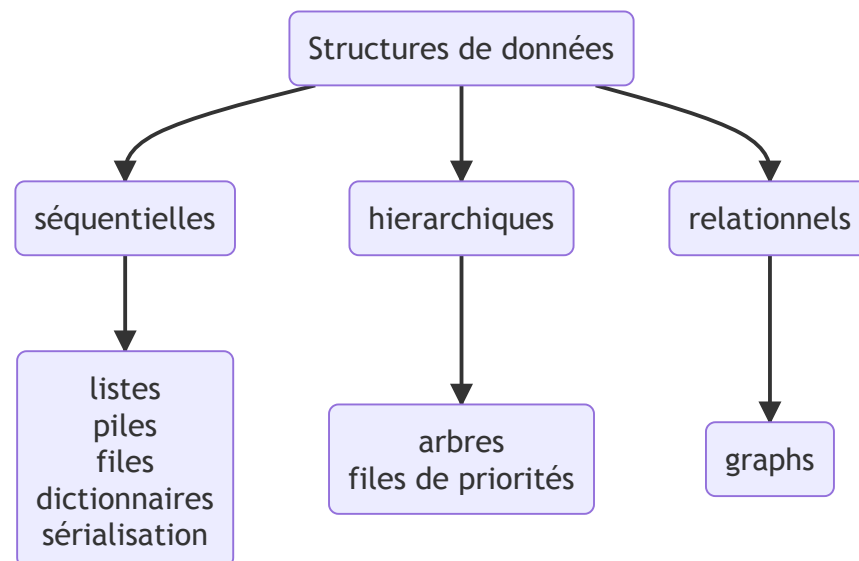


I. Introduction - définition

Utilisation des arbres

- Arbres syntaxiques
- Arbre lexicographique
- Arbre de décision / classification (ML)
- Compression de données
- Expressions mathématiques

Classification



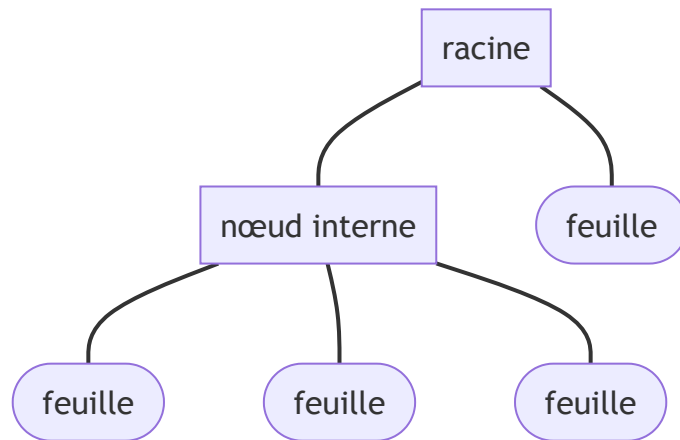
Définitions

Un arbre (enraciné) est

- soit un ensemble vide
- soit un ensemble fini non vide A muni d'une relation binaire $<$ (est le fils de ...) telle que
- il existe $r \in A$ tel que $\forall x \in A, r < x$ (il existe un ancêtre)
- $\forall x \in A \setminus \{r\}, \exists! y \in A, x < y$ (il y a un père unique)
- $\forall x \in A \setminus \{r\}, \exists n \in \mathbb{N}$ et $(x_1, x_2, \dots, x_n) \in A^n, x < x_1 < x_2 < \dots < x_n < r$ (chaque élément descend de l'ancêtre)

On parle aussi de frères, de descendance et d'ancêtres.

L'*arité* d'un père correspond au nombre de ses fils

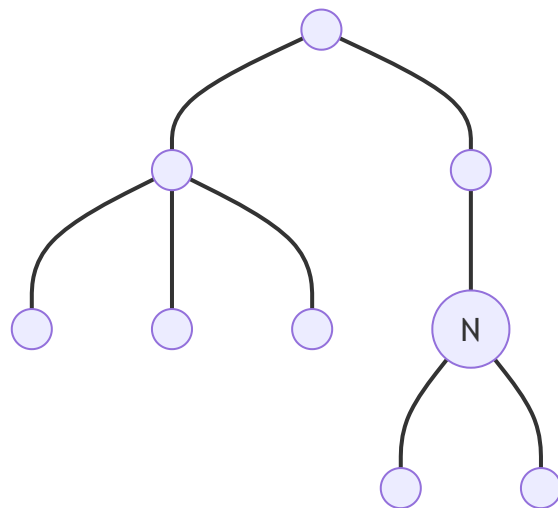


Un *arbre n -aire* est un arbre dont les nœuds sont d'arité maximale n . Dans l'exemple, chaque nœud a au maximum une arité de 3 donc l'arbre est un arbre ternaire.

La *taille* de l'arbre est le nombre de nœuds qui le compose. Dans l'exemple, la taille est 6.

La *profondeur* d'un nœud est:

- -1 si l'arbre est vide
- 0 pour la racine
- le nombre de nœuds depuis la racine avant d'atteindre le nœud



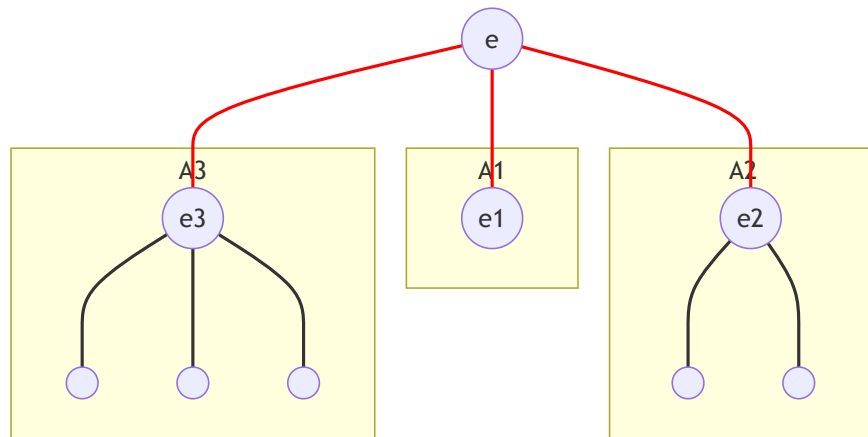
Par exemple, le nœud N a une profondeur de 2.

La *hauteur* de l'arbre est la profondeur maximale de ses feuilles. Dans l'exemple, la hauteur est de 3.

Définition inductive

Soit A un ensemble fini appelé nœuds.

