Structures de Données

Pr. Salma AZZOUZI

Plan

Les structures de Données non Linéaires(Arborecsantes)

- Introduction

INTRODUCTION : Les structures de données non Linéaires

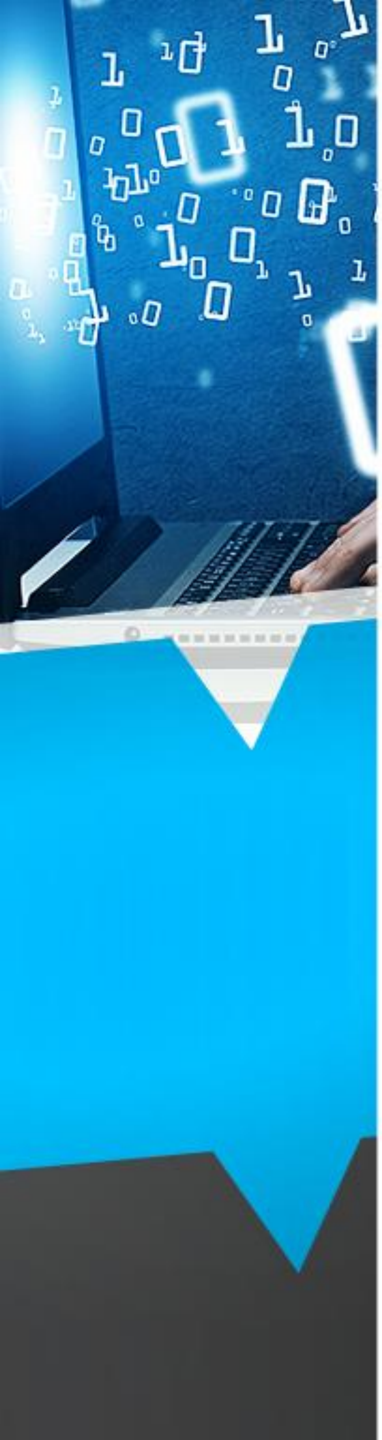
- La structure de données est considérée comme **linéaire** si les éléments de données construisent une séquence d'une liste linéaire.
- Les éléments sont attachés de manière adjacente les uns aux autres et dans un ordre spécifié.
- Il consomme de l’espace mémoire linéaire et les éléments de données devant être stockés de manière séquentielle dans la mémoire.
- Lors de la mise en œuvre de la structure de données linéaire, la quantité de mémoire nécessaire est déclarée précédemment (Tableaux).
- Cela ne permet pas une bonne utilisation de la mémoire et entraîne un gaspillage de mémoire. Les éléments de données sont visités de manière séquentielle, un seul élément pouvant être directement atteint.

INTRODUCTION : Les structures de données non Linéaires

- **La structure de données non linéaire** n'organise pas les données de manière consécutive, mais plutôt dans un ordre de tri.
- En cela, les éléments de données peuvent être attachés à plusieurs éléments présentant la relation hiérarchique qui implique la relation entre l'enfant, le parent et les grands-parents.
- Dans la structure de données non linéaire, la traversée d'éléments de données et l'insertion ou la suppression ne sont pas effectuées de manière séquentielle.
- La structure de données non linéaire utilise efficacement la mémoire et ne nécessite pas de déclaration préalable de la mémoire. Il existe deux exemples courants de la structure de données non linéaire - **arbre** et **graphique** . Une structure de données arborescente organise et stocke les éléments de données dans une relation hiérarchique.
- .

INTRODUCTION : Les structures de données non Linéaires

- La structure de données non linéaire utilise efficacement la mémoire et ne nécessite pas de déclaration préalable de la mémoire.
-
- Il existe deux exemples courants de la structure de données non linéaire :
  - **Arbres,**
  - **Graphes**.
- .

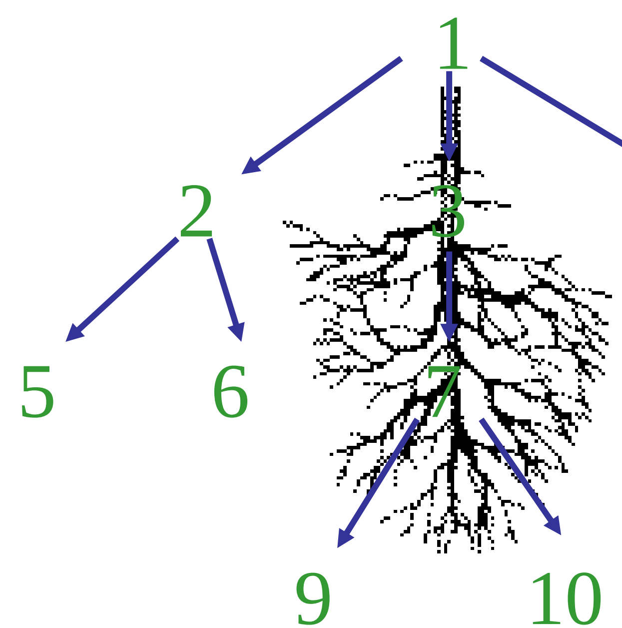


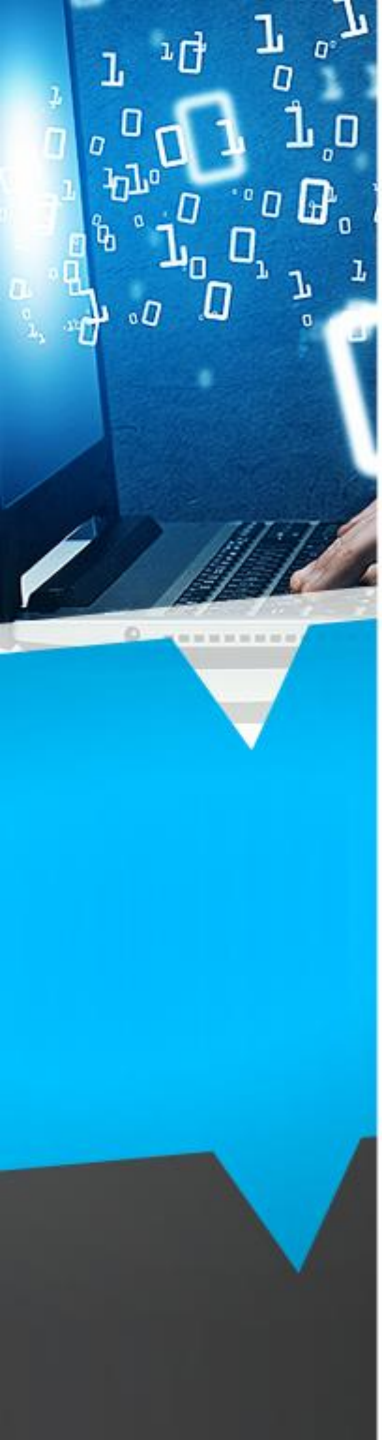
ARBRES : Définition

- Un arbre est un ensemble d'éléments appelés nœuds, liés par une relation induisant une structure hiérarchique;

- Un nœud parmi les nœuds de l'arbre se singularise → la racine,

- Un nœud, comme tout élément d'une liste, peut-être de n'importe quel type.





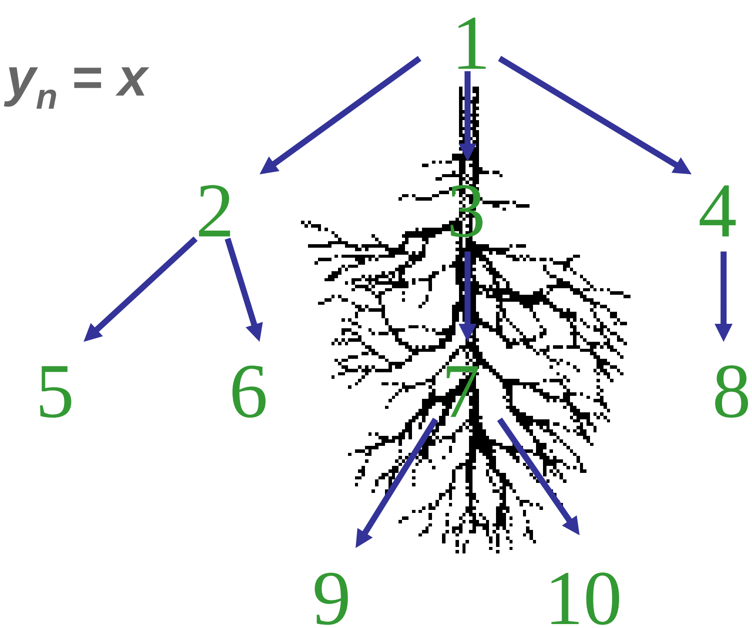
ARBRES : Définition formelle

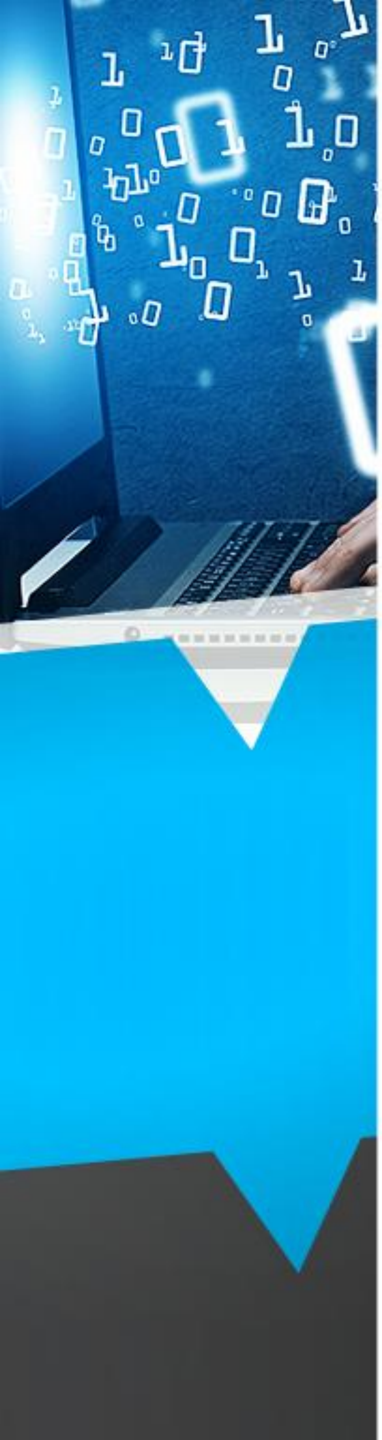
Un Arbre : $A = (N,P)$

- **N ensemble des nœuds**
- **P relation binaire « parent de »**
- **$r \in N$ la racine**

$\forall\, x \in N \;\; \exists$ un seul chemin de r vers x

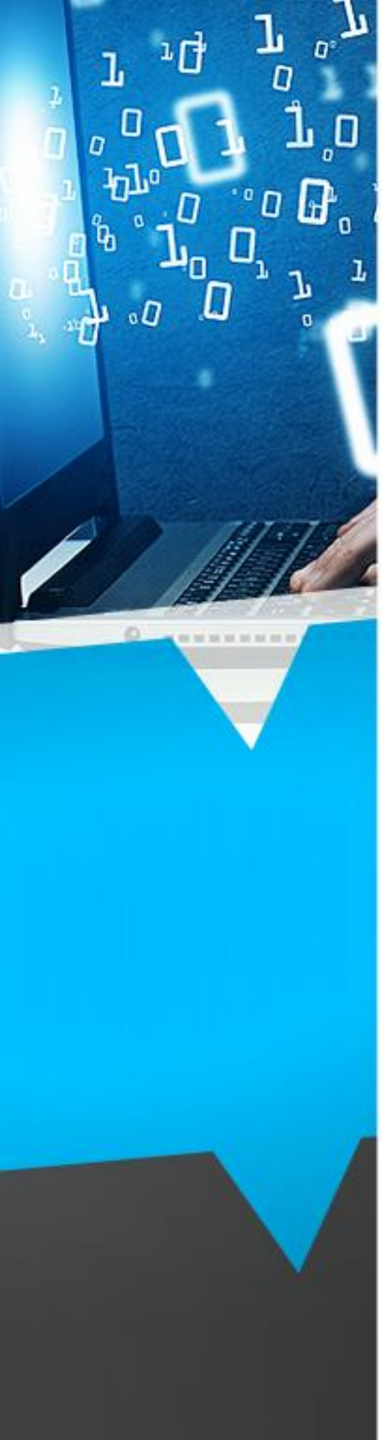
$$r = y_o \;\; P \;\; y_1 \;\; P \;\; y_2 \; \ldots \; P \;\; y_n = x$$





ARBRES : Vocabulaires

- **nœud :** caractérisé par une donnée associée + un nombre fini de fils, possède un unique père;
- **feuille :** nœud sans fils;
- **nœud interne :** un nœud qui n’est pas une feuille;
- **arité d’un nœud n :** nombre de fils du nœud n;
- **arité d’un arbre a :** nombre maximal de fils d’un nœud de a;
- **racine d’un arbre a** : c’est le seul nœud sans père;

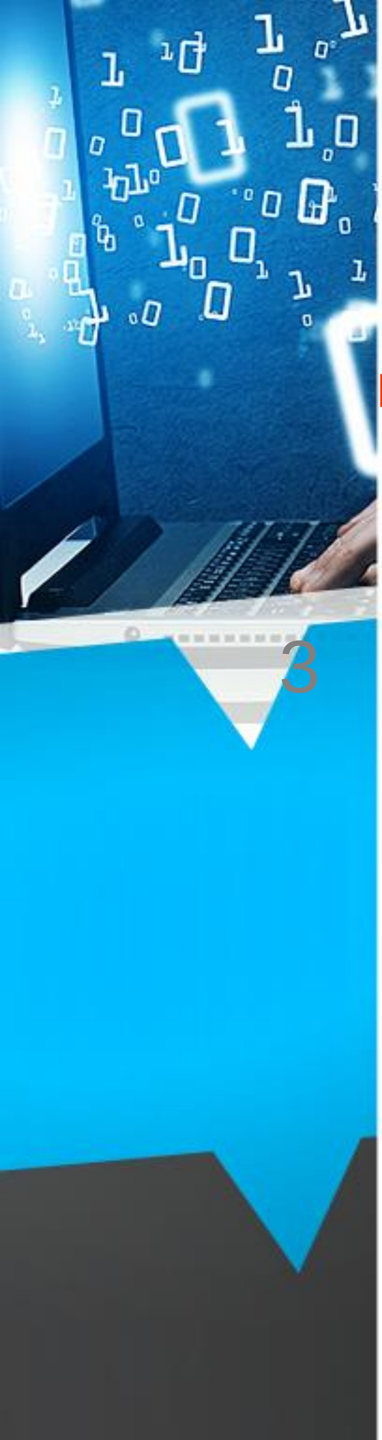


ARBRES : Vocabulaires

- **profondeur d'un nœud n :** nombre de nœuds sur la branche entre la racine et le nœud n.

- **hauteur d'un arbre a :** c'est le nombre de nœuds sur la branche qui va de la racine de a à la feuille de profondeur maximale

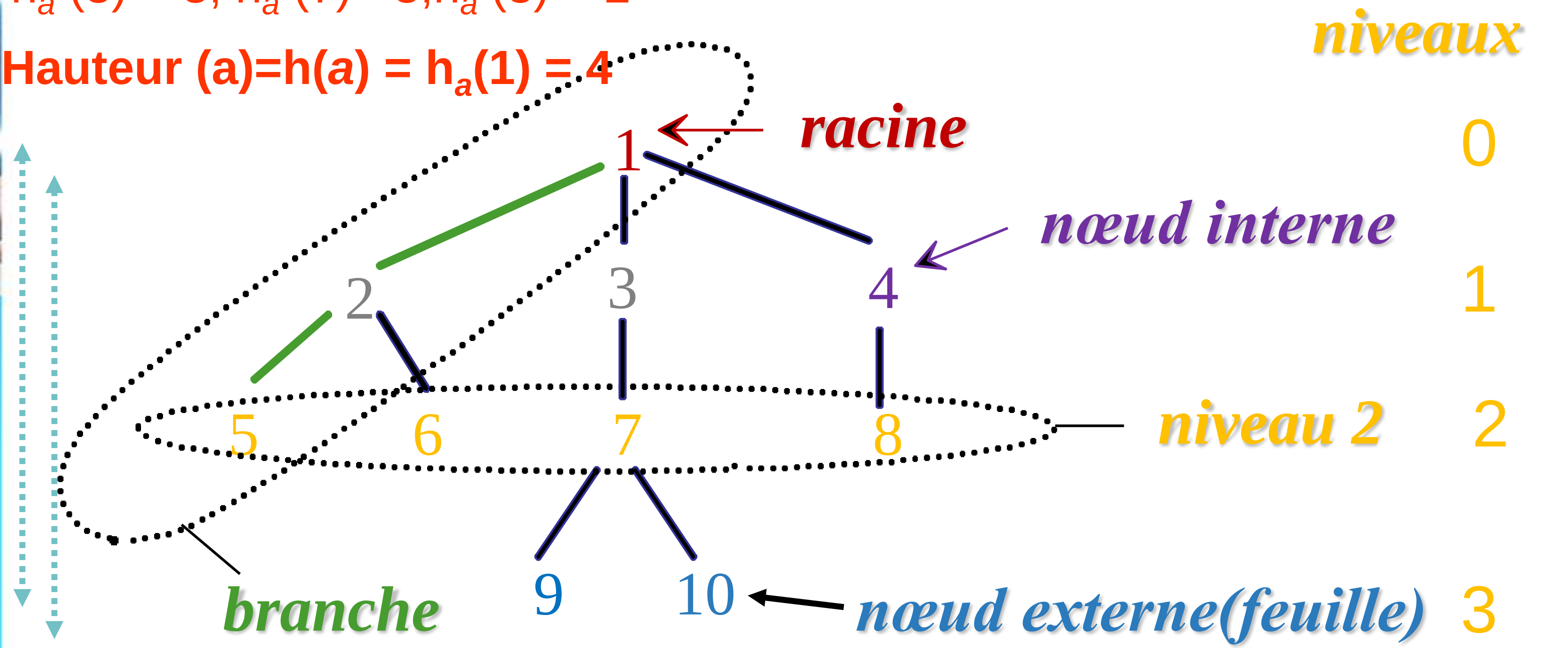
- **La taille d'un arbre B :** correspond au nombre de ses nœuds;



ARBRES : Vocabulaires

$h_a(8) = 3$, $h_a(7) = 3$, $h_a(3) = 2$

Hauteur (a)=h(*a*) = $h_a(1) = 4$

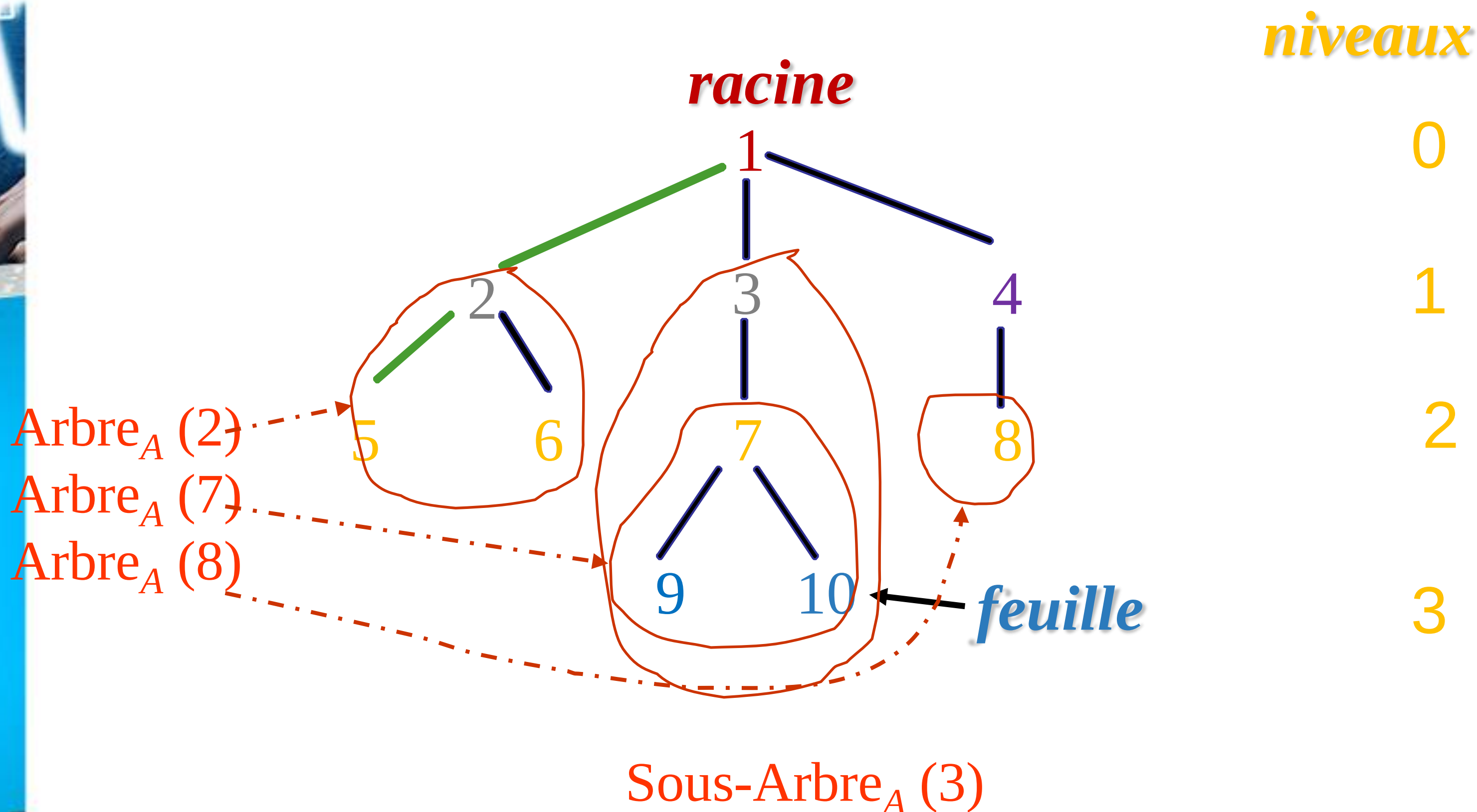


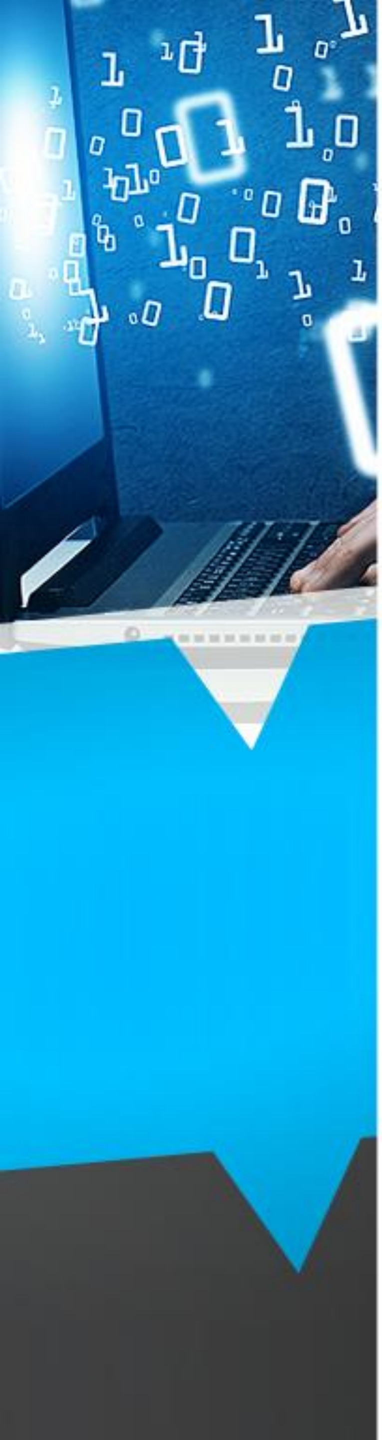
2, 3, 4 ***enfants de 1***
3, 4 ***frères de 2***
2 ***fils de 1***
1 ***père de 2, 3 et 4***
1, 3 ***ancêtres de 7***
9, 10 ***descendants de 7***

Taille(a)=10

ARBRES : Vocabulaires

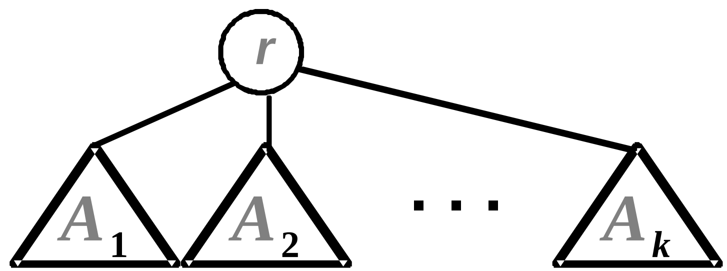
Sous Arbre





ARBRES : Définition Récursive

- un arbre est :
  - soit vide
  - soit constitué d'un nœud auquel sont chaînées un ou plusieurs sous arbres



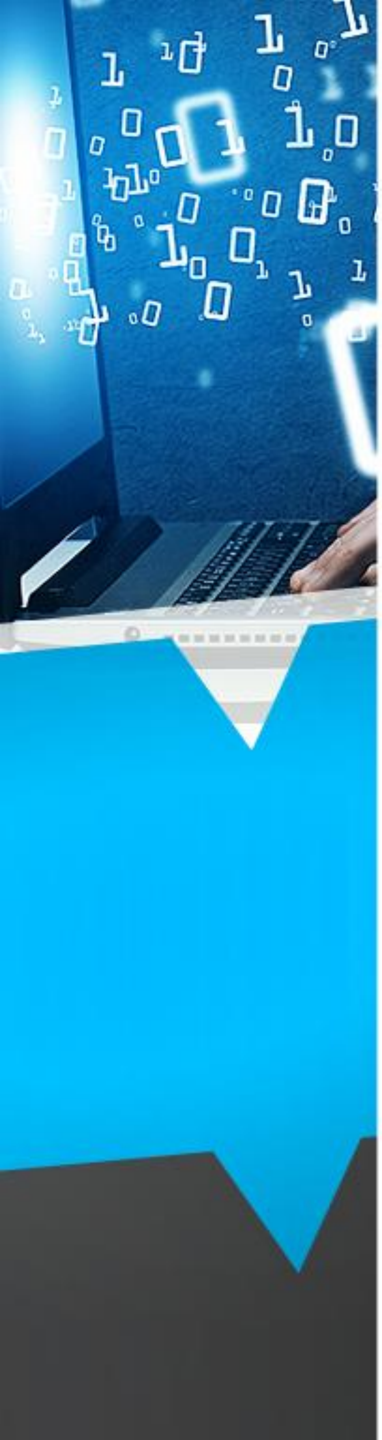
<u>Définition récursive formelle</u>

$$\text{Arbre A} = \begin{cases} \text{arbre vide ou} \\ (r, \{A_1, ..., A_k\}) \end{cases}$$

Avec r élément, $A_1, ..., A_k$ arbres

ARBRES : Exemples

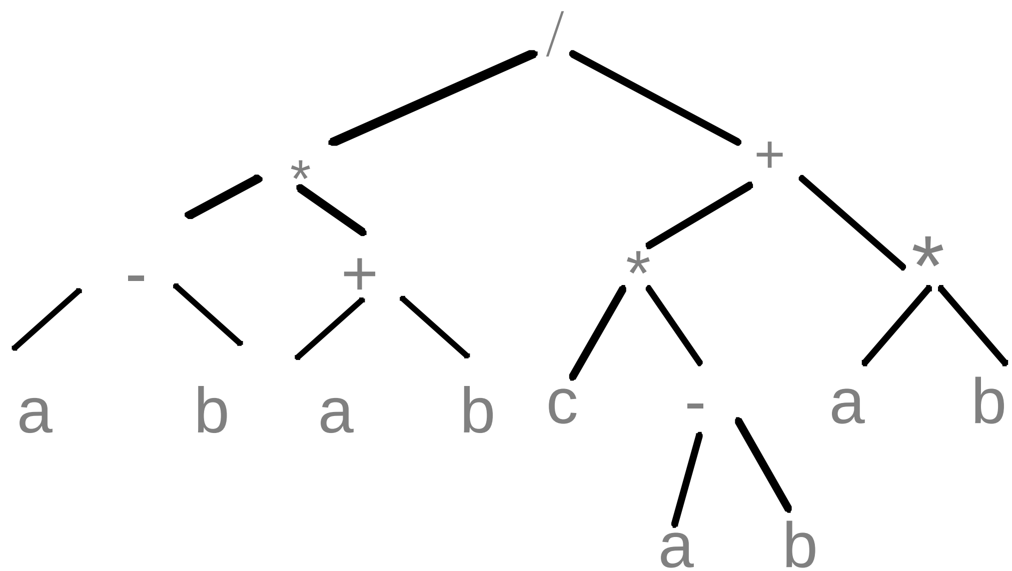
tables des matières



ARBRES : Exemples

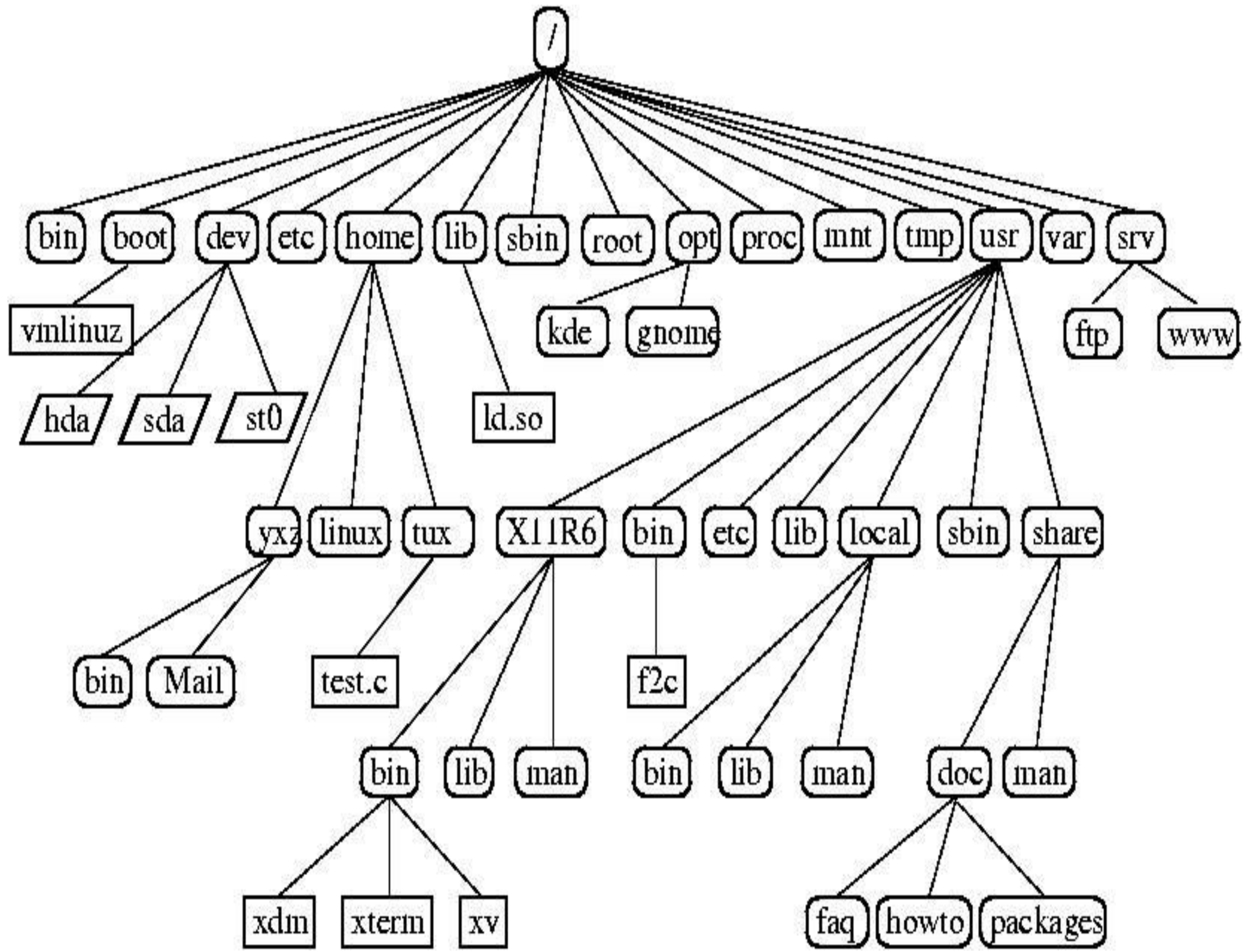
- Expression arithmétique

$$(a - b) * (a + b) \ / \ (c * (a - b) + (a*b))$$



ARBRES : Exemples

- Arborescence des fichiers

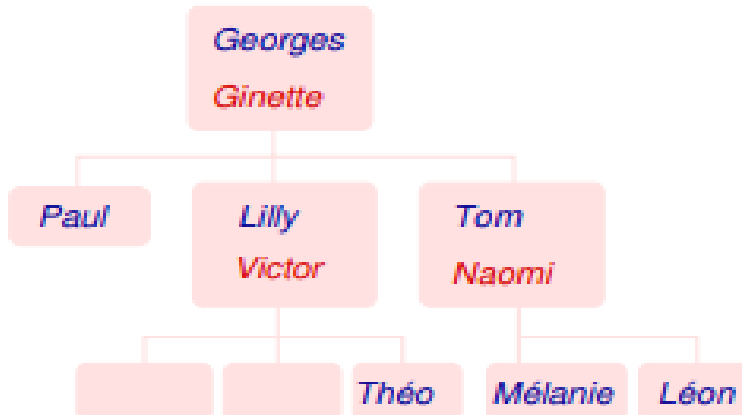


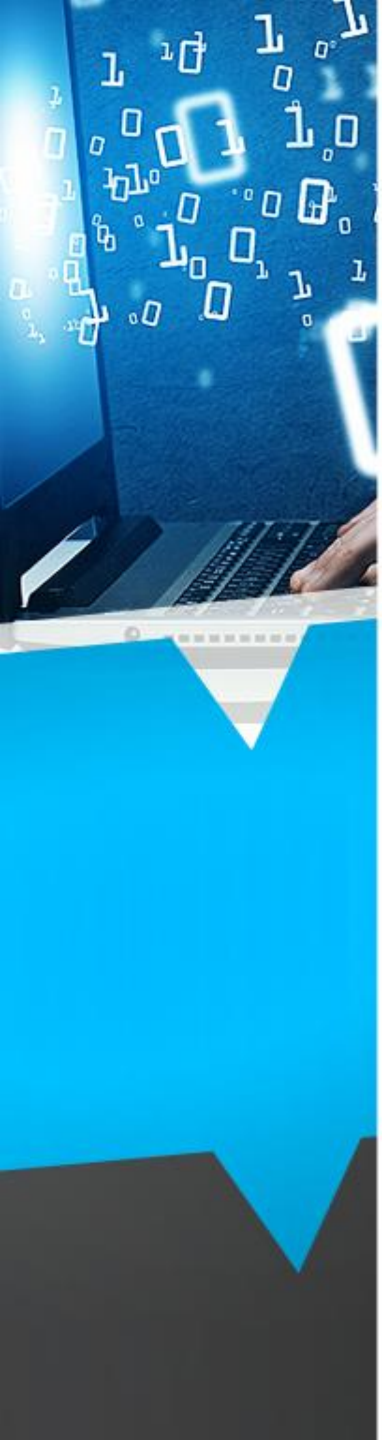
ARBRES : Exemples

- Arbre Syntaxique

ARBRES : Exemples

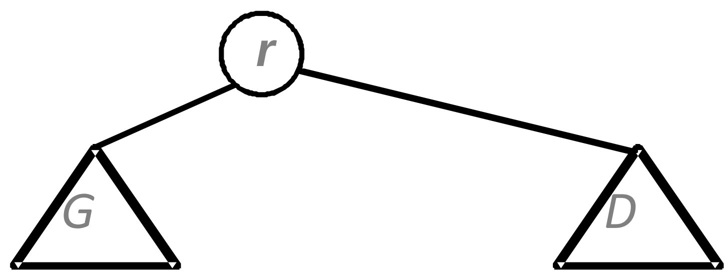
- Arbres généalogiques





ARBRES BINAIRES

- Lorsqu'un arbre admet, pour chaque nœud, au plus n fils, l'arbre est dit n-aire
- si n est égale 2, l'arbre est dit binaire

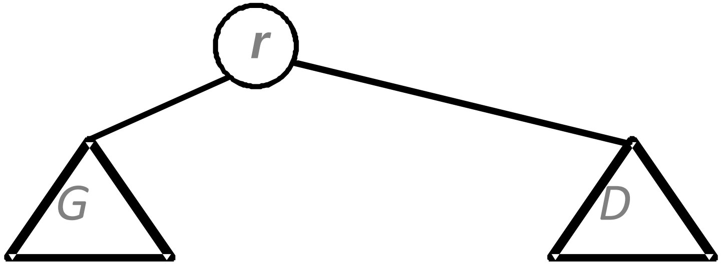


ARBRES BINAIRES

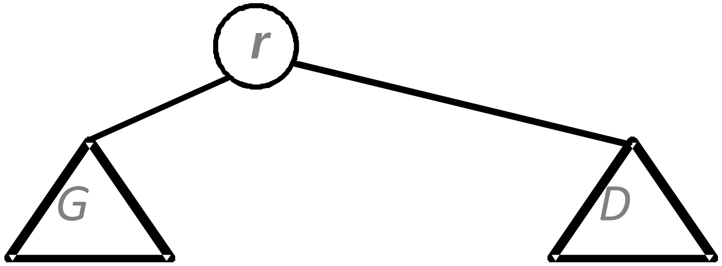
Définition

Un arbre binaire est soit :

- l'arbre vide,
- soit un triplet (e, g, d), appelé nœud, dans lequel
  - e est l'élément, ou encore étiquette, de la racine de l'arbre,
  - g est le sous-arbre gauche de l'arbre ;
  - et d est le sous-arbre droit de l'arbre.



ARBRES BINAIRES



Les sous-arbres gauche et droit d'un arbre binaire non vide sont eux-mêmes des arbres binaires.

La structure d'arbre binaire est donc une structure récursive

ARBRES BINAIRES

Implémentation

type

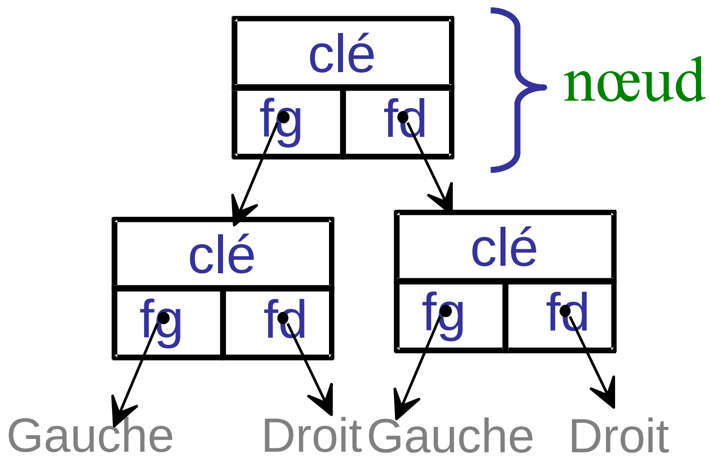
nœud = Enregistrement

Info : typeElement;

fg, fd : Arbre;

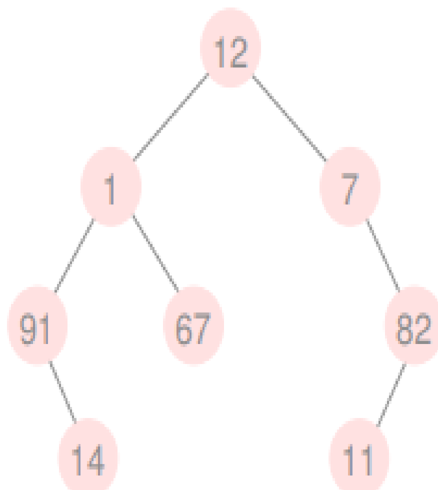
Fin_Enreg;

Arbre = ^nœud;

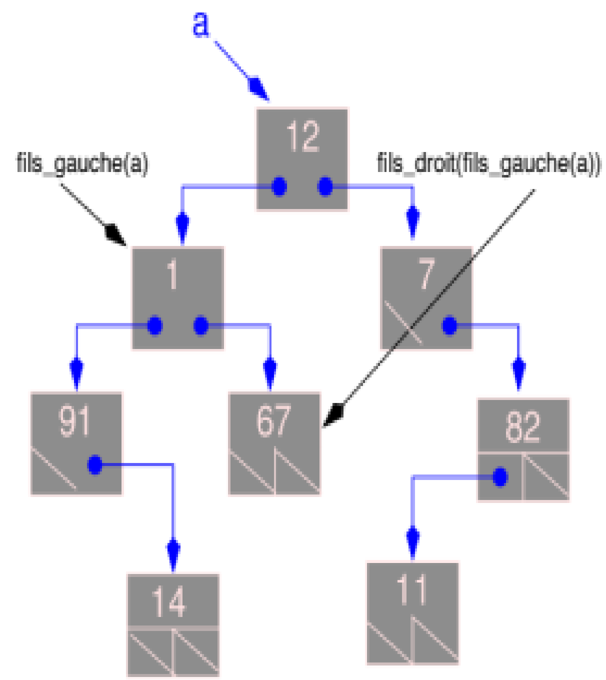


ARBRES BINAIRES

Arbre Binaire



Implémentation



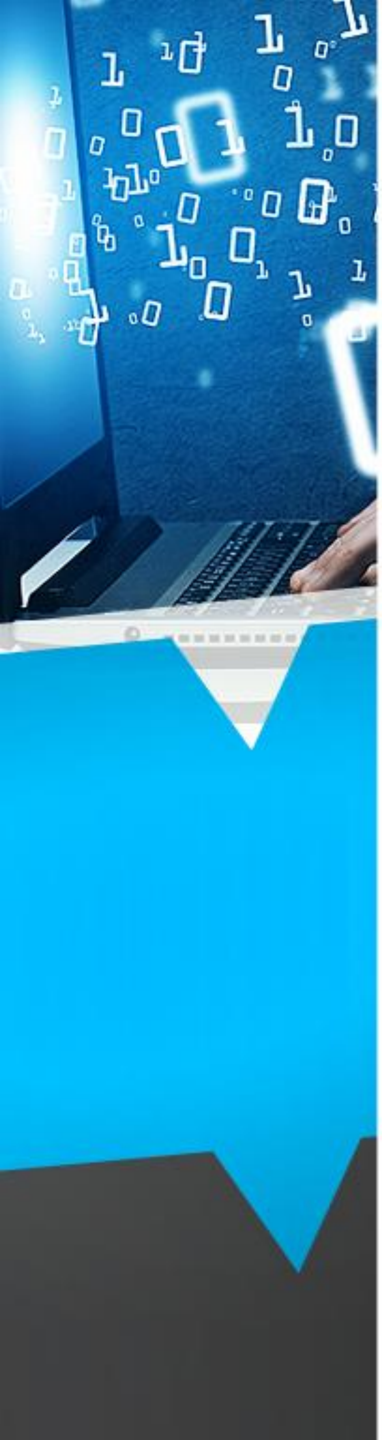
ARBRES BINAIRES: Opérations courantes

Parcours:

Le parcours le plus simple à programmer est le parcours dit en profondeur .

pour parcourir un arbre non vide a, on parcourt récursivement son sous-arbre gauche, puis son sous-arbre droit, la racine de l’arbre pouvant

- être traitée au début → ordre préfixe
- entre les deux parcours → ordre infixe
- Ou à la fin → ordre postfixe

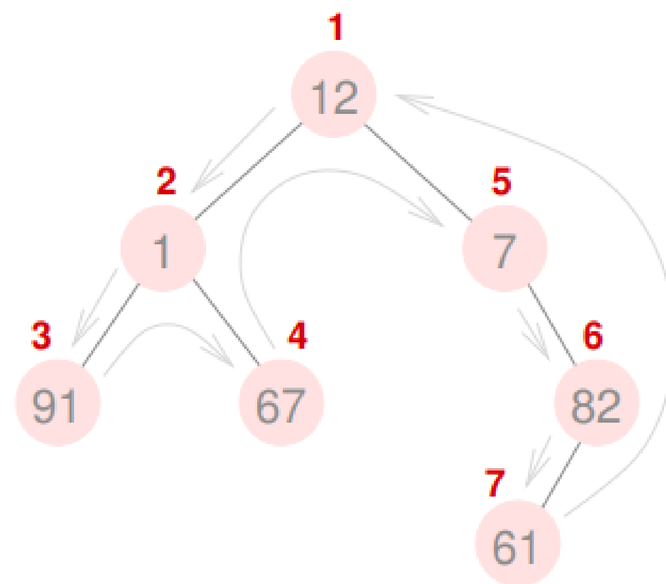


ARBRES BINAIRES: Opérations courantes

Parcours préfixe:

1. racine
2. sous-arbre gauche
3. sous-arbre droit

```
procedure prefixe(a: Arbre );
{
    Si (a<>nil) Alors
        traiter(a) ;
        préfixé(a^.fg);
        préfixé(a^.fd);
    Fin_Si
}
```



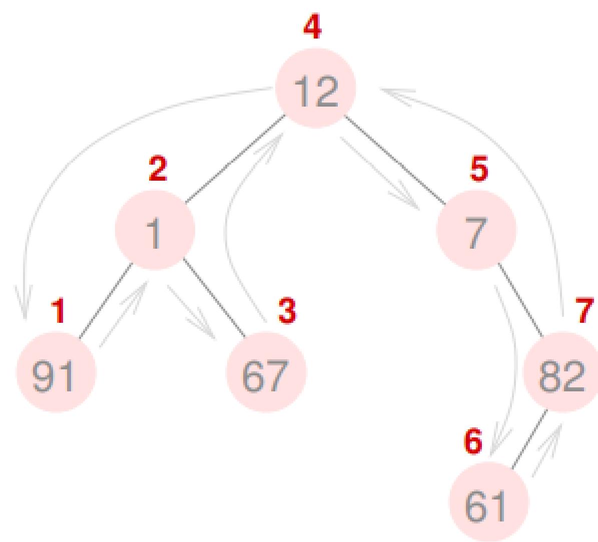
12 1 91 67 7 82 61

ARBRES BINAIRES: Opérations courantes

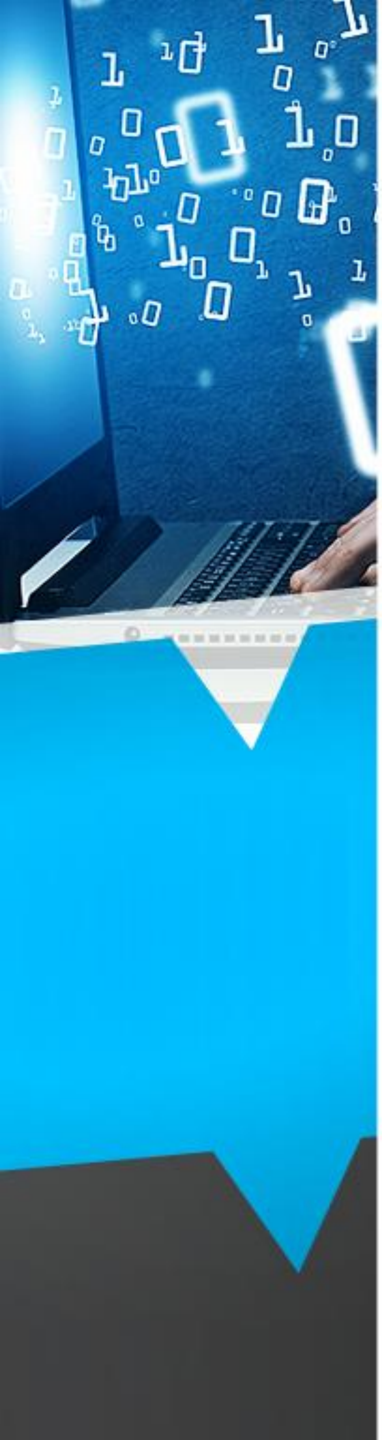
Parcours infixe:

1. sous-arbre gauche
2. racine
3. sous-arbre droit

```
procedure infixe(a: Arbre );
{
    Si (a<>nil) Alors
          Infixé(a^.fg);
          traiter(a) ;
          Infixé(a^.fd);
    Fin_Si
}
```



91 1 67 12 7 61 82

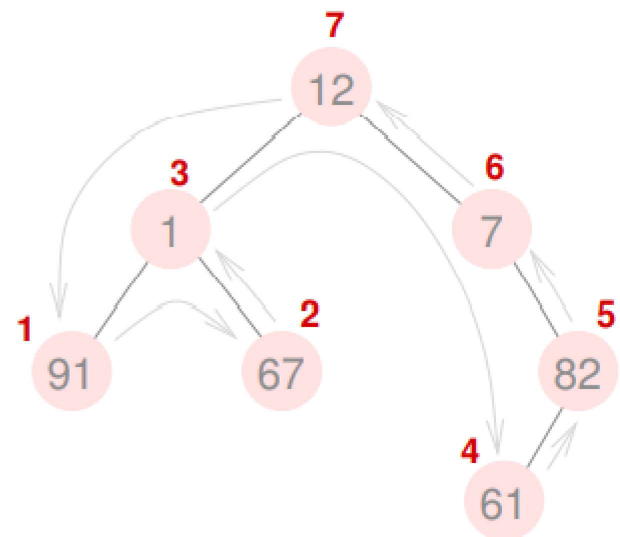


ARBRES BINAIRES: Opérations courantes

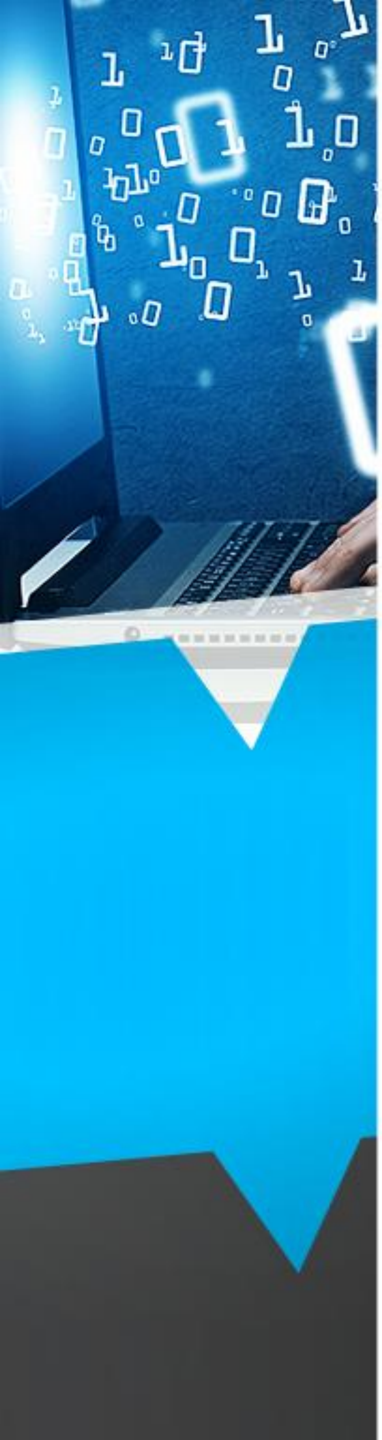
Parcours postfixe:

1. sous-arbre gauche
2. sous-arbre droit
3. racine

```
procedure postfixe(a: Arbre );
{
    Si (a<>nil) Alors
        postfixé(a^.fg);
        postfixé(a^.fd);
        traiter(a) ;
    Fin_Si
}
```



91 67 1 61 82 7 12



ARBRES BINAIRES: Opérations courantes

Rechercher un élément

```
Fonction Recherche(a:Arbre, val: typeElement) : boolean
{
        Si (a=null)  alors
           Retourner faux
        Sinon Si (a^.info =val) Alors
        Retourner true
            Sinon
      Retourner Recherche(a^.fg, val) ou Recherche(a^. fd, val);
end;
```

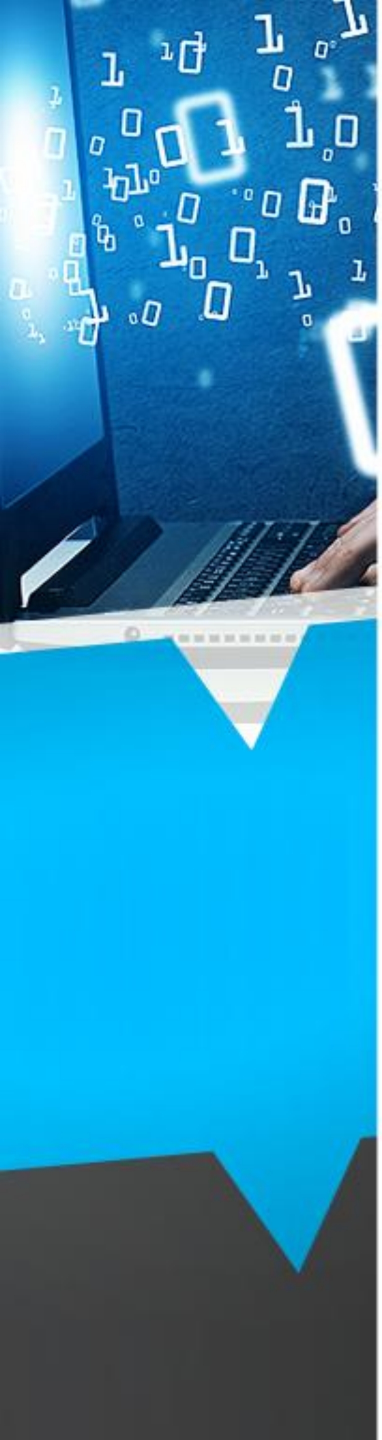
ARBRES BINAIRES: Opérations courantes

Calculer la taille d'un arbre:

Taille (A) =

- **0 si A arbre vide**
- **1 + Taille(G)+ Taille(D) avec A=(*r*, G, D)**

```
fonction taille(a: Arbre):Entier;
{
        Si(a=null)alors
            Retourner 0;
        Sinon
            Retourner 1+ taille(a^.fg) + taille(a^.fd);
}
```



ARBRES BINAIRES: Opérations courantes

Calculer le nombre de feuilles d'un arbre:

NbreFeuilles (A) =

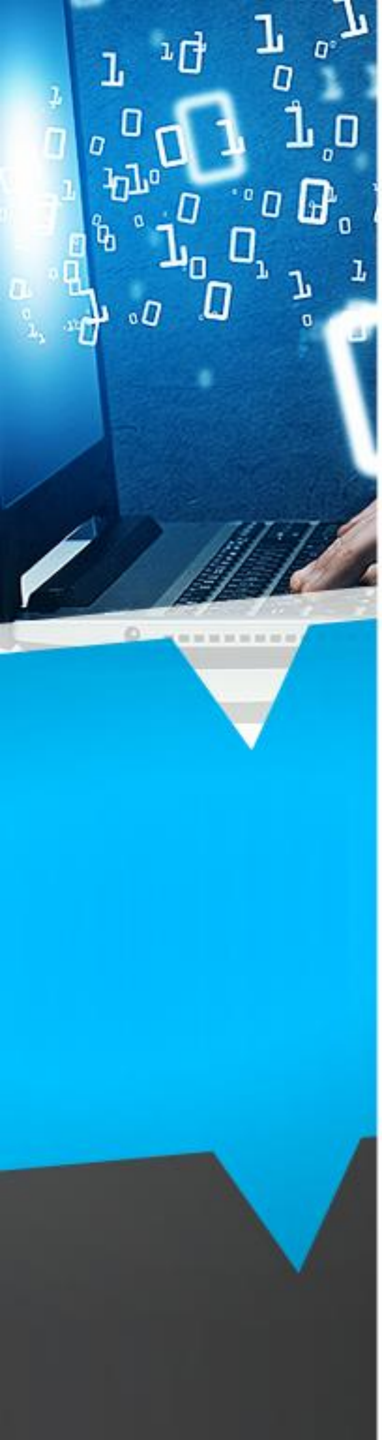
0	si arbre vide
1	si A est une feuille
nbFeuille(G)+ nbFeuille(D)	si A=(r,G, D)

```
fonction NbreFeuilles(a: Arbre): Entier
{
        Si (a=NULL) Alors
            Retourner 0 ;
        Sinon Si (a^.fg= null and a^.fd=null ) alors
                Retourner 1;
         Sinon
            Retourner NbreFeuilles(a^.fg) + NbreFeuilles(a^.fd);
}
```

ARBRES BINAIRES DE RECHERCHE: Définition

- Un arbre binaire de recherche (ou ABR) est une structure de donnée qui permet de représenter un ensemble de valeurs si l'on dispose d'une relation d'ordre sur ces valeurs.

- Les opérations caractéristiques sur les arbres binaires de recherche sont :
    - l'insertion,
    - la suppression,
    - et la recherche d'une valeur.



ARBRES BINAIRES DE RECHERCHE: Définition

- **Définition**

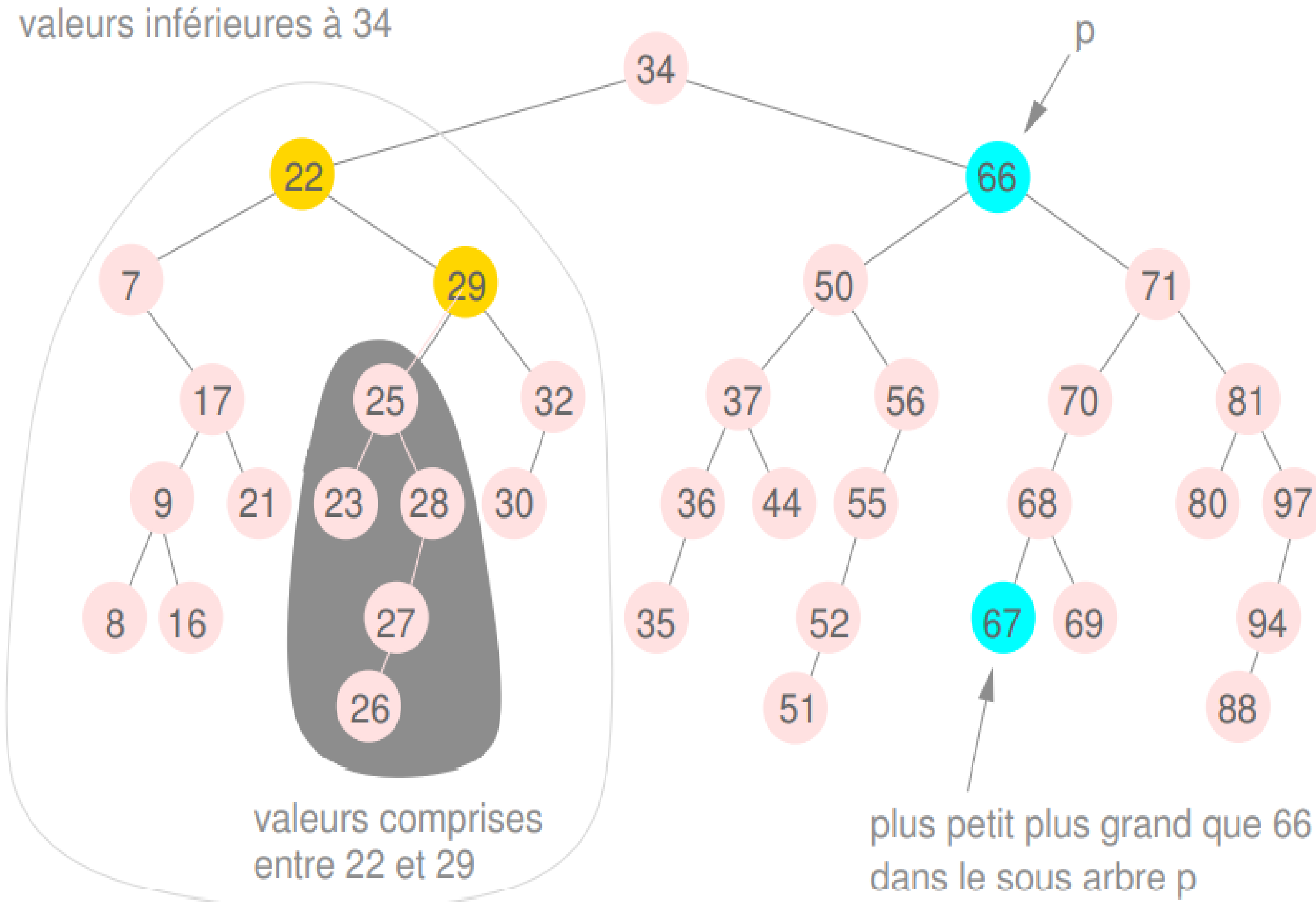
Soit E un ensemble muni d'une relation d'ordre, et soit A un arbre binaire portant des valeurs de E.

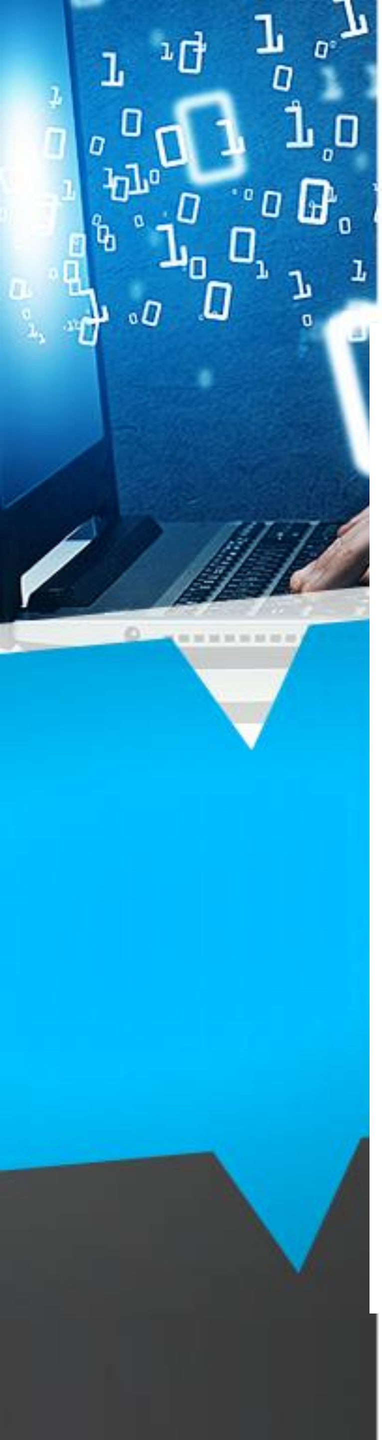
L'arbre A est un arbre binaire de recherche si pour tout noeud p de A, la valeur de p est strictement plus grande que les valeurs figurant dans son sous-arbre gauche et strictement plus petite que les valeurs figurant dans son sous-arbre droit.

Cette définition suppose donc qu'une valeur n'apparaît au plus qu'une seule fois dans un arbre de recherche.

ARBRES BINAIRES DE RECHERCHE: Définition

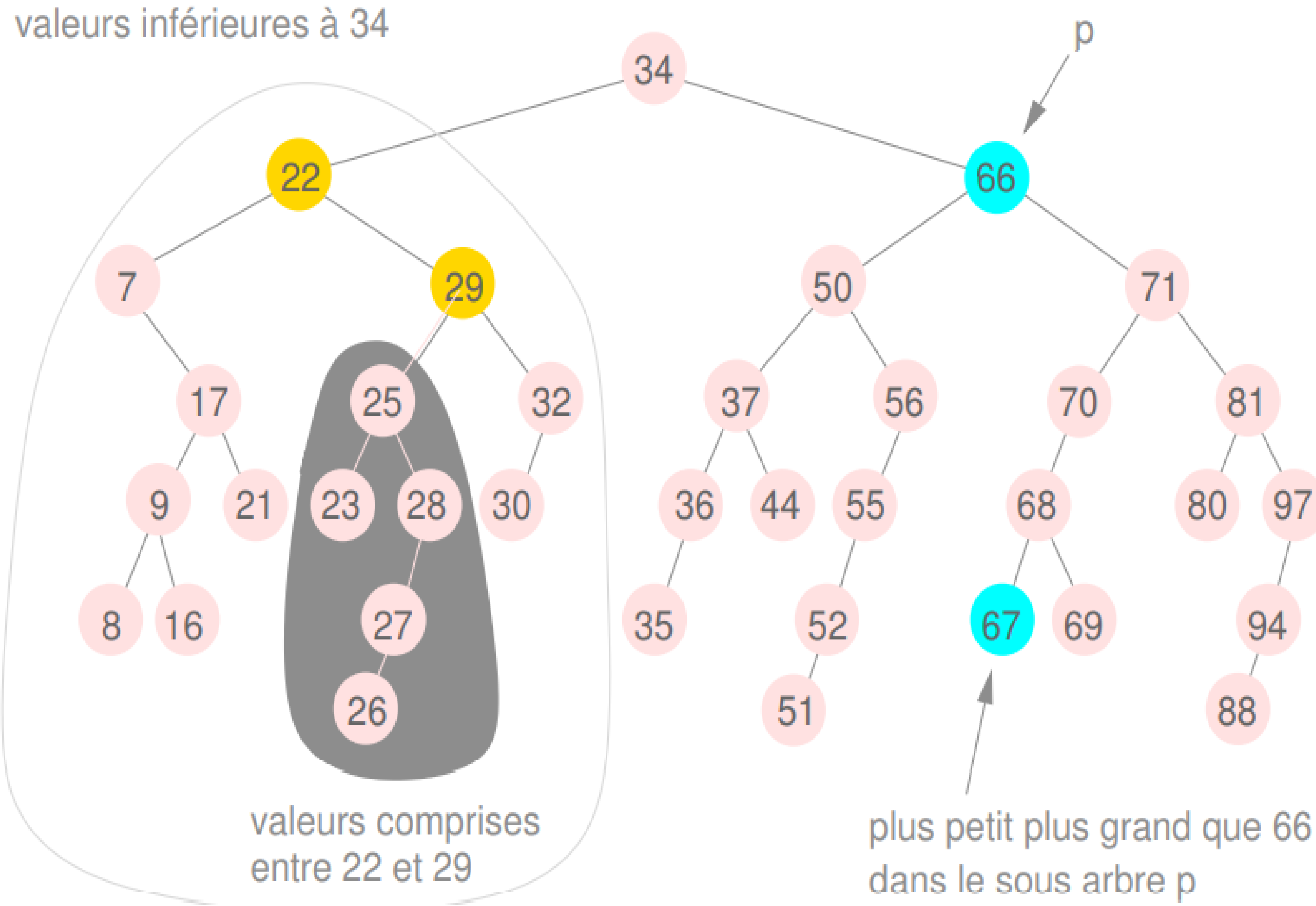
- **Exemple**

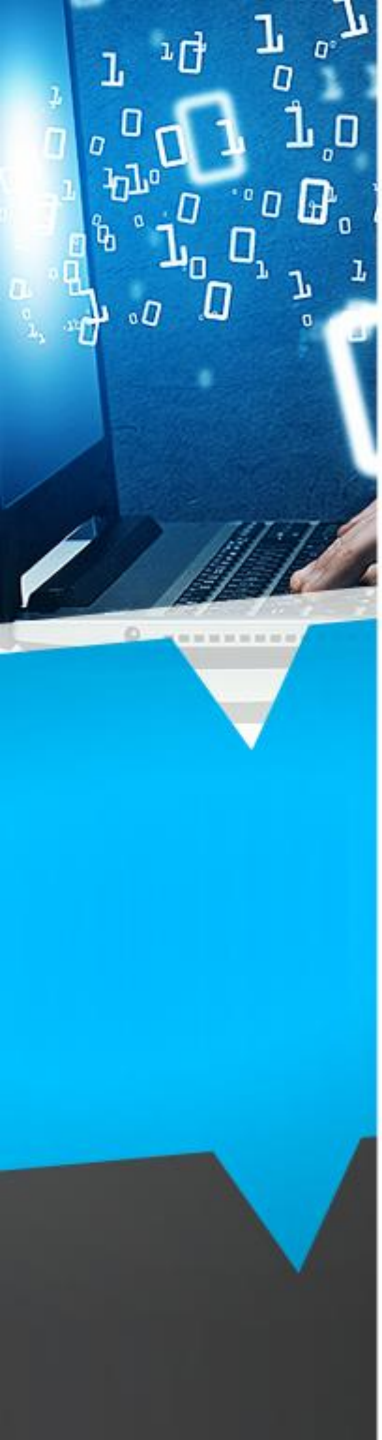




ARBRES BINAIRES DE RECHERCHE: Définition

- **Exemple**



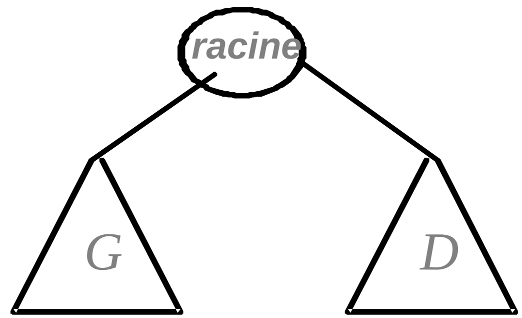


ARBRES BINAIRES DE RECHERCHE: Opération courante

- **<u>Recherche</u>**
- La recherche dans un ABR d'un nœud ayant une clé particulière est un procédé récursif.
- On commence par examiner la racine.
  - Si sa clé est la clé recherchée, l'algorithme se termine et renvoie la racine.
  - Si elle est strictement inférieure, alors elle est dans le sous-arbre gauche, sur lequel on effectue alors récursivement la recherche.
  - De même si la clé recherchée est strictement supérieure à la clé de la racine, la recherche continue dans le sous-arbre droit.
- Si on atteint une feuille dont la clé n'est pas celle recherchée, on sait alors que la clé recherchée ne figure donc pas dans l'arbre de recherche..

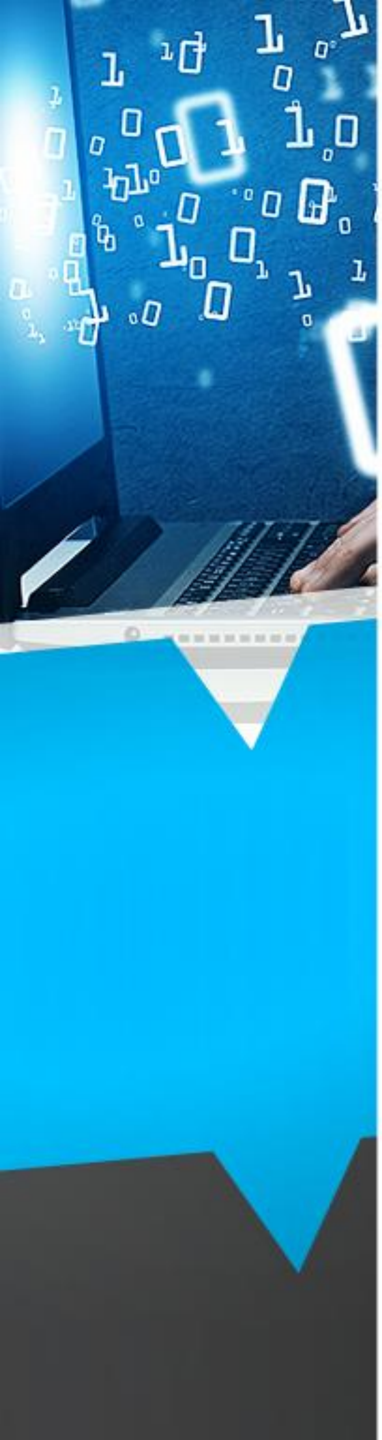
ARBRES BINAIRES DE RECHERCHE: Opération courante

- <u>**Recherche**</u>



Recherche (A, clé) =

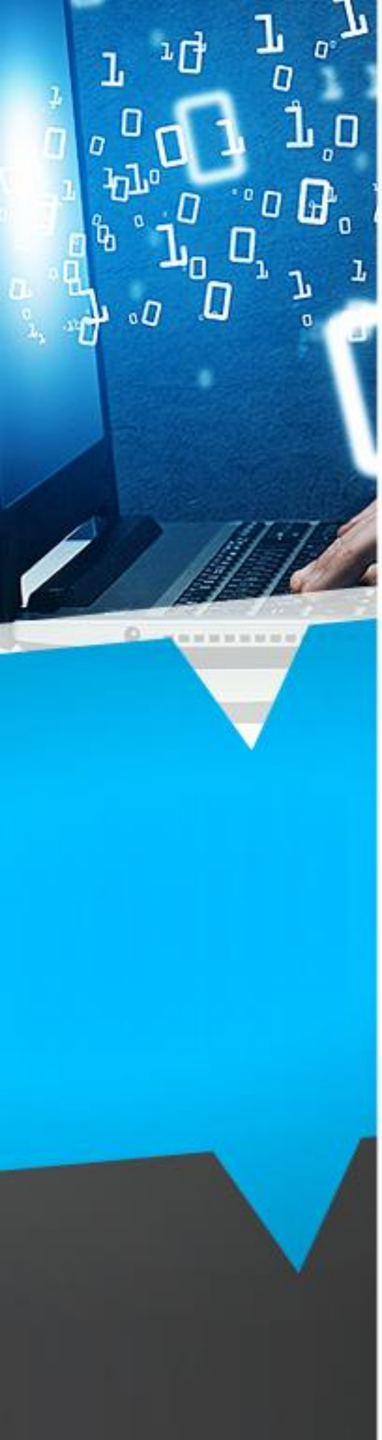
faux	***si A vide***
vrai	***si clé = racine***
Recherche (G(A), clé)	***si clé< racine***
Recherche (D(A), clé)	***si clé > racine***



ARBRES BINAIRES DE RECHERCHE: Opération courante

- **Recherche :** ***Version récursive***

```
fonction RechercheRecursif(a:Arbre, cle:typeElement):
    Boolean
{
    Si (a=Null) Alors
    Retouner faux
    Sinon Si (a^.info = cle)  Alors
                Retourner True
            Sinon  Si(cle < a^.info) Alors
                    RechercheRecursif (a^.fg, val)
                Sinon
                    RechercheRecursif (a^.fd, val);
                Fin_Si
            Fin_Si
    Fin_Si
}
```



ARBRES BINAIRES DE RECHERCHE: Opération courante

- **Recherche :** ***Version Itérative***

```
fonction RechercheItérative(a:Arbre, cle:typeElement): Boolean
Var  trouve:boolean;
{
Trouve<-false;
        Tant que (a<> Null and not(trouve)) faire
                Si (a^.info = cle)  Alors
                    Triouve <- true;
                 Sinon Si (cle <a^.info ) Alors
                            a<-  a^.fg;
                        Sinon
                             a<-  a^.fd;
                        Fin_Si
                  Fin_Si
          Fin_ Tant_que
    Retourner trouve;
   Fin_Si
}
```

ARBRES BINAIRES DE RECHERCHE: Opération courante

- **<u>Insertion</u>**

- L'insertion d'un nœud dans un ABR commence par une recherche :

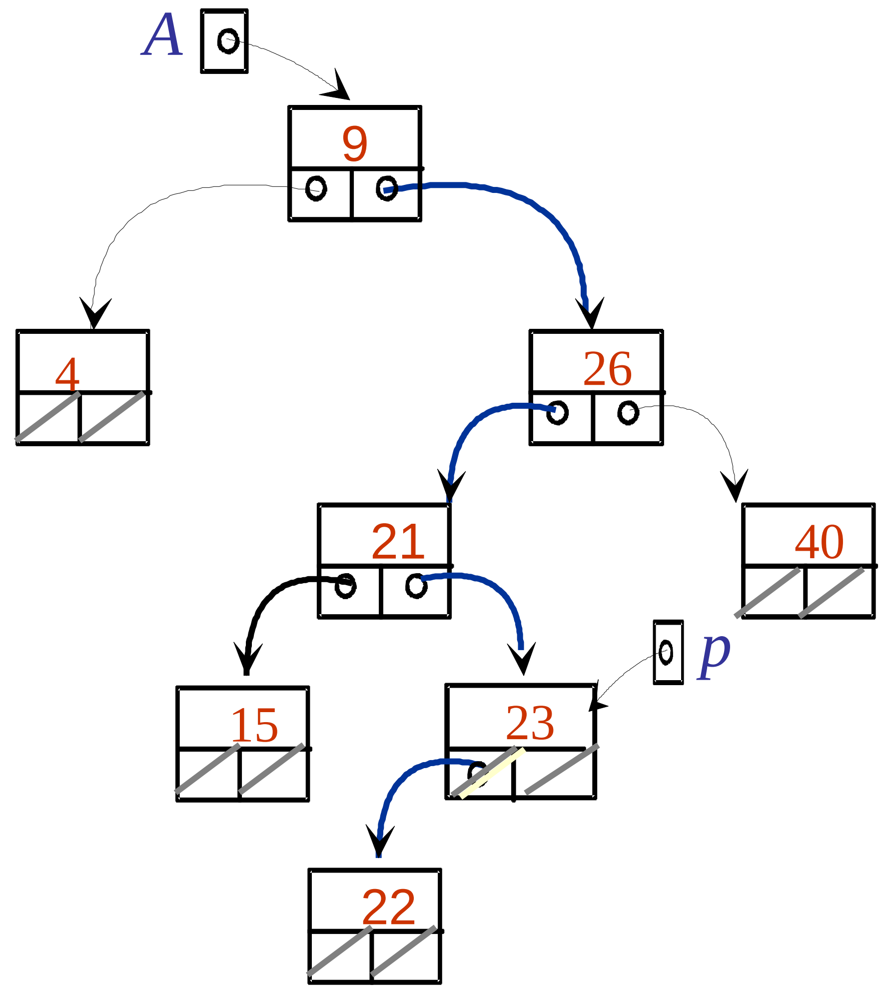
  - on cherche la clé du nœud à insérer ;
  - lorsqu'on arrive à une feuille, on ajoute le nœud comme fils de la feuille en comparant sa clé à celle de la feuille :

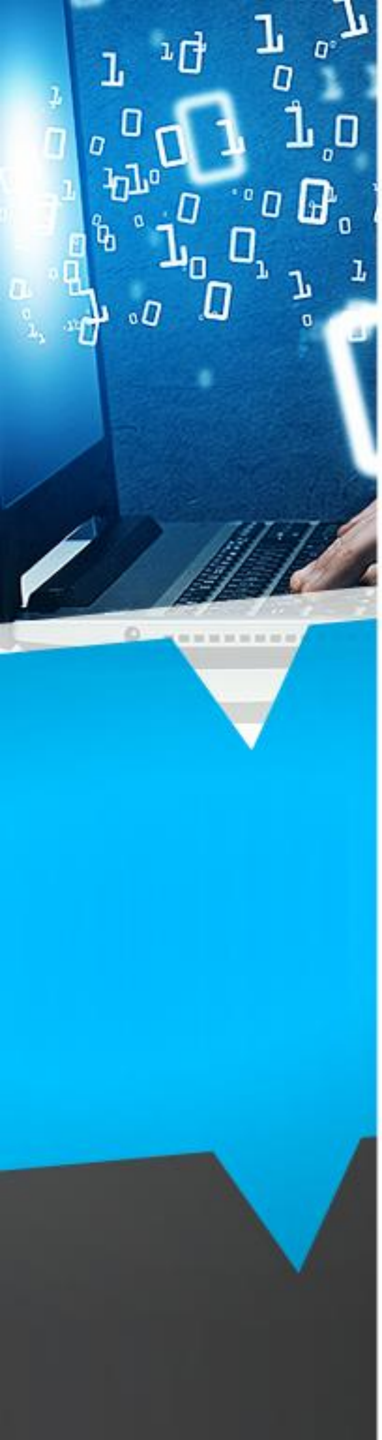
    - si elle est inférieure, le nouveau nœud sera à gauche;
    - sinon il sera à droite.

ARBRES BINAIRES DE RECHERCHE: Opération courante

- **Insertion**

cle = 22

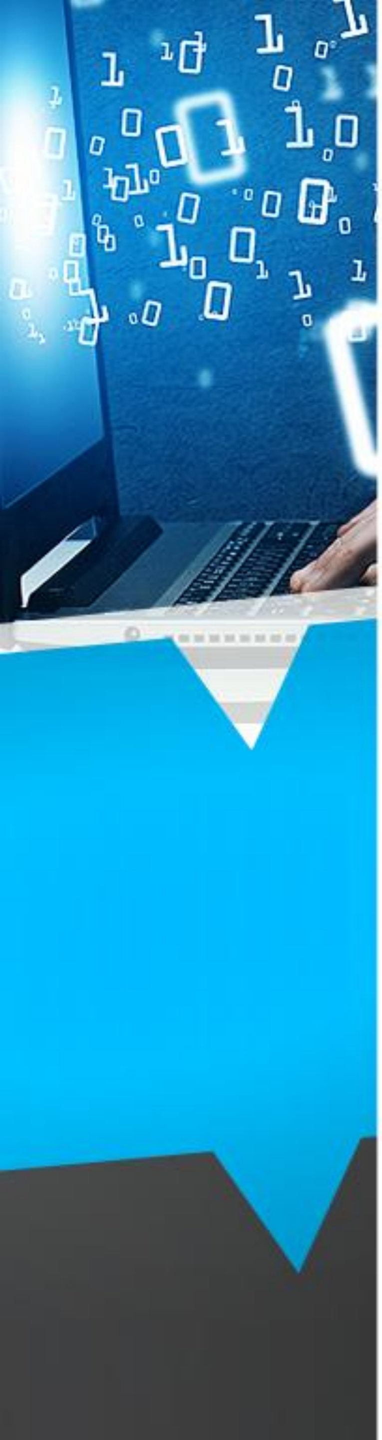




ARBRES BINAIRES DE RECHERCHE: Opération courante

- **Insertion:** ***Version Itérative***

```
procedure InsertionIterative(var a:ARbre, cle:Entier);
var p , pere:Arbre;
{
     p<- a;
   pere <- Null;
     Tant que (p<> Null) faire
              pere <- p;
              Si (cle <= p^.info) Alors
                    p <- p^.fg;
              Sinon
                 p <- p^.fd;
            Fin_Si
    Fin_Tant_QUe
      Allouer(p);
      p^.info <-cle;
      p^.fg <- Null;
      p^.fd <- Null;
     Si (pere =Null) Alors
         a <- p
     Sinon Si  (cle <=pere^.info)Alors
             pere^.fg <- p;
              Sinon
              pere^.fd <- p;
}
```



ARBRES BINAIRES DE RECHERCHE: Opération courante

- **Insertion:** ***Version Récursive***

```
procedure InsertionRecursive (var a:Arbre, cle:TypeElement
{
    Si (a =Null)  Alors
                    allouer(a);
                    a^.info <- cle;
                    a^.fg <- Null ;
                    a^.fd <- Null;
     Sinon si (cle <= a^.info) Alors
                     InsertionRecursive (a^.fg, cle);
            Sinon
                 InsertionRecursive (a^.fd, cle);
            Fin_Si
     Fin_Si
}
```

ARBRES BINAIRES DE RECHERCHE: Opération courante

- **Suppression**

- On commence par rechercher la clé du noeud à supprimer dans l'arbre. Plusieurs cas sont à considérer, une fois que le nœud à supprimer a été trouvé à partir de sa clé :

  Cas1. Suppression d'une feuille : Il suffit de l'enlever de l'arbre puisqu'elle n'a pas de fils.

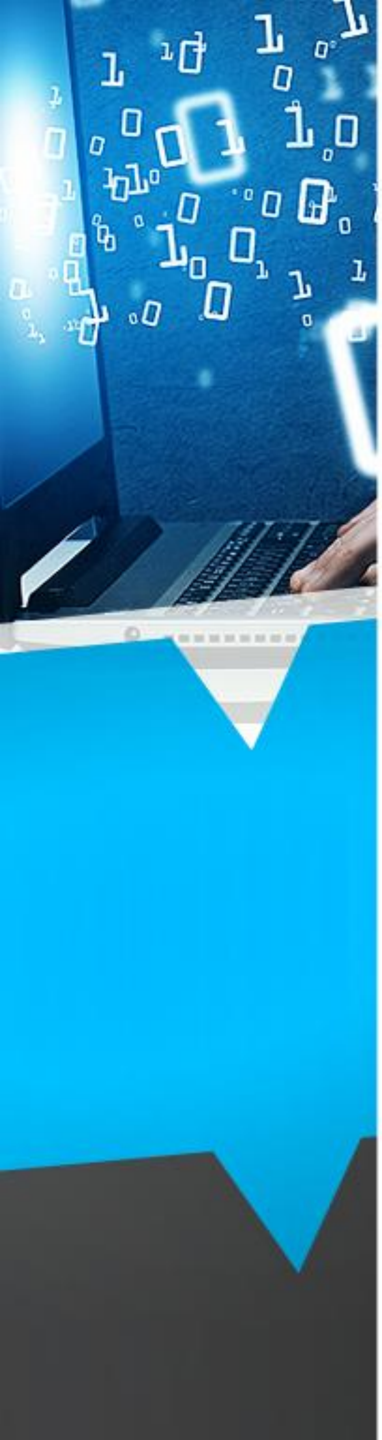
  Cas2. Suppression d'un nœud avec un enfant : Il faut l'enlever de l'arbre en le remplaçant par son fils.

ARBRES BINAIRES DE RECHERCHE: Opération courante

- **Suppression**

- On commence par rechercher la clé du noeud à supprimer dans l'arbre. Plusieurs cas sont à considérer, une fois que le nœud à supprimer a été trouvé à partir de sa clé :

  Cas3. Suppression d'un nœud avec deux enfants : Supposons que le nœud à supprimer soit appelé *N.* On échange le nœud N avec son successeur le plus proche (le nœud le plus à gauche du sous-arbre droit) ou son plus proche prédécesseur (le nœud le plus à droite du sous-arbre gauche). Cela permet de garder à la fin de l'opération une structure d'arbre binaire de recherche. Puis on applique à nouveau la procédure de suppression à *N*, qui est maintenant une feuille ou un nœud avec un seul fils.

