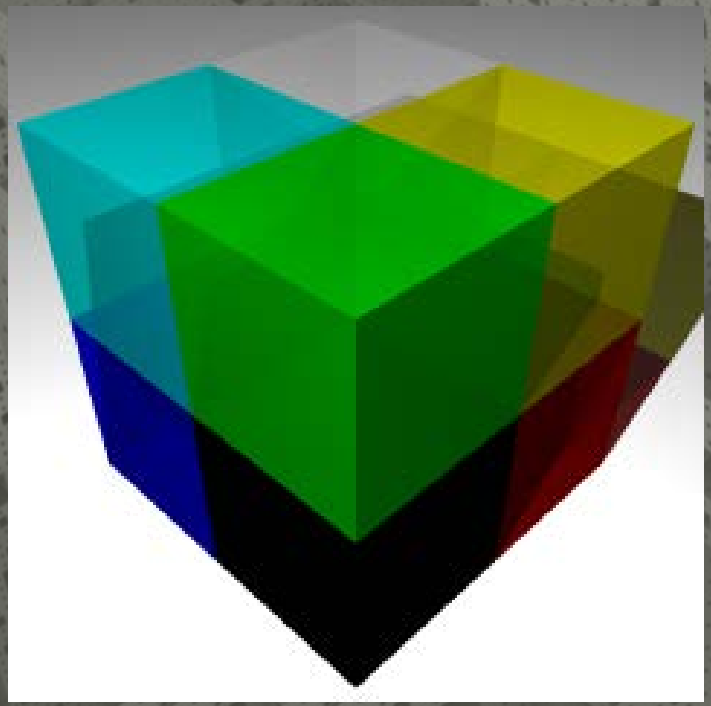
- Compression d'images





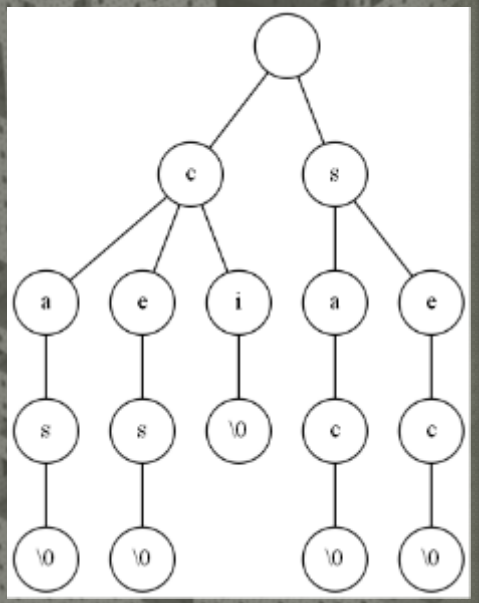
Octree (3D)

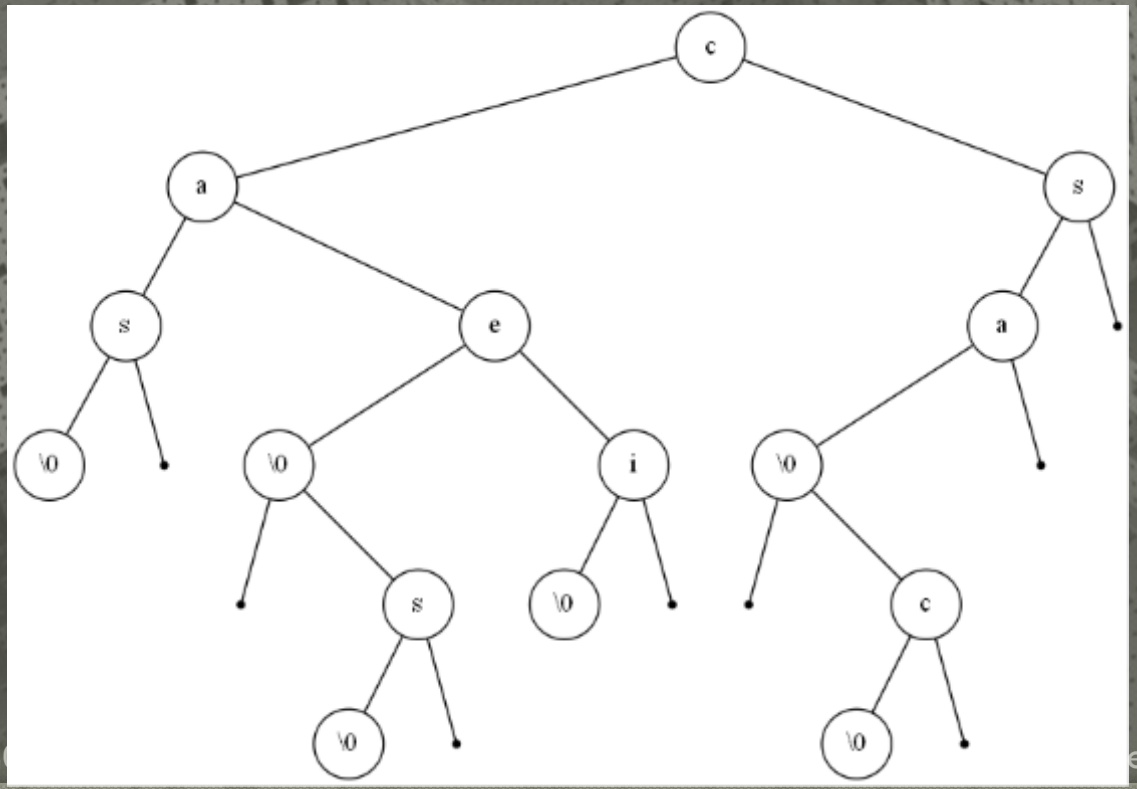
- Subdivision de l'espace 3D
  - Détection de collision en 3D