- les arbres de hauteur 0 sont les éléments de A noté $(e, 0)$ où e est la racine de l'arbre
- Si $e \in A$ et Si A_1, A_2, \dots, A_n sont des arbres de hauteur respectives h_1, h_2, \dots, h_n et dont les racines respectives sont e_1, e_2, \dots, e_n , alors en connectant e à e_1, e_2, \dots, e_n , on définit $(e, (A_1, A_2, \dots, A_n))$ l'arbre de racine e , de hauteur $1 + \max_{k \in [1, n]} (h_k)$ de sous arbres A_1, A_2, \dots, A_n



Remarques:

Arbre marqué: Chaque nœud associé à une étiquette / valeur

Arbre non marqué: Les nœuds n'ont pas de valeurs associés

Un arbre est un graph:

- Simple: pas de boucles ou d'arêtes multiples
- Non orienté: les parcours $a \rightarrow b$ et $b \rightarrow a$ sont toujours possible
- Acyclique: pas de "boucles"
- Connexe: tous les nœuds sont accessibles

Propriété:

Pour un arbre de hauteur h d'arité $a > 1$, le nombre n de nœuds vérifie

$$h + 1 \geq n \geq \frac{a^{h+1} - 1}{a - 1}$$

Preuve:

Le nombre minimal de nœuds pour une hauteur donnée est un arbre contenant 1 nœud par hauteur. Le nombre maximal de nœuds pour une hauteur donnée est un arbre contenant i nœud par niveau i .

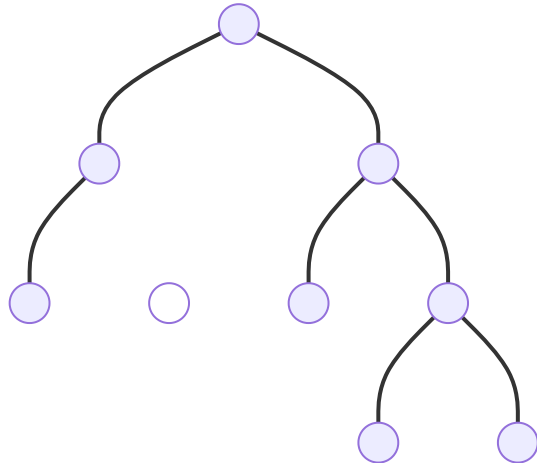
$$\sum_{i=0}^h 1 \geq n \geq \sum_{i=0}^h a^i$$

$$\Leftrightarrow h + 1 \geq n \geq \frac{a^{h+1} - 1}{a - 1}$$

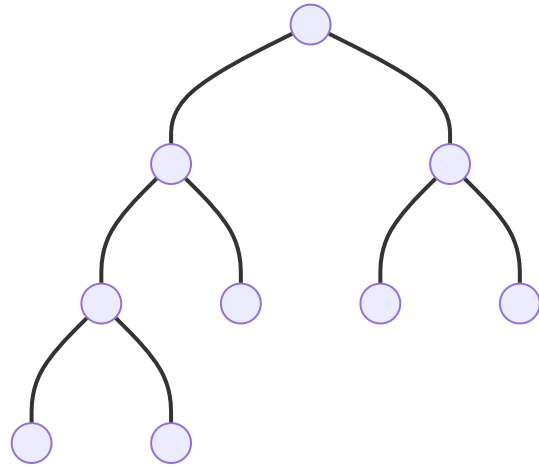
II. Arbres particuliers

Définitions**Arbres binaires:**

Tous les nœuds sont d'arité au maximum 2.

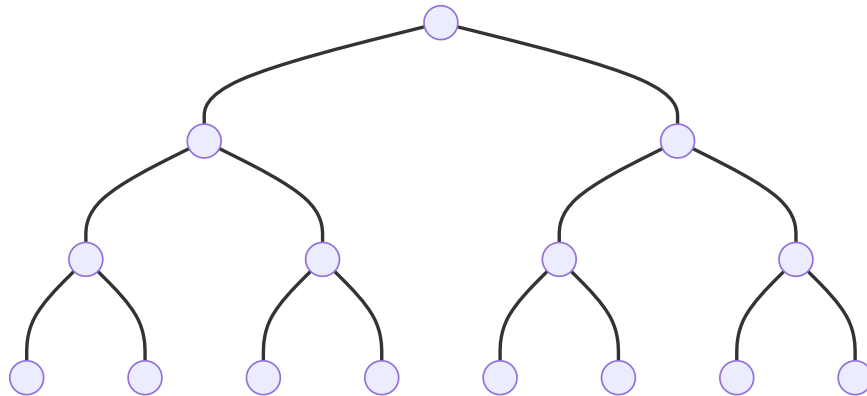
**Arbre binaire entier:**

Tous les nœuds sont d'arité 2



Arbre binaire complet

Arbre binaire dont les feuilles ont toutes la même profondeur. Il possède le nombre maximal de feuilles pour une hauteur donnée. On parle donc de fils gauche et de fils droit.



Propriétés

Nombre de nœuds $n \leq 2^p$ (pour un niveau p)

Hauteur h : $\lfloor \log_2(n) \rfloor < h \leq n - 1$

Arbre binaire à i nœuds interne (et la racine) d'arité 2 et f feuilles : $f = i + 1$

Nombre de feuilles $f \leq 2^h$

Profondeur maximale d'un arbre d'arité a composée de n nœuds:

$$\log_a((a-1) \times n + 1) - 1 \leq h \leq n - 1$$

Remarque:

$$\begin{aligned} & \log_a((a-1) \times n + 1) - 1 \leq h \\ \Rightarrow & \log_a((a-1) \times n) < \log_a((a-1) \times n + 1) \leq h + 1 \\ \Rightarrow & \lfloor \log_a((a-1) \times n) \rfloor < h + 1 \\ \Rightarrow & \lfloor \log_a((a-1) \times n) \rfloor \leq h \end{aligned}$$

soit $\lfloor \log_a((a-1) \times n) \rfloor \leq h \leq n - 1$

Transformation d'un arbre en arbre binaire

Algorithme de transformation d'un arbre n -aire A en arbre binaire:

