

# Arbres AVL et arbres quaternaires

Algorithmes et structures de données, 2025-2026

---

P. Albuquerque (B410) et O. Malaspinas (A401), ISC, HEPIA  
2026-04-01

En partie inspiré des supports de cours de P. Albuquerque

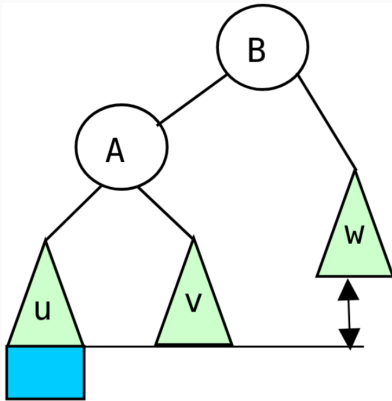
## Rappel : Algorithme d'insertion

- Insérer le noeud comme d'habitude.
- Mettre à jour les facteurs d'équilibre jusqu'à la racine (ou au premier noeud déséquilibré).
- Rééquilibrer le noeud si nécessaire.

# Les cas de déséquilibre

## Cas 1a

- $u$ ,  $v$ ,  $w$  même hauteur.
- déséquilibre en B après insertion dans  $u$



## Cas 1a

- Comment rééquilibrer ?

Figure 1 : Après insertion

# Les cas de déséquilibre

## Cas 1a

- u, v, w même hauteur.
- déséquilibre en B après insertion dans u

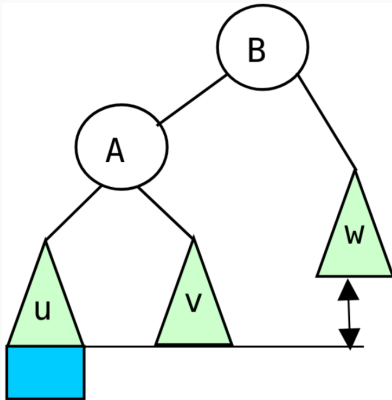


Figure 1 : Après insertion

## Cas 1a

- Comment rééquilibrer ?
- ramène u, v w à la même hauteur.
- v à droite de A (gauche de B)

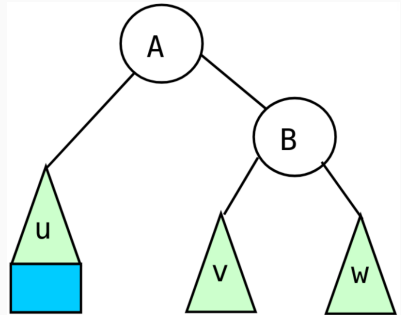


Figure 2 : Après équilibrage

# Les cas de déséquilibre

## Cas 1b (symétrique 1a)

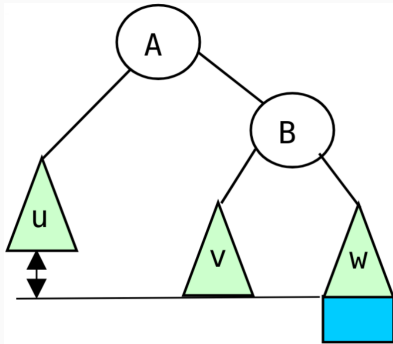


Figure 3 : Après insertion

## Cas 1b (symétrique 1a)

- Comment rééquilibrer ?

# Les cas de déséquilibre

Cas 1b (symétrique 1a)

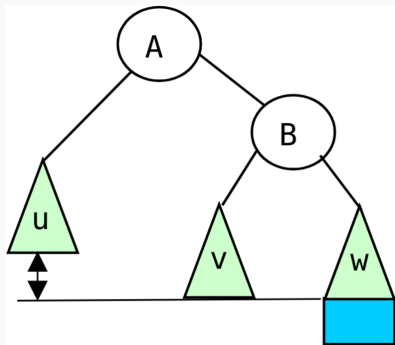


Figure 3 : Après insertion

Cas 1b (symétrique 1a)

- Comment rééquilibrer ?

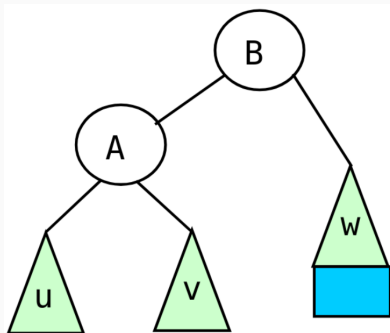
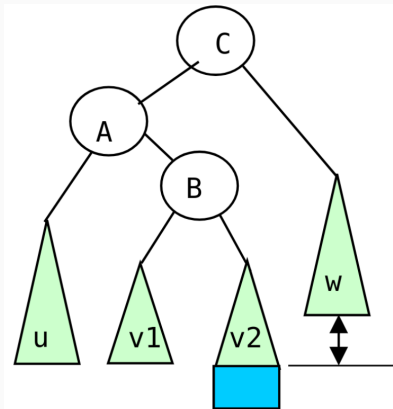


Figure 4 : Après équilibrage

# Les cas de déséquilibre

## Cas 2a

- $h(v1)=h(v2)$ ,  $h(u)=h(w)$ .
- déséquilibre en C après insertion dans v2



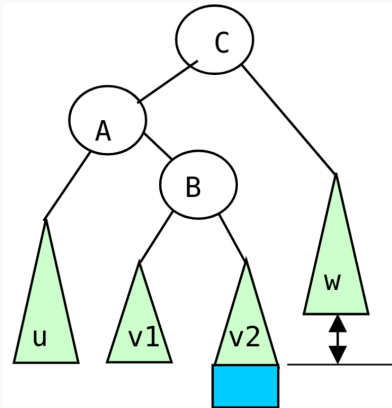
## Cas 2a

- Comment rééquilibrer ?

# Les cas de déséquilibre

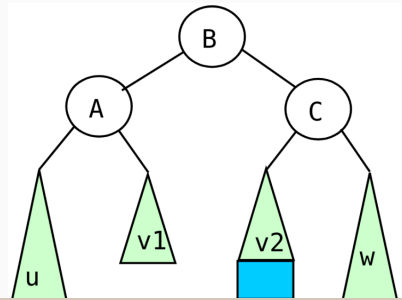
## Cas 2a

- $h(v1)=h(v2)$ ,  $h(u)=h(w)$ .
- déséquilibre en C après insertion dans v2



## Cas 2a

- Comment rééquilibrer ?
- ramène u, v2, w à la même hauteur (v1 pas tout à fait).
- v2 à droite de B (gauche de C)
- B à droite de A (gauche de C)
- v1 à droite de A (gauche de B)





# Les cas de déséquilibre

## Cas 2b (symétrique 2a)

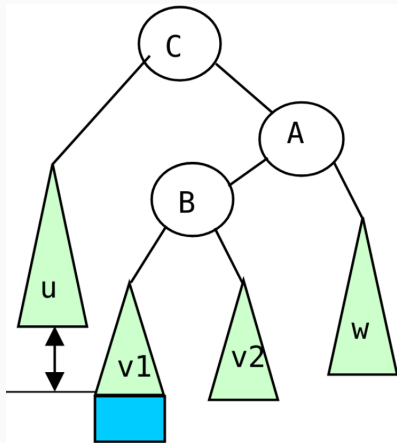


Figure 7 : Après insertion

## Cas 2b (symétrique 2a)

- Comment rééquilibrer ?

# Les cas de déséquilibre

Cas 2b (symétrique 2a)

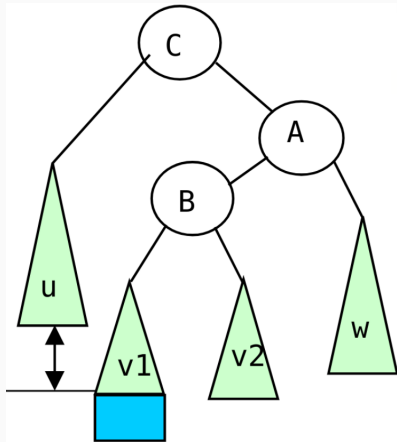


Figure 7 : Après insertion

Cas 2b (symétrique 2a)

- Comment rééquilibrer ?

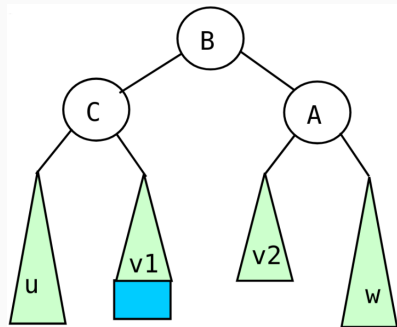


Figure 8 : Après équilibrage

# Le facteur d'équilibre (balance factor)

## Définition

$fe(arbre) = hauteur(droite(arbre)) - hauteur(gauche(arbre))$

**Valeurs possibles ?**

# Le facteur d'équilibre (balance factor)

## Définition

$fe(arbre) = hauteur(droite(arbre)) - hauteur(gauche(arbre))$

## Valeurs possibles ?

$fe = \{-1, 0, 1\}$  // arbre AVL

$fe = \{-2, 2\}$  // arbre déséquilibré

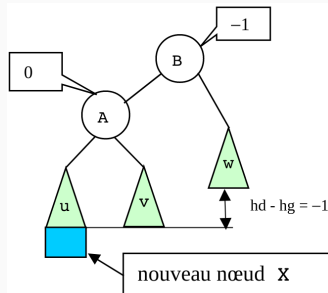


Figure 9 : Illustration du  $fe$

# Algorithme d'insertion

- Insérer le noeud comme d'habitude.
- Mettre à jour les facteurs d'équilibre jusqu'à la racine (ou au premier noeud déséquilibré).
- Rééquilibrer le noeud si nécessaire.

## Cas possibles

**Sous-arbre gauche (avant)**

$$fe(P) = 1$$

$$fe(P) = 0$$

$$fe(P) = -1$$

**Sous-arbre gauche (après)**

# Algorithme d'insertion

- Insérer le noeud comme d'habitude.
- Mettre à jour les facteurs d'équilibre jusqu'à la racine (ou au premier noeud déséquilibré).
- Rééquilibrer le noeud si nécessaire.

## Cas possibles

### Sous-arbre gauche (avant)

$$fe(P) = 1$$

$$fe(P) = 0$$

$$fe(P) = -1$$

### Sous-arbre gauche (après)

$$\Rightarrow fe(P) = 0$$

$$\Rightarrow fe(P) = -1$$

$$\Rightarrow fe(P) = -2 \text{ // Rééquilibrer P}$$

# Algorithme d'insertion

- Insérer le noeud comme d'habitude.
- Mettre à jour les facteurs d'équilibre jusqu'à la racine (ou au premier noeud déséquilibré).
- Rééquilibrer le noeud si nécessaire.

## Cas possibles

### Sous-arbre droit (avant)

$$fe(P) = 1$$

$$fe(P) = 0$$

$$fe(P) = -1$$

### Sous-arbre droit (après)

# Algorithme d'insertion

- Insérer le noeud comme d'habitude.
- Mettre à jour les facteurs d'équilibre jusqu'à la racine (ou au premier noeud déséquilibré).
- Rééquilibrer le noeud si nécessaire.

## Cas possibles

### Sous-arbre droit (avant)

$$fe(P) = 1$$

$$fe(P) = 0$$

$$fe(P) = -1$$

### Sous-arbre droit (après)

$$\Rightarrow fe(P) = 0$$

$$\Rightarrow fe(P) = +1$$

$$\Rightarrow fe(P) = +2 \text{ // Rééquilibrer P}$$



# Rééquilibrage

## Lien avec les cas vus plus tôt

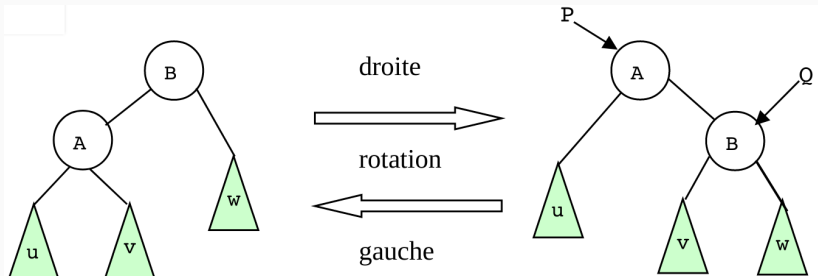
$fe(P) = -2 \ \&\& \ fe(gauche(P)) = -1 \Rightarrow$  cas 1a

$fe(P) = -2 \ \&\& \ fe(gauche(P)) = +1 \Rightarrow$  cas 2a

$fe(P) = +2 \ \&\& \ fe(droite(P)) = -1 \Rightarrow$  cas 2b

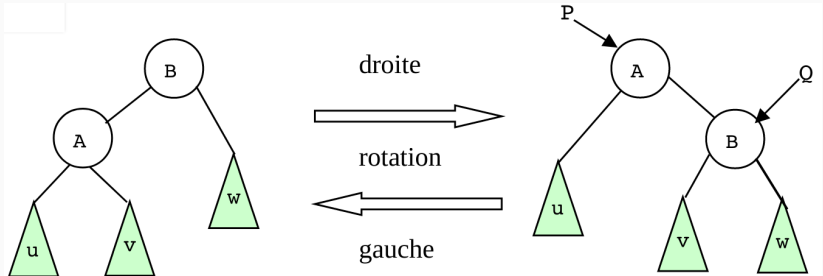
$fe(P) = +2 \ \&\& \ fe(droite(P)) = +1 \Rightarrow$  cas 1b

Dessiner les différents cas, sur le dessin ci-dessous



# La rotation

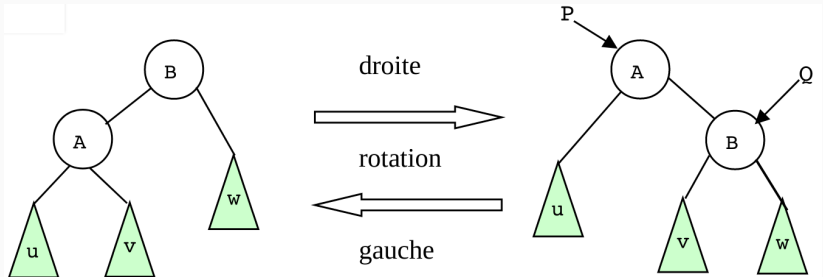
## La rotation gauche (5min, matrix)



**Figure 11** : L'arbre de droite devient celui de gauche. Comment ?

# La rotation

## La rotation gauche (5min, matrix)



**Figure 11** : L'arbre de droite devient celui de gauche. Comment ?

```
arbre rotation_gauche(arbre P)
  si est_non_vide(P)
    Q = droite(P)
    droite(P) = gauche(Q)
    gauche(Q) = P
    retourne Q
```

# La rotation en C (1/2)

## La rotation gauche

```
arbre rotation_gauche(arbre P)
    si est_non_vide(P)
        Q = droite(P)
        droite(P) = gauche(Q)
        gauche(Q) = P
        retourne Q
    retourne P
```

## Écrire le code C correspondant (5min, matrix)

1. Structure de données
2. Fonction `tree_t rotation_left(tree_t tree)`

# La rotation en C (1/2)

## La rotation gauche

```
arbre rotation_gauche(arbre P)
    si est_non_vide(P)
        Q = droite(P)
        droite(P) = gauche(Q)
        gauche(Q) = P
        retourne Q
    retourne P
```

## Écrire le code C correspondant (5min, matrix)

1. Structure de données
2. Fonction `tree_t rotation_left(tree_t tree)`

```
typedef struct _node {
    int key;
    struct _node *left, *right;
    int bf; // balance factor
} node;
```

## La rotation en C (2/2)

```
node* rotation_left(node* tree) {  
    node* subtree = NULL;  
    if (NULL != tree) {  
        subtree = tree->right;  
        tree->right = subtree->left;  
        subtree->left = tree;  
    }  
    return subtree;  
}
```

## La rotation en C (2/2)

```
node* rotation_left(node* tree) {  
    node* subtree = NULL;  
    if (NULL != tree) {  
        subtree = tree->right;  
        tree->right = subtree->left;  
        subtree->left = tree;  
    }  
    return subtree;  
}
```

- Et la rotation à droite, pseudo-code (5min) ?

## La rotation en C (2/2)

```
node* rotation_left(node* tree) {  
    node* subtree = NULL;  
    if (NULL != tree) {  
        subtree = tree->right;  
        tree->right = subtree->left;  
        subtree->left = tree;  
    }  
    return subtree;  
}
```

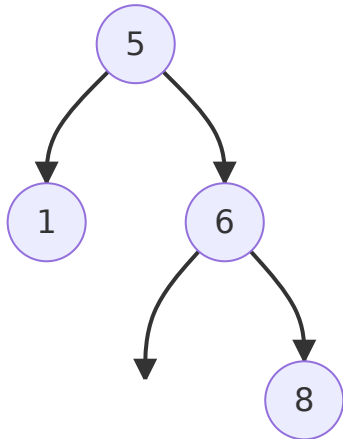
- Et la rotation à droite, pseudo-code (5min) ?

```
arbre rotation_droite(arbre P)  
    si est_non_vide(P)  
        Q = gauche(P)  
        gauche(P) = droite(Q)  
        droite(Q) = P  
        retourne Q  
    retourne P
```



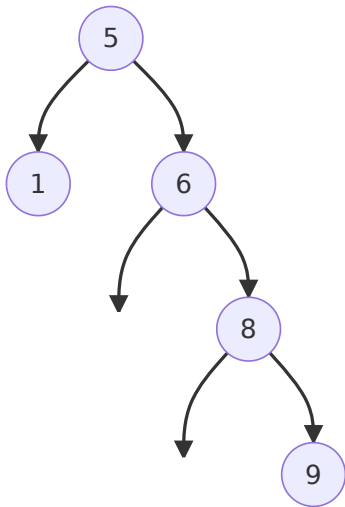
## Exemple de rotation (1/2)

Insertion de 9 ?



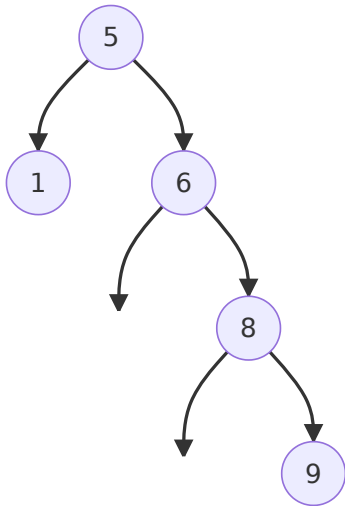
## Exemple de rotation (2/2)

Quelle rotation et sur quel noeud (5 ou 6) ?

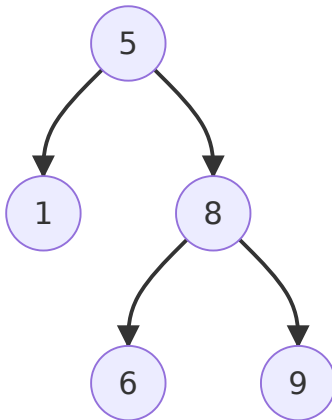


## Exemple de rotation (2/2)

Quelle rotation et sur quel noeud (5 ou 6) ?

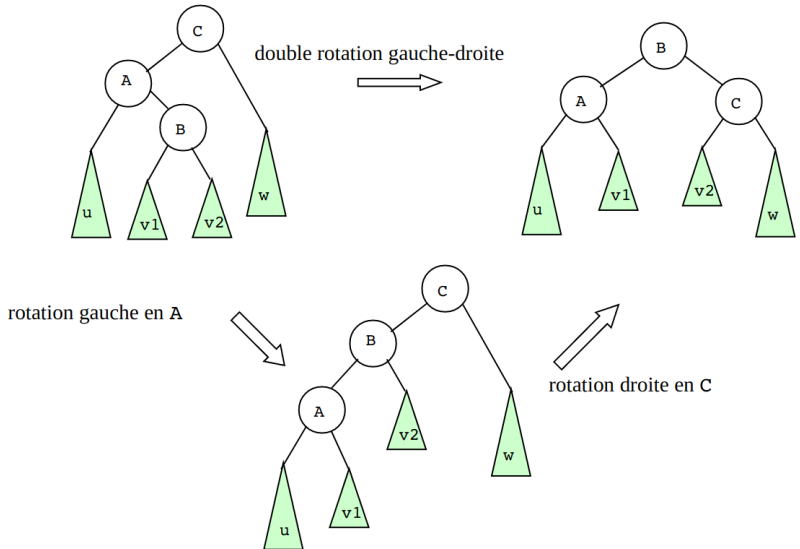


Sur le plus jeune évidemment !



# La rotation gauche-droite

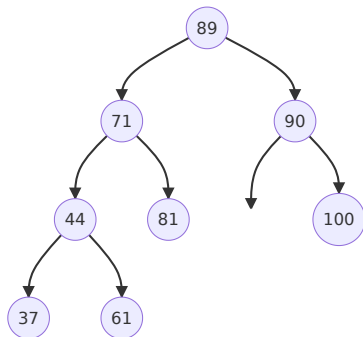
Là c'est plus difficile (cas 2a/b)



**Faire l'implémentation de la double rotation (pas corrigé, 5min)**

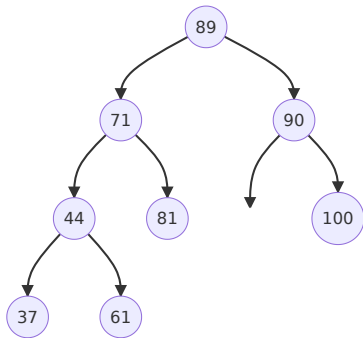
# Exercices

Insérer 50, ex 10min (matrix)

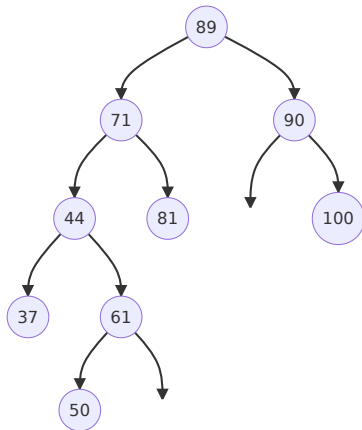


# Exercices

Insérer 50, ex 10min (matrix)

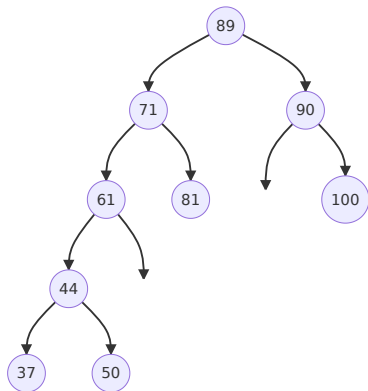


Où se fait la rotation ?



# Exercices

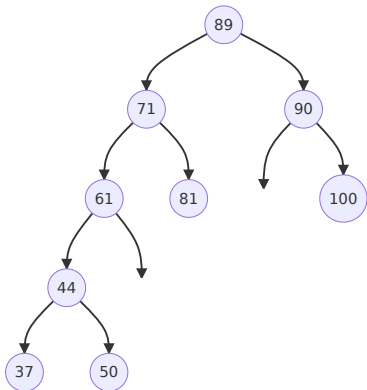
## Rotation gauche en 44



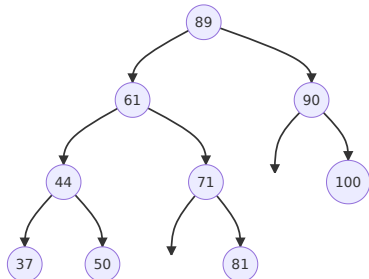


# Exercices

## Rotation gauche en 44

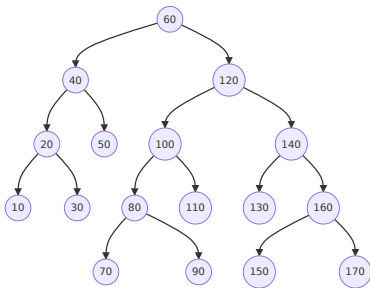


## Rotation à droite en 71



# Exercice de la mort

Soit l'arbre AVL suivant :



1. Montrer les positions des insertions de feuille qui conduiront à un arbre déséquilibré.
2. Donner les facteurs d'équilibre.
3. Dessiner et expliquer les modifications de l'arbre lors de l'insertion de la valeur 65. On mentionnera les modifications des facteurs d'équilibre.

## Encore un petit exercice

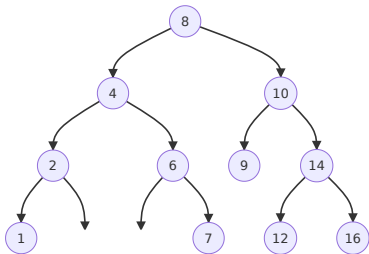
- Insérer les nœuds suivants dans un arbre AVL

25 | 60 | 35 | 10 | 5 | 20 | 65 | 45 | 70 | 40  
| 50 | 55 | 30 | 15

**Un à un et le/la premier/ère qui poste la bonne réponse sur matrix a un point**

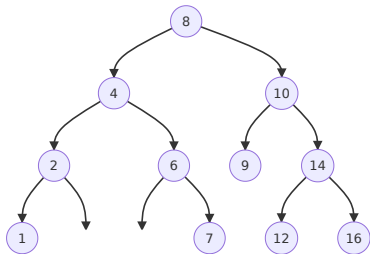
# Suppression dans un arbre AVL

Algorithme par problème :  
supprimer 10

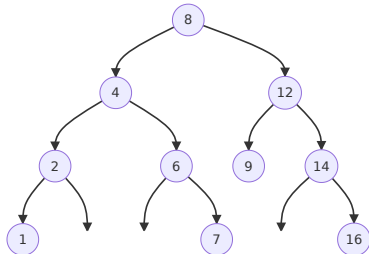


# Suppression dans un arbre AVL

Algorithme par problème :  
supprimer 10

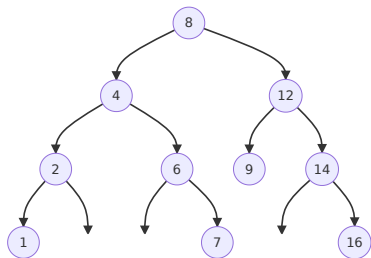


Algorithme par problème :  
supprimer 10



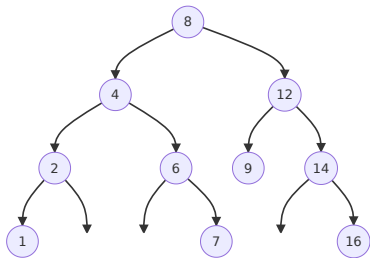
# Suppression dans un arbre AVL

Algorithme par problème :  
supprimer 8

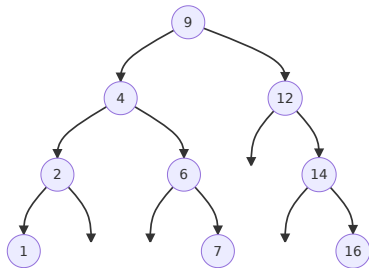


# Suppression dans un arbre AVL

Algorithme par problème :  
supprimer 8

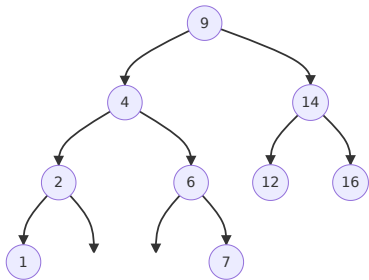


Algorithme par problème :  
rotation de 12



# Suppression dans un arbre AVL

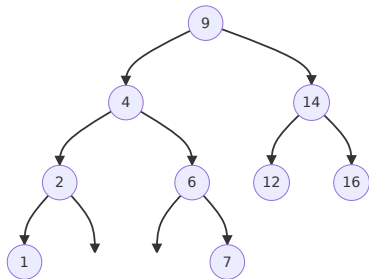
Algorithme par problème :  
rotation de 12





# Suppression dans un arbre AVL

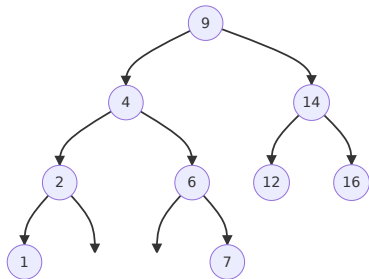
**Algorithme par problème :**  
**rotation de 12**



1. On supprime comme d'habitude.
2. On rééquilibre si besoin à l'endroit de la suppression.
  - Facile non ?

# Suppression dans un arbre AVL

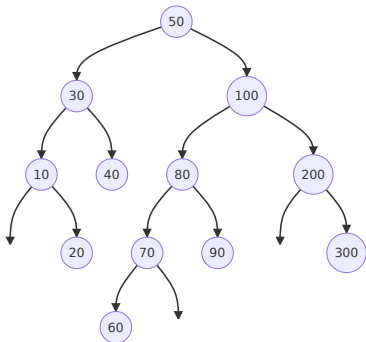
**Algorithme par problème :**  
**rotation de 12**



1. On supprime comme d'habitude.
2. On rééquilibre si besoin à l'endroit de la suppression.
  - Facile non ?
  - Plus dur....

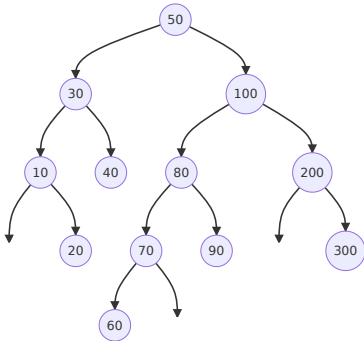
# Suppression dans un arbre AVL 2.0

Algorithme par problème :  
suppression de 30

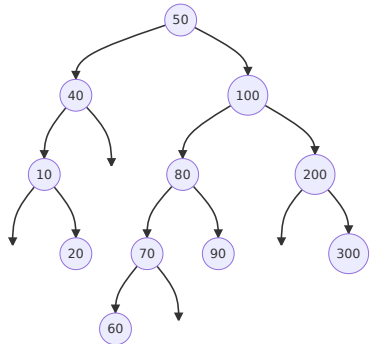


# Suppression dans un arbre AVL 2.0

Algorithme par problème :  
suppression de 30

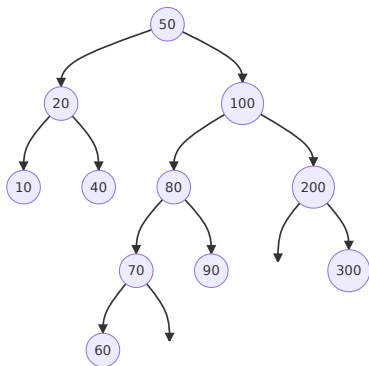


Algorithme par problème :  
rotation GD autour de 40



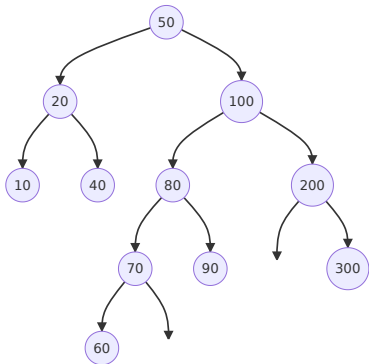
# Suppression dans un arbre AVL 2.0

**Arg! 50 est déséquilibré !**

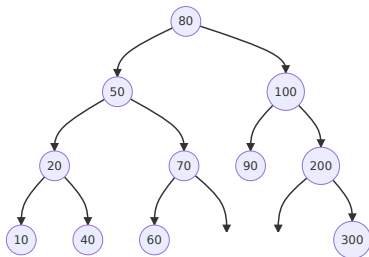


# Suppression dans un arbre AVL 2.0

Arg! 50 est déséquilibré !



Algorithme par problème :  
rotation DG autour de 50



# Résumé de la suppression

1. On supprime comme pour un arbre binaire de recherche.
2. Si un nœud est déséquilibré, on le rééquilibre.
  - Cette opération peut déséquilibrer un autre nœud.
3. On continue à rééquilibrer tant qu'il y a des nœuds à équilibrer.

## Un petit exercice

- Insérer les nœuds suivants dans un arbre AVL

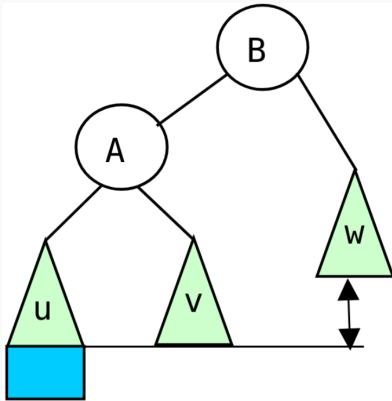
25 | 60 | 35 | 10 | 5 | 20 | 65 | 45 | 70 | 40  
| 50 | 55 | 30 | 15



# Rappel : les cas de déséquilibre

## Cas 1a

- $u$ ,  $v$ ,  $w$  même hauteur.
- déséquilibre en B après insertion dans  $u$



## Cas 1a

- Comment rééquilibrer ?

Figure 13 : Après insertion

# Rappel : les cas de déséquilibre

## Cas 1a

- u, v, w même hauteur.
- déséquilibre en B après insertion dans u

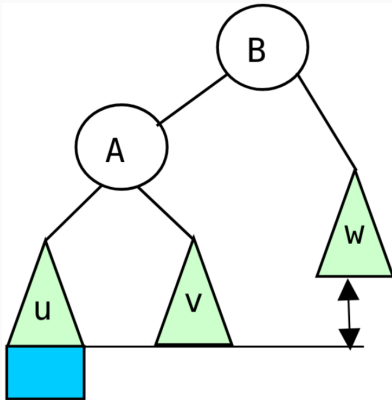


Figure 13 : Après insertion

## Cas 1a

- Comment rééquilibrer ?
- ramène u, v w à la même hauteur.
- v à droite de A (gauche de B)

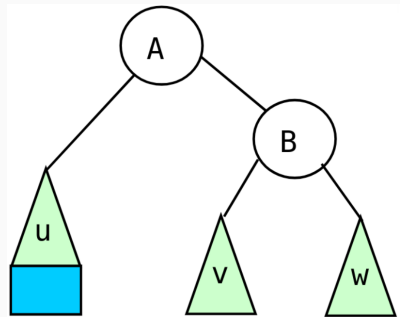
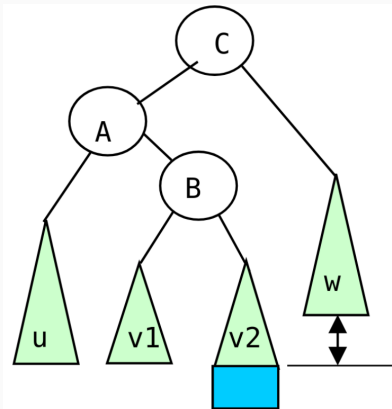


Figure 14 : Après équilibrage

# Rappel : Les cas de déséquilibre

## Cas 2a

- $h(v1)=h(v2)$ ,  $h(u)=h(w)$ .
- déséquilibre en C après insertion dans v2



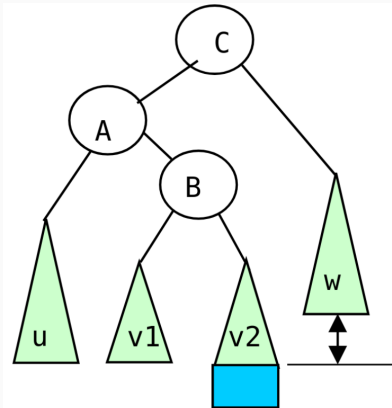
## Cas 2a

- Comment rééquilibrer ?

# Rappel : Les cas de déséquilibre

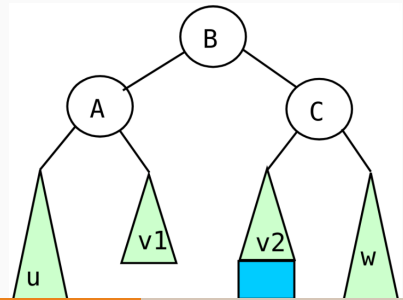
## Cas 2a

- $h(v1)=h(v2)$ ,  $h(u)=h(w)$ .
- déséquilibre en C après insertion dans v2



## Cas 2a

- Comment rééquilibrer ?
- ramène u, v2, w à la même hauteur (v1 pas tout à fait).
- v2 à droite de B (gauche de C)
- B à droite de A (gauche de C)
- v1 à droite de A (gauche de B)



# Rappel : Rééquilibrage

## Rotation simple

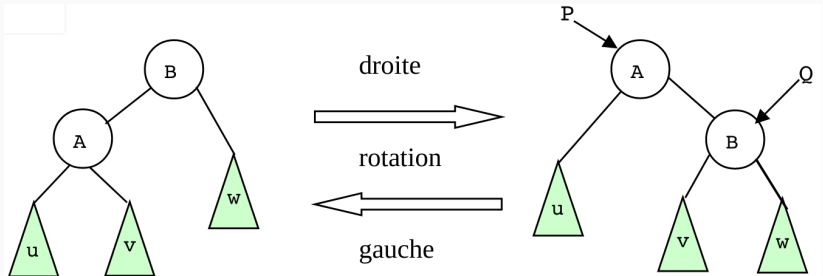
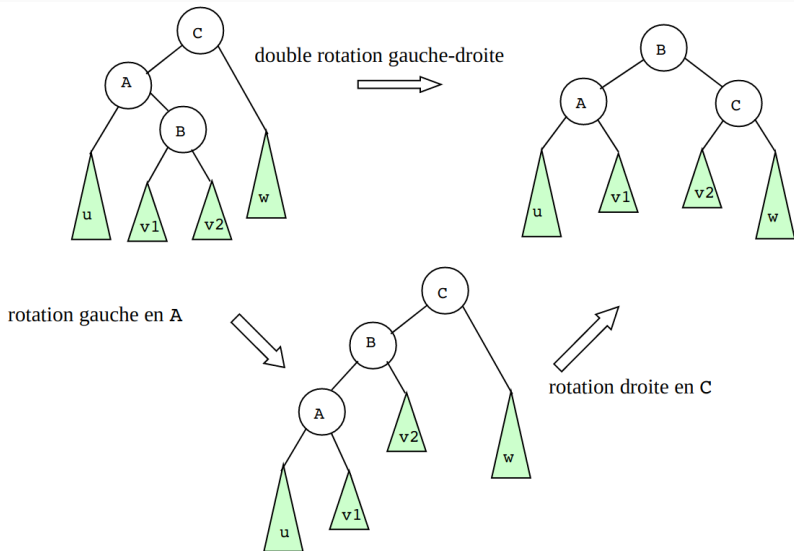


Figure 17 : On verra un peu après les rotations.