ANaire → ABin

- **Théorème**
  - Tout arbre n-aire peut être représenté par 1 arbre binaire

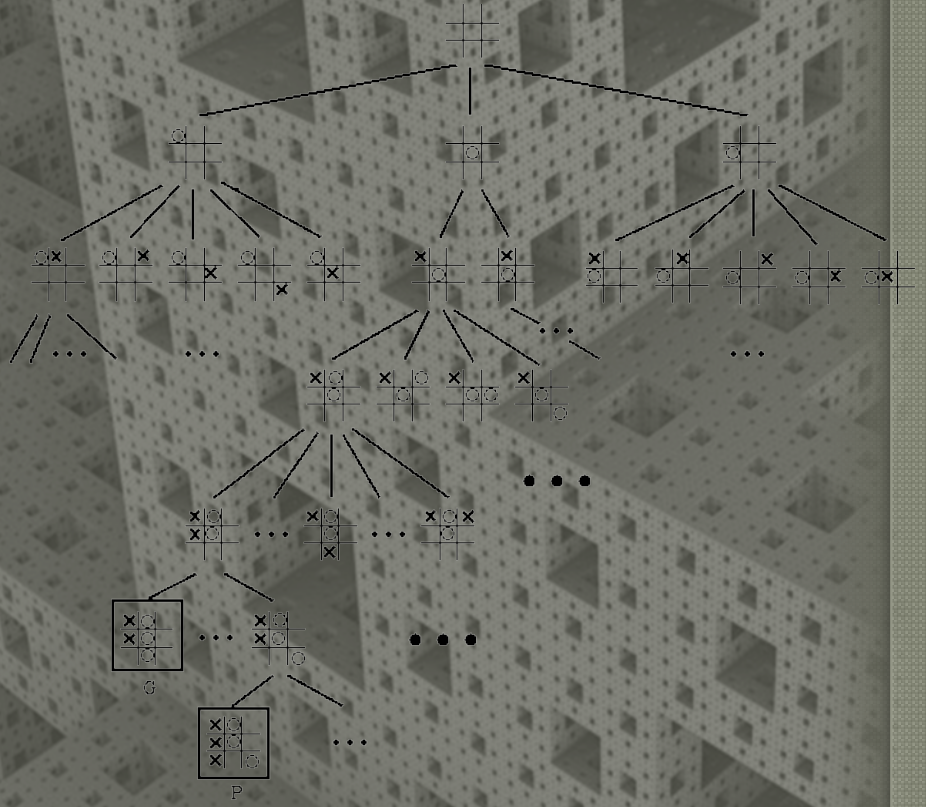




cas
ce
ces
ci
sa
sac

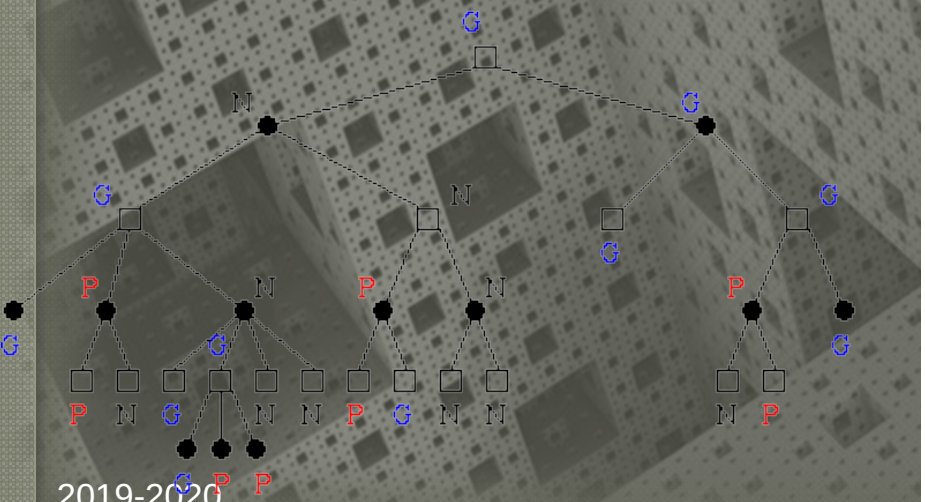
Utilisation ANaire

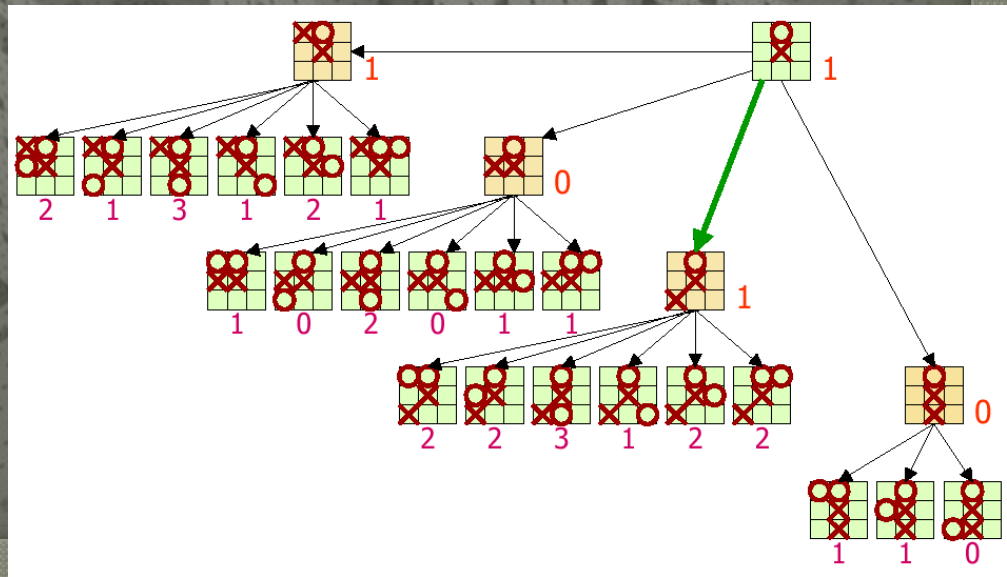
- Recherche dans un espace d'états
  - Arbre de recherche
    - Peut être infini alors que l'ensemble des états est finis
  - Plusieurs stratégies
    - Recherche en aveugle
    - Recherche avec heuristique



Algorithme Min-Max (Minimax)

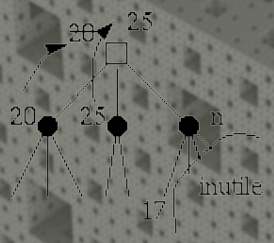
- **Jeu à 2 joueurs**
  - On suppose que chaque joueur choisira son meilleur coup, le meilleur coup de l'adversaire correspondant au moins bon coup du joueur courant.
    On "remonte" de proche en proche les valeurs des feuilles (fins de parties) jusqu'à la racine (position courante) ; on peut ainsi choisir le coup qui mène à la position la plus favorable