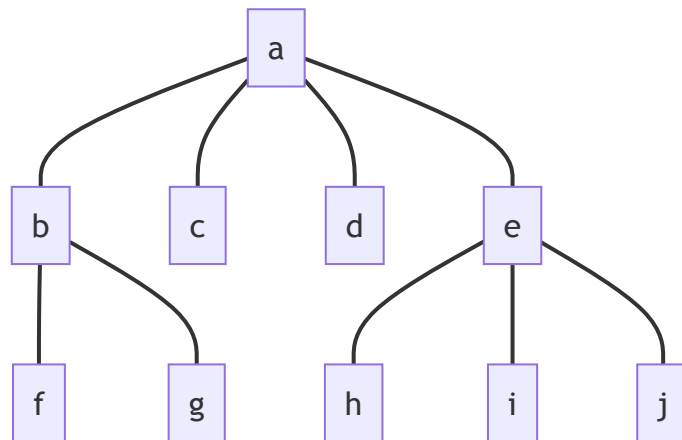
Pour chaque $e \in A$

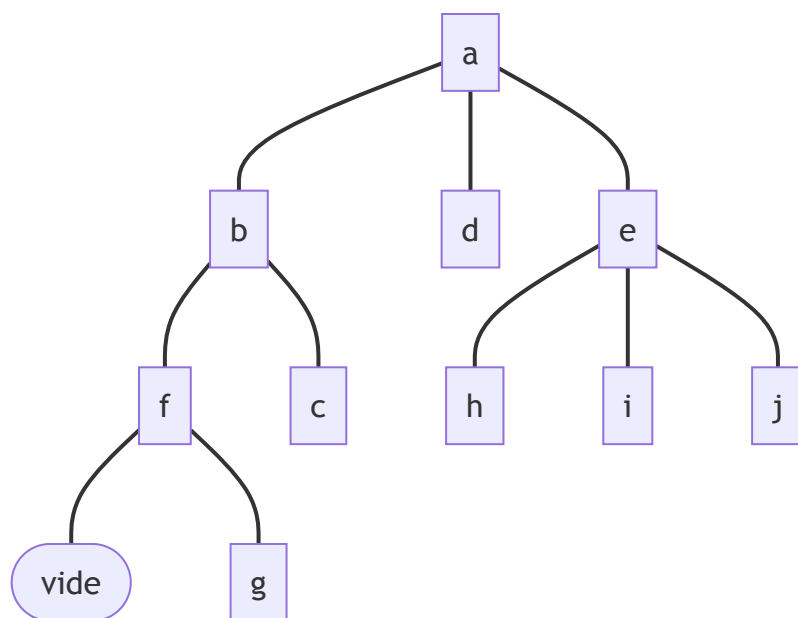
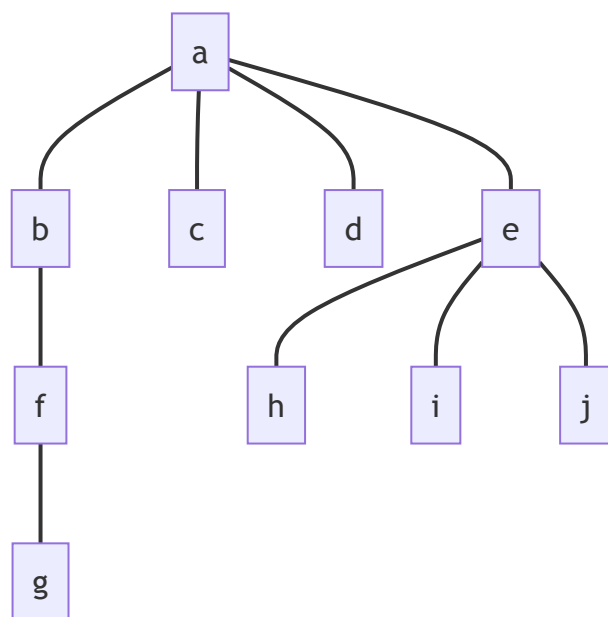
Laisser e_g (fils le plus à gauche) en place

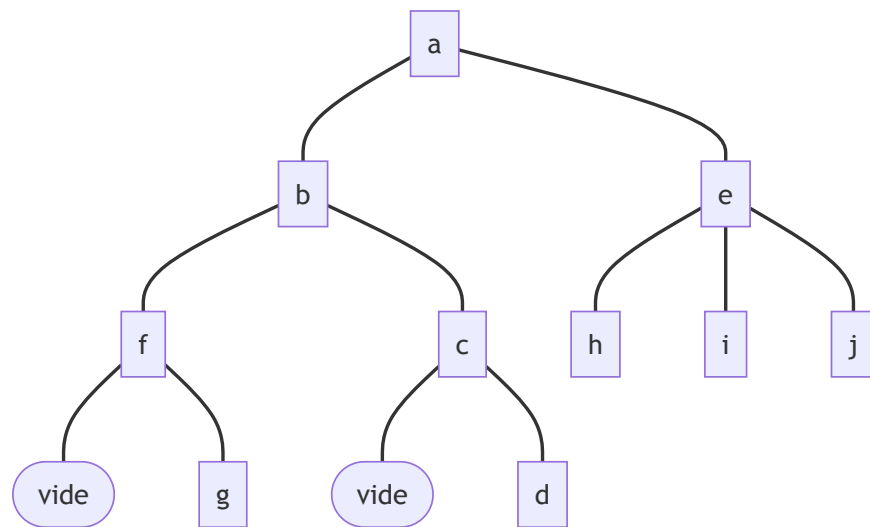
Supprimer les autres fils de e et les chaîner à e_g

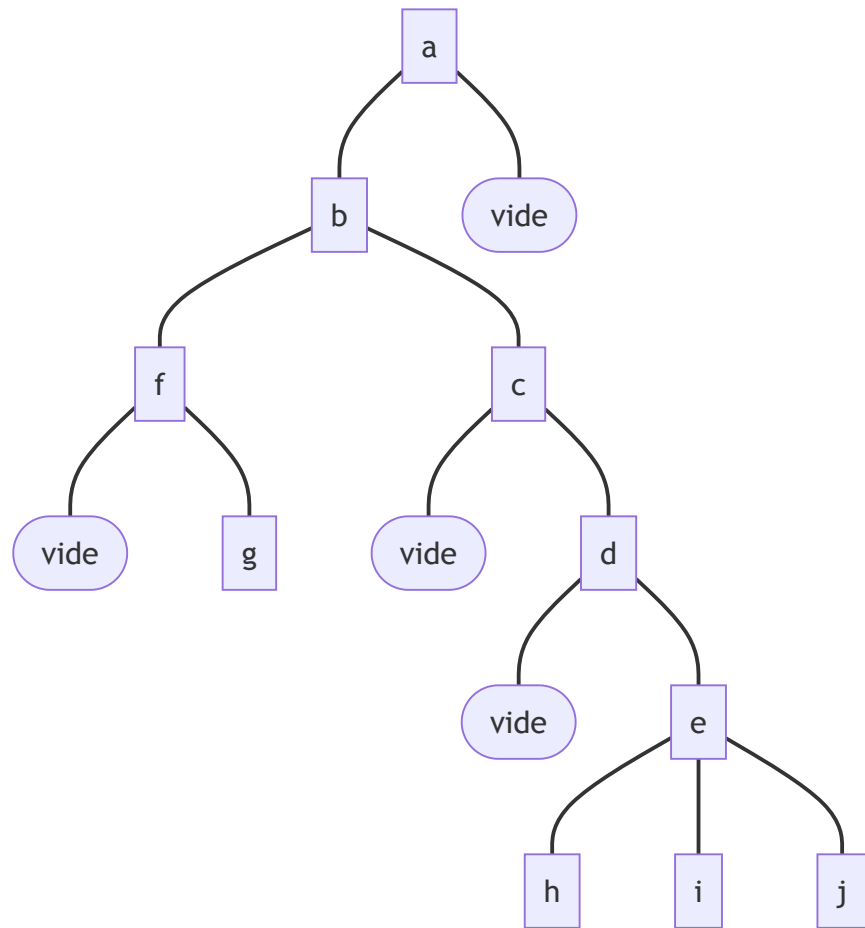
Le nœud e_d est le prochain frère de e

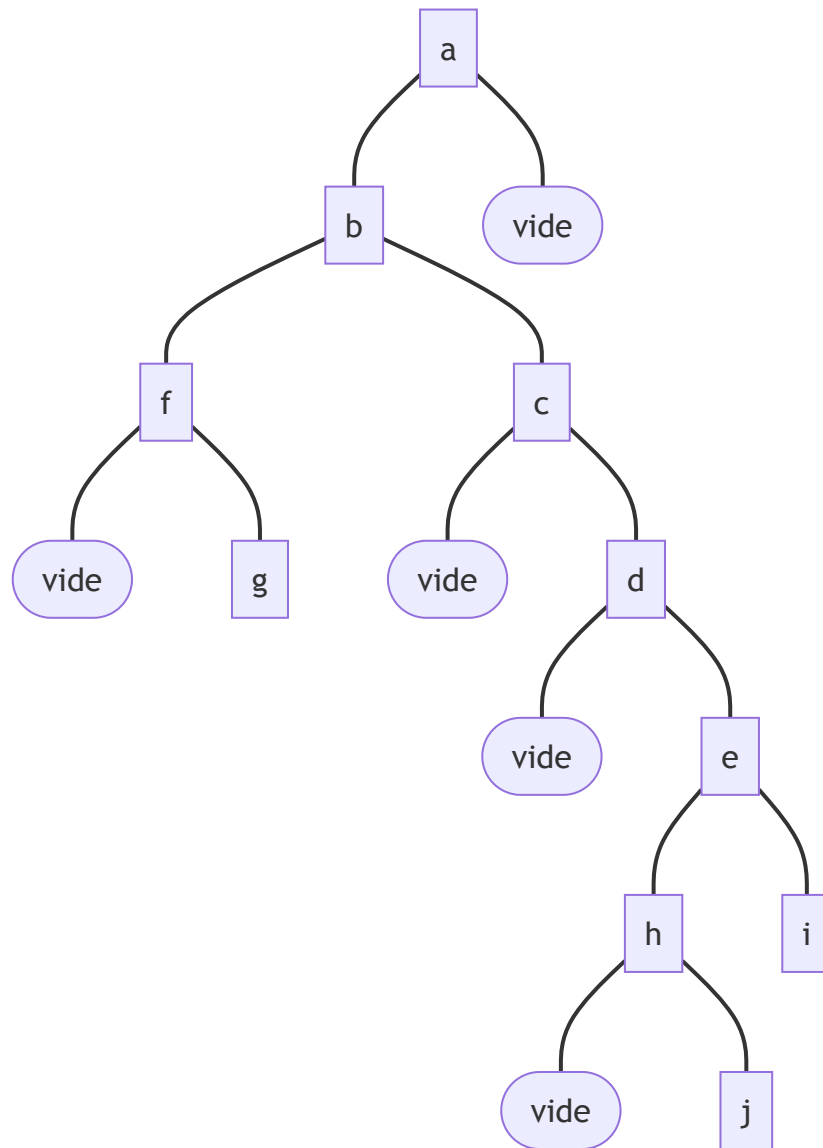
Fin pour

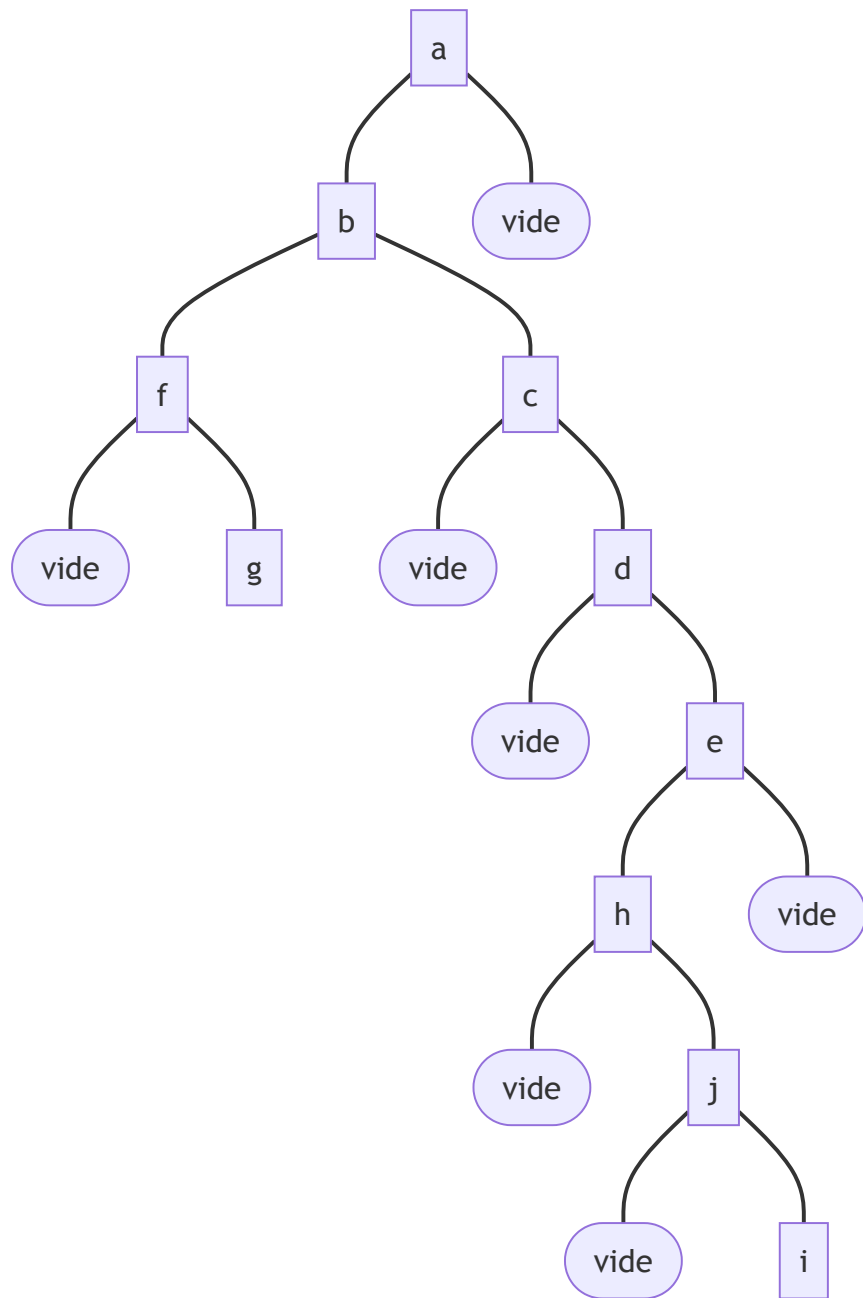




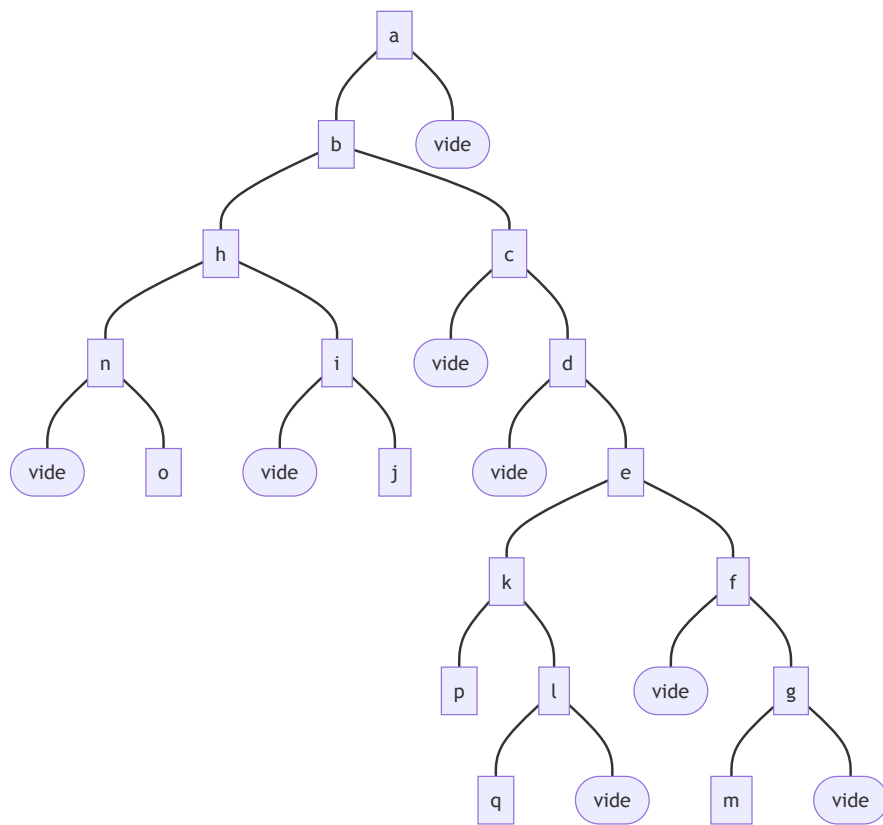
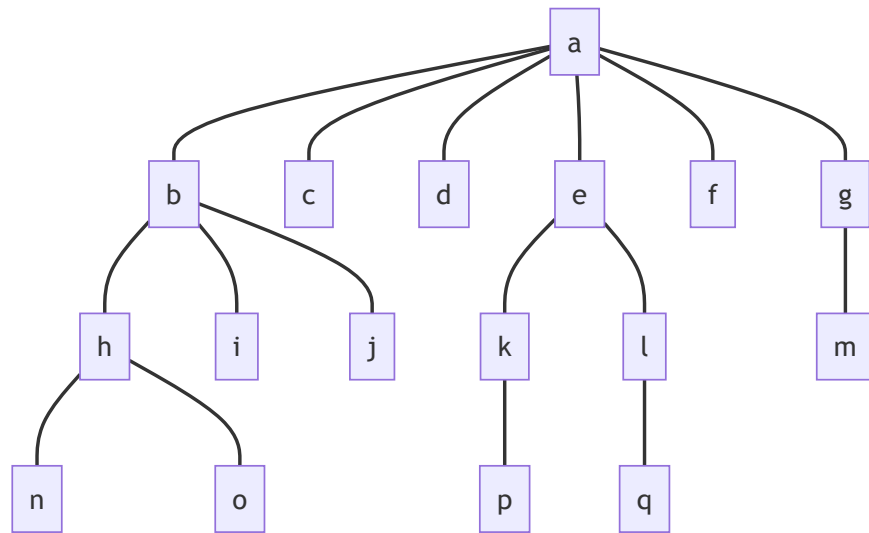








Exercise:



Arbres binaires : implémentation sous forme de tableau en C (sérialisation)

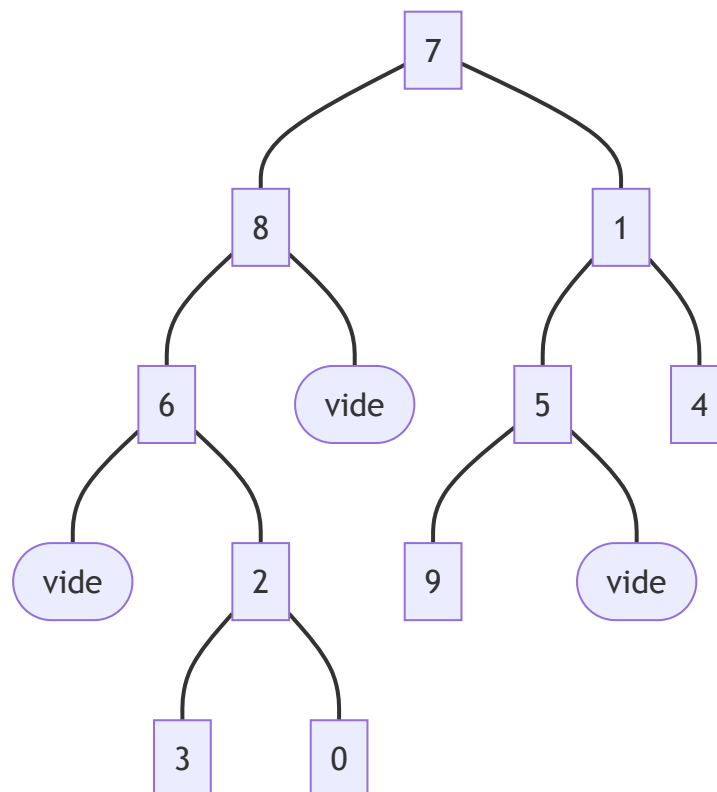
On stocke un arbre binaire dans un tableau d'enregistrements. Chaque enregistrement E est défini par:

- E.clé : valeur du nœud
- E.gauche : indice du fils gauche dans le tableau
- E.droit : indice du fils droit dans le tableau

Une valeur de E.gauche ou E.droit égale à -1 indique qu'il n'existe pas de fils.

Indice dans le tableau	0	1	2	3	4	5	6	7	8	9
Clé	23	2	3	5	7	11	13	37	41	19
Gauche	-1	5	3	-1	-1	9	-1	8	6	-1
Droit	-1	4	0	-1	-1	-1	2	1	-1	-1

1. Représenter cet arbre. Est-il binaire ? Entier ? Complet ?



C'est un arbre binaire mais il n'est pas complet ni entier.

2. Définir le code en C permettant de définir ce tableau (statique)

```
typedef struct item {  
    int val;  
    int left;  
    int right;  
};
```

```
item table[10];
```

3. Écrire une fonction en C qui renvoie l'enregistrement de la racine

```
int getRoot(item* table, int len, item* root) {  
    for(int i = 0; i < len; i++) {  
        for(int j = 0; j < len; j++) {  
            // on vérifie si i est un fils de j  
            if(table[j].left == i || table[j].right == i) {  
                break; // nouvelle itération de la boucle extérieure  
            }  
        }  
  
        return i;  
    }  
  
    // pas de racine, il y a une boucle  
    return -1;  
}
```

Solution:

```
item getRoot(item* table) {  
    int search[10] = {0};  
  
    for(int i = 0; i < 10; i++) {  
        if(table[i].left != -1) {  
            search[table[i].left] = 1;  
        }  
  
        if(table[i].right != -1) {  
            search[table[i].right] = 1;  
        }  
    }  
  
    for(int i = 0; i < 10; i++) {  
        if(search[i] == 0)  
            return table[i];  
    }  
}
```

```
    }
}
```

Solution 2:

```
item getRoot(item* table, int len) {
    int root = 0;

    for(int i = 0; i < len; i++) {
        root += i;

        if(table[i].left != -1) {
            root -= table[i].left;
        }

        if(table[i].right != -1) {
            root -= table[i].right;
        }
    }

    return table[root];
}
```

4. Écrire une fonction en C qui affiche la valeur de toutes les feuilles de T

```
void showValues(item* table, int len) {
    item root;
    int rootIndex = getRoot(table, len, &root);

    showChildValues(rootIndex, table);
}

void showChildValues(int index, item* table) {
    item val = table[index];

    if(val.left == -1 && val.right == -1) {
        printf("%d\n", val.val);
    }

    if(val.left != -1)
        showChildValues(val.left, table);

    if(val.right != -1)
        showChildValues(val.right, table);
}
```

Solution:

```
void showValues(item* table, int len) {
```

```

for(int i = 0; i < len; i++) {

    // uniquement vrai si gauche = droite = -1
    if(table[i].left == table[i].right) {
        printf("%d", table[i].val);
    }
}
}

```

Arbres binaires de recherche

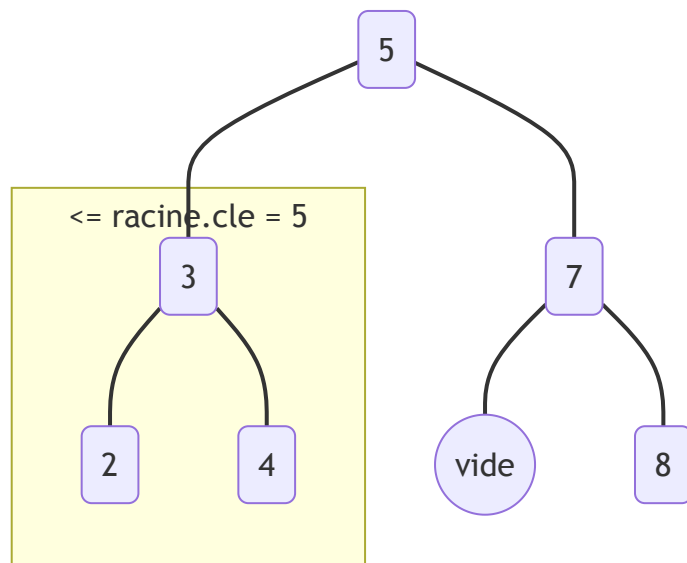
Arbre binaire marqué:

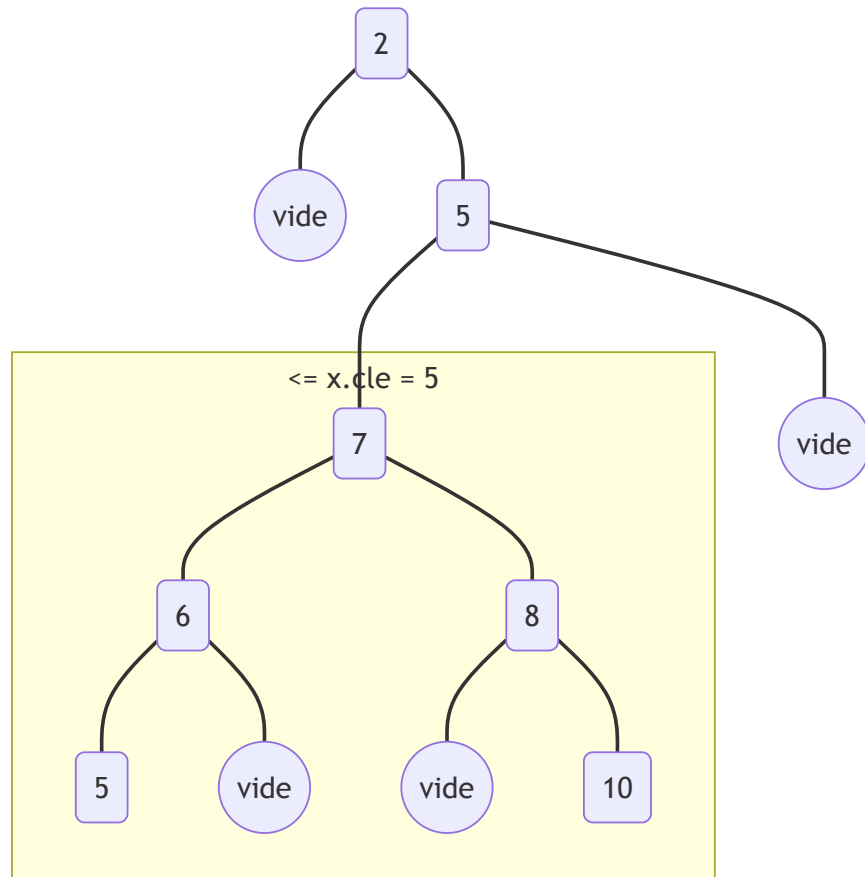
- Présence d'une étiquette pour chaque nœud x de A
- Étiquettes
 - Même type
 - Présence d'une clé appartenant à un ensemble E
 - Relation d'ordre total sur E
 - Notation $x.clé$

Organisation de l'arbre binaire de recherche: soit un nœud x de A .

- Alors toutes les étiquettes des nœuds y des sous arbres gauches de x sont telles que $y.clé \leq x.clé$
- Et toutes les étiquettes des nœuds y des sous arbres droits de x sont telles que $y.clé \geq x.clé$

Exemples:





Nœud:

- Étiquette
 - clé
 - information
- Fils gauche
- Fils droit

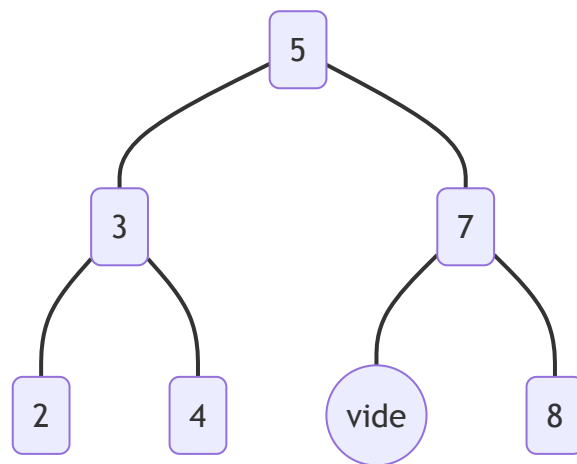
Opérations fondamentales des ABR:

- Constructeur
- Accesseurs
 - Recherche d'une clé
 - Rechercher le minimum
 - Rechercher le maximum
 - Rechercher un successeur
 - Rechercher un prédécesseur

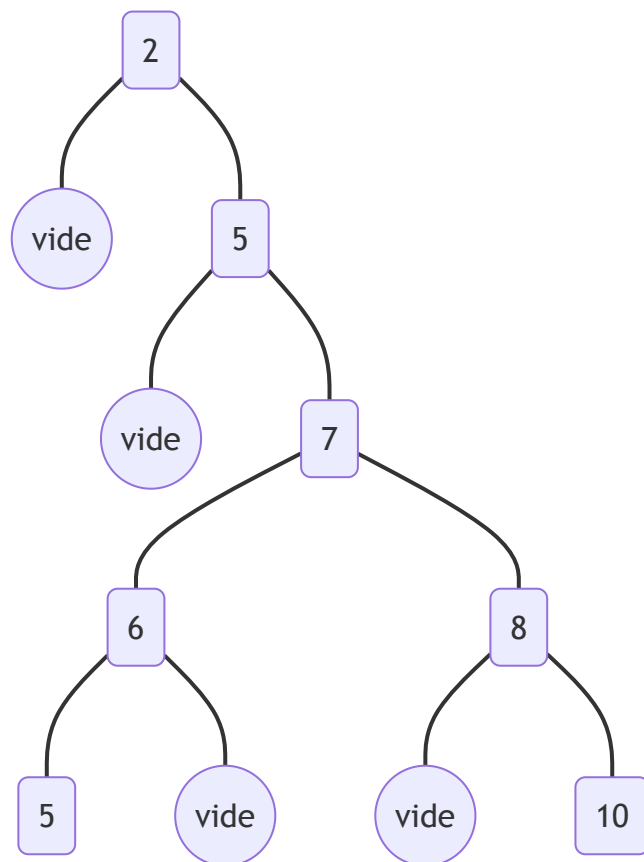
- Transformateurs
 - Insérer un nœud
 - Supprimer un nœud

Utilité: faciliter une recherche avec une recherche par clé dans l'ABR

- temps de recherche proportionnel à la hauteur h de l'arbre
 - $T(n) = O(n)$ recherche linéaire dans le pire des cas (liste chaînée)
 - $T(n) = O(\log_2(n))$ recherche en temps logarithmique en moyenne pour un ABR complet.
- Dépend de la construction de l'arbre:



6 nœuds, hauteur 2 : Efficace



6 nœuds, hauteur 4 : Peu efficace

L'efficacité augment quand la hauteur diminue.

Arbre dégénéré (filiforme) : chaque nœud n'a qu'un enfant.

Arbres binaires bicolores (rouge - noir)

Arbre de recherche: possibilité d'arbre déséquilibré (filiforme, dégénéré)

- Recherche en $O(n)$ où n est le nombre de nœuds
- aucun apport par rapport à une liste chaînée

Arbre bicolore: arbre de recherche particulier, approximativement équilibré

- Recherche en $\Theta(\log_2(n))$

Arbre binaire bicolore \rightarrow marqué

- présence d'une étiquette pour chaque nœud x de A

- Fils gauche, fils droit
- Étiquettes
 - même type
 - clé
 - information
 - information supplémentaire : *couleur* (1 bit)

Propriétés rouge noir:

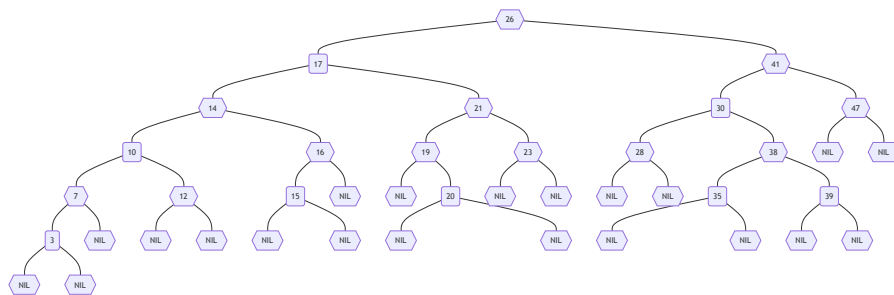
- chaque nœud est soit rouge, soit noir
- chaque feuille est noire
- la racine est noire
- si un nœud est rouge, ses deux enfants sont noirs
- pour chaque nœud, tous les chemins simples reliant le nœud à des feuilles situées plus bas dans l'arbre contiennent le même nombre de nœuds noirs

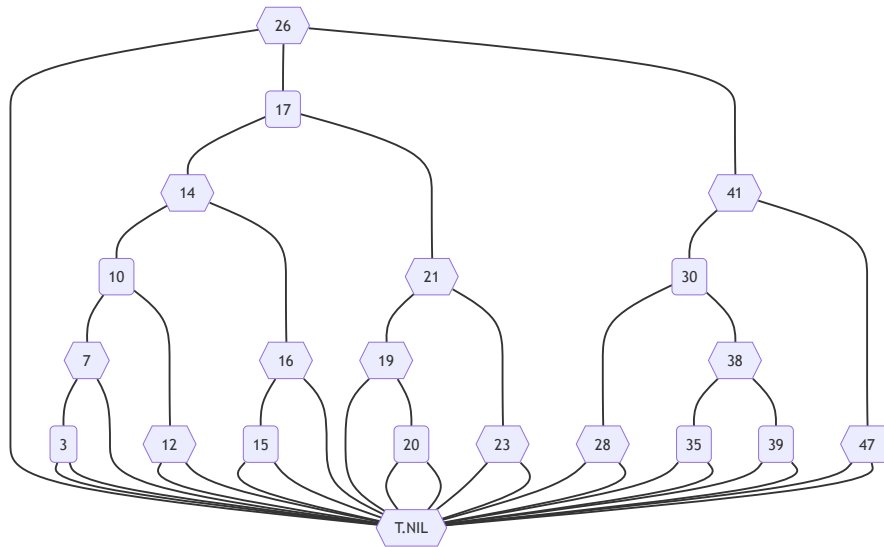
Remarque:

- Ajout de nœuds noirs vide (pointeur sur NIL) pour générer les feuilles
- Informations contenues dans les nœuds internes

Exemple:

Les nœuds rouges sont entre $[]$ et les nœuds noirs entre $\langle \rangle$

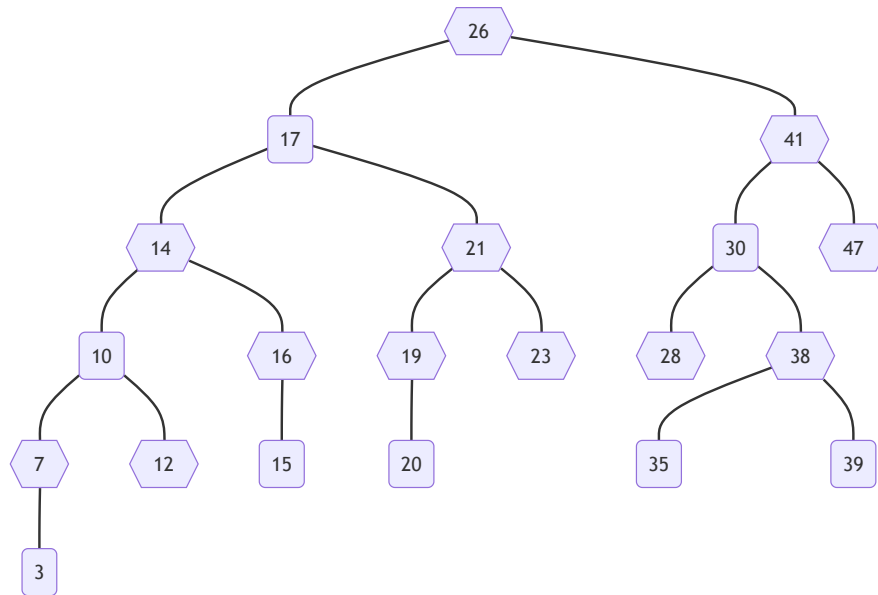




T.NIL : sentinelle

- Simplification des conditions aux limites
- Même attributs qu'un nœud ordinaire:
 - Valeur: noir
 - Autres attributs: valeurs quelconques
- Aussi parent de la racine

Hauteur noire omise



Propriété rouge - noire : la longueur L d'un chemin allant de la racine à une feuille est comprise entre h_n et $2h_n$.

Chemin: suite de nœuds dont chacun est le prédécesseur ou le successeur du suivant.

h_n étant la profondeur noire de l'arbre.

Justification:

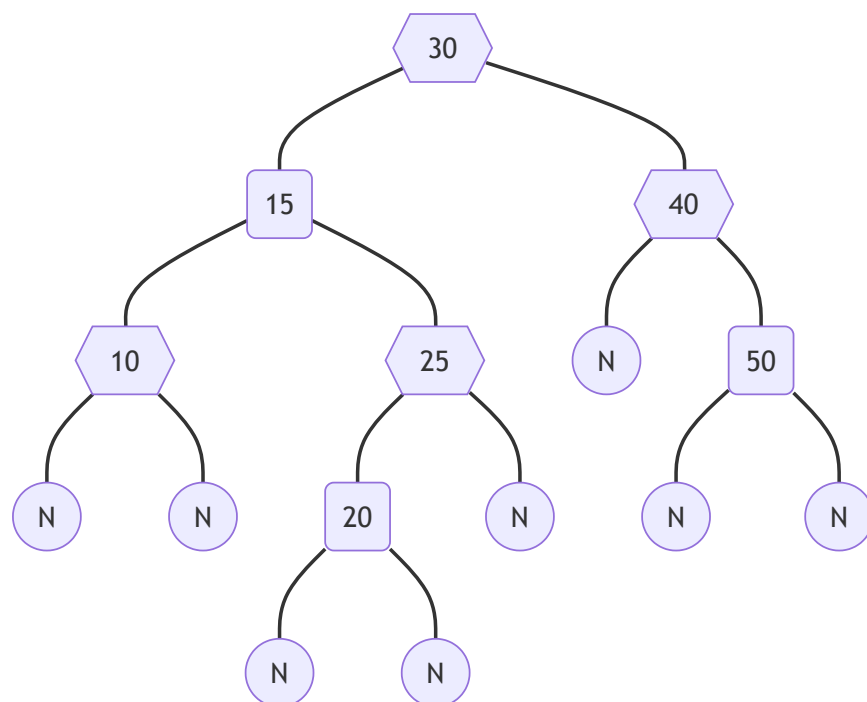
Si tous les nœuds sont noirs, alors h_n est égale à la profondeur de l'arbre.

S'il y a systématiquement une alternance entre nœuds rouges et noirs, alors la hauteur totale h de l'arbre vaut $2h_n$

Exercice 1:

- Arbre 1 : non (rouge \rightarrow rouge)
- Arbre 2 : non (racine rouge)
- Arbre 3 : non (nombre de nœuds noirs \neq)
- Arbre 4 : non (n'est pas un ABR car $10 < 22$)

Exercice 2:



Exercice 3:

1. hauteurs $h_n(30) = 3$, $h_n(20) = 3$, $h_n(35) = 2$ et $h_n(50) =$