# Rappel : La rotation gauche-droite

## Le cas 2a/b



## Un petit exercice

- Insérer les nœuds suivants dans un arbre AVL

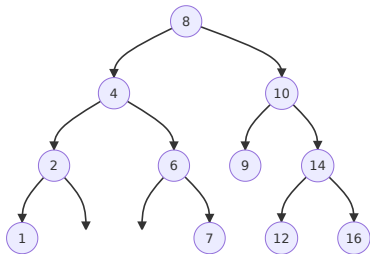
25 | 60 | 35 | 10 | 5 | 20 | 65 | 45 | 70 | 40  
| 50 | 55 | 30 | 15

# La suppression dans un arbre AVL



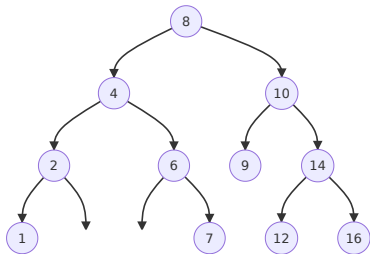
# Suppression dans un arbre AVL

Algorithme par problème :  
supprimer 10

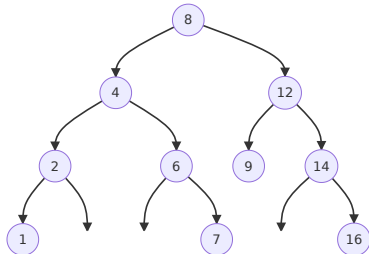


# Suppression dans un arbre AVL

Algorithme par problème :  
supprimer 10

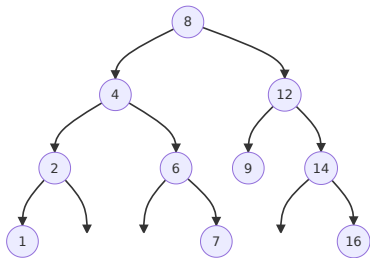


Algorithme par problème :  
supprimer 10



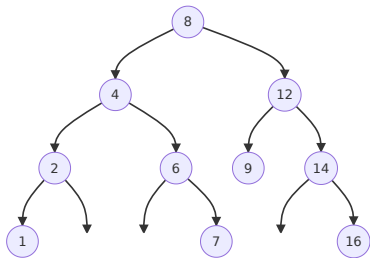
# Suppression dans un arbre AVL

Algorithme par problème :  
supprimer 8

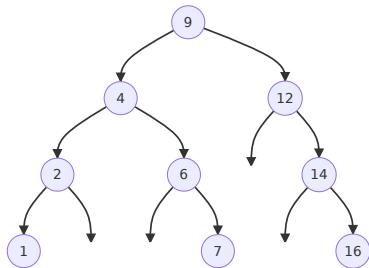


# Suppression dans un arbre AVL

Algorithme par problème :  
supprimer 8

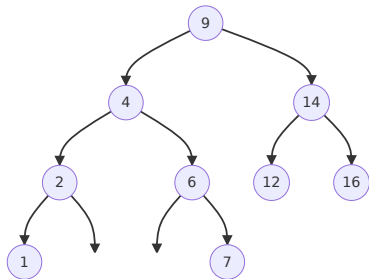


Algorithme par problème :  
rotation de 12



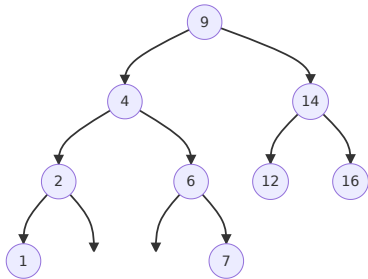
# Suppression dans un arbre AVL

Algorithme par problème :  
rotation de 12



# Suppression dans un arbre AVL

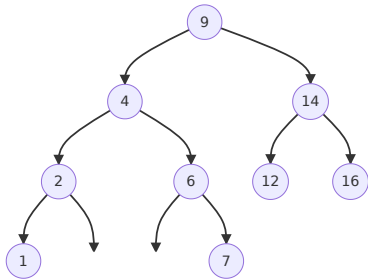
**Algorithme par problème :**  
**rotation de 12**



1. On supprime comme d'habitude.
2. On rééquilibre si besoin à l'endroit de la suppression.
  - Facile non ?

# Suppression dans un arbre AVL

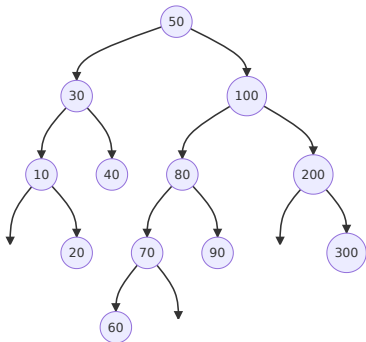
**Algorithme par problème :**  
**rotation de 12**



1. On supprime comme d'habitude.
2. On rééquilibre si besoin à l'endroit de la suppression.
  - Facile non ?
  - Plus dur....

# Suppression dans un arbre AVL 2.0

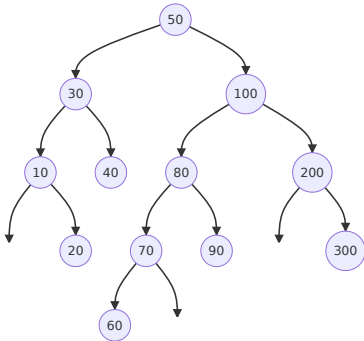
Algorithme par problème :  
suppression de 30



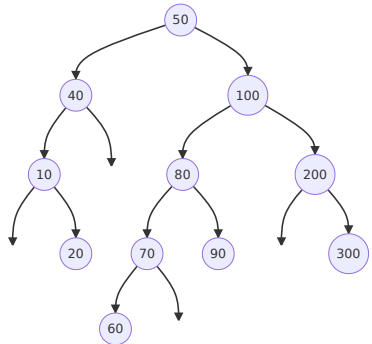


# Suppression dans un arbre AVL 2.0

Algorithme par problème :  
suppression de 30

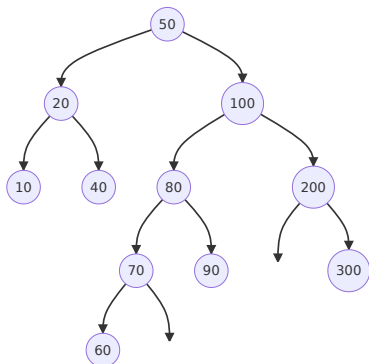


Algorithme par problème :  
rotation GD autour de 40



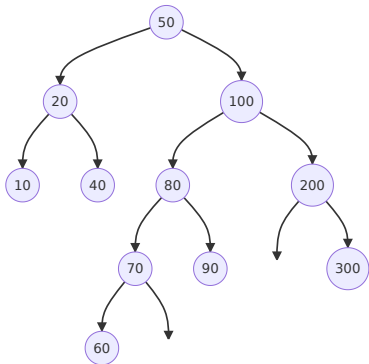
# Suppression dans un arbre AVL 2.0

Arg! 50 est déséquilibré !

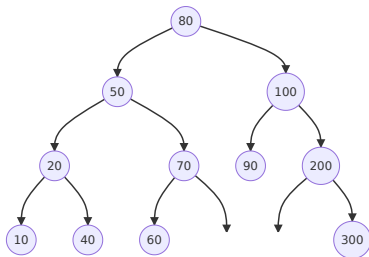


# Suppression dans un arbre AVL 2.0

Arg! 50 est déséquilibré !



Algorithme par problème :  
rotation DG autour de 50



# Résumé de la suppression

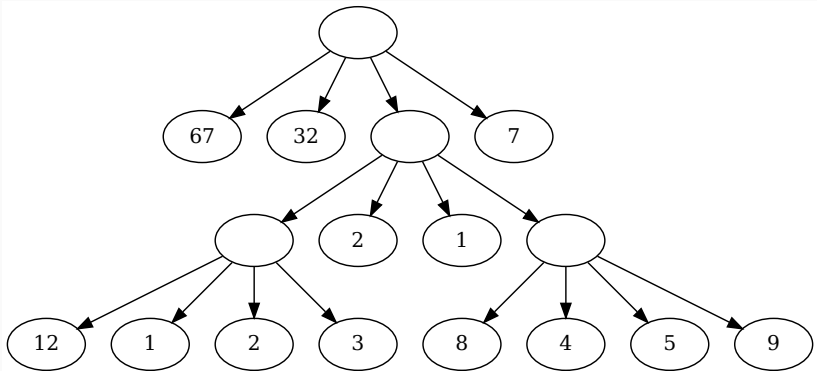
1. On supprime comme pour un arbre binaire de recherche.
2. Si un nœud est déséquilibré, on le rééquilibre.
  - Cette opération peut déséquilibrer un autre nœud sur le chemin menant au nœud supprimé.
3. On continue à rééquilibrer tant qu'il y a des nœuds à équilibrer en remontant le chemin. »»»> main

# Les arbres quaternaires

# Les arbres quaternaires

## Définition

Arbre dont chaque nœud a 4 enfants ou aucun.



**Figure 19** : Un exemple d'arbre quaternaire.

# Les arbres quaternaires

## Cas d'utilisation

Typiquement utilisés pour représenter des données bidimensionnelles.

Son équivalent tri-dimensionnel est l'octree (chaque nœud a 8 enfants ou aucun).

## Cas d'utilisation : images

- Stockage : compression.
- Transformations : symétries, rotations, etc.

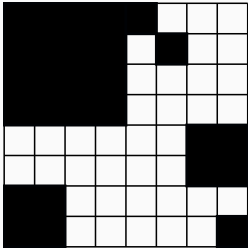
## Cas d'utilisation : simulation

- Indexation spatiale.
- Détection de collisions.
- Simulation de galaxies (algorithme de Barnes-Hut).

# Exemple de compression

Comment représen-  
ter l'image

Sous la forme d'un arbre quaternaire ?

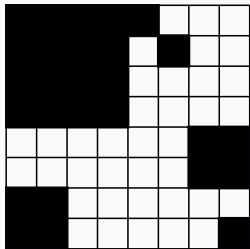


**Figure 20 :** Image  
noir/blanc.



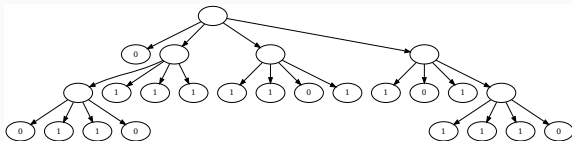
# Exemple de compression

Comment représenter l'image



**Figure 20 :** Image noir/blanc.

Sous la forme d'un arbre quaternaire ?

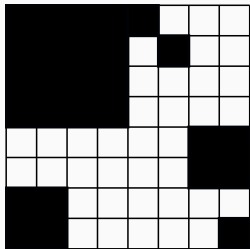


**Figure 21 :** L'arbre quaternaire correspondant.

Économie ?

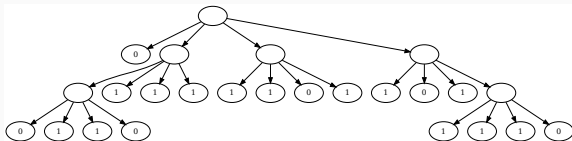
## Exemple de compression

## Comment représen- ter l'image



**Figure 20 :** Image noir/blanc.

### Sous la forme d'un arbre quaternaire ?



**Figure 21 :** L'arbre quaternaire correspondant.

## Économie ?

Image 64 pixels, arbre 25 nœuds.

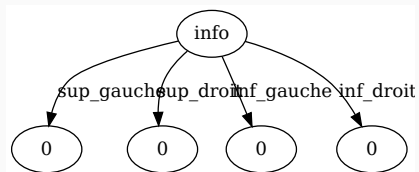
Pseudo-code ?

# Structure de données

Pseudo-code ?

```
struct node
    info
    node sup_gauche, sup_droit,
        inf_gauche, inf_droit
```

En C ?

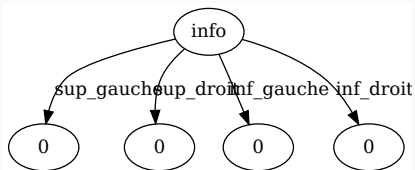


**Figure 22 :** Un nœud d'arbre quaternaire.

# Structure de données

## Pseudo-code ?

```
struct node
    info
    node sup_gauche, sup_droit,
        inf_gauche, inf_droit
```



**Figure 22 :** Un nœud d'arbre quaternaire.

## En C ?

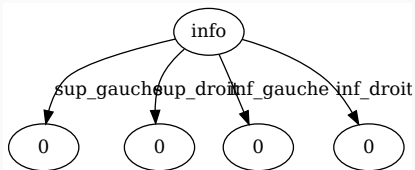
```
struct _node {
    int info;
    struct _node *sup_left;
    struct _node *sup_right;
    struct _node *inf_left;
    struct _node *inf_right;
};
```

- Pourquoi le \* est important ?

# Structure de données

## Pseudo-code ?

```
struct node
    info
    node sup_gauche, sup_droit,
        inf_gauche, inf_droit
```



**Figure 22 :** Un nœud d'arbre quaternaire.

## En C ?

```
struct _node {
    int info;
    struct _node *sup_left;
    struct _node *sup_right;
    struct _node *inf_left;
    struct _node *inf_right;
};
```

- Pourquoi le \* est important ?
- Type récursif => taille inconnue à la compilation.