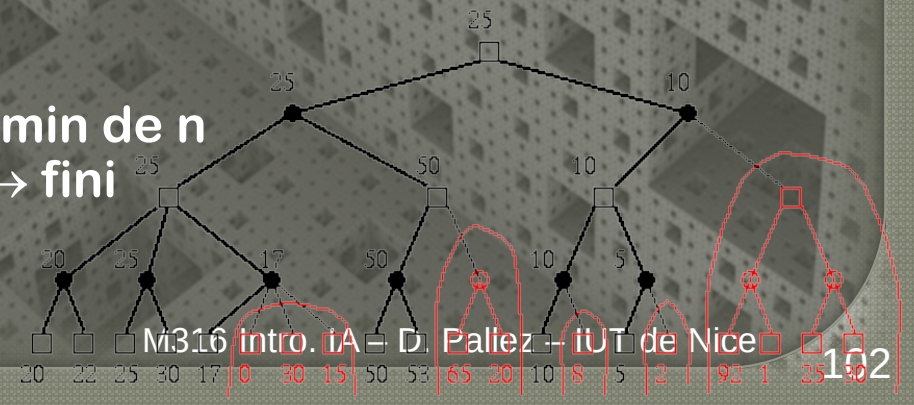


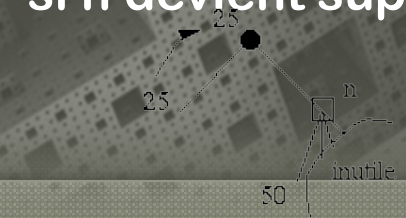


Algorithme $\alpha - \beta$

- **Jeu à 2 joueurs**
  - Optimisation de min-max
    - Élaguer des branches qu'il est inutile d'explorer
    - Si la valeur d'un nœud atteint un seuil , il est inutile de continuer à explorer les descendants de ce nœud → leur valeur n'interviendra pas
    - Coupe α (alpha)
      - n nœud min
        Seuil alpha = Max des ancêtres Max de n
        si n devient inférieur à Alpha → fini
    - Coupe β (beta)
      - n nœud max
        Seuil beta = min des ancêtres min de n
        si n devient supérieur à Beta → fini







Sources

- Florent Hivert, Algorithmique
- Elise Bouzon, Algo et Structures récursives
- Francois Denis, Algo & Structures de données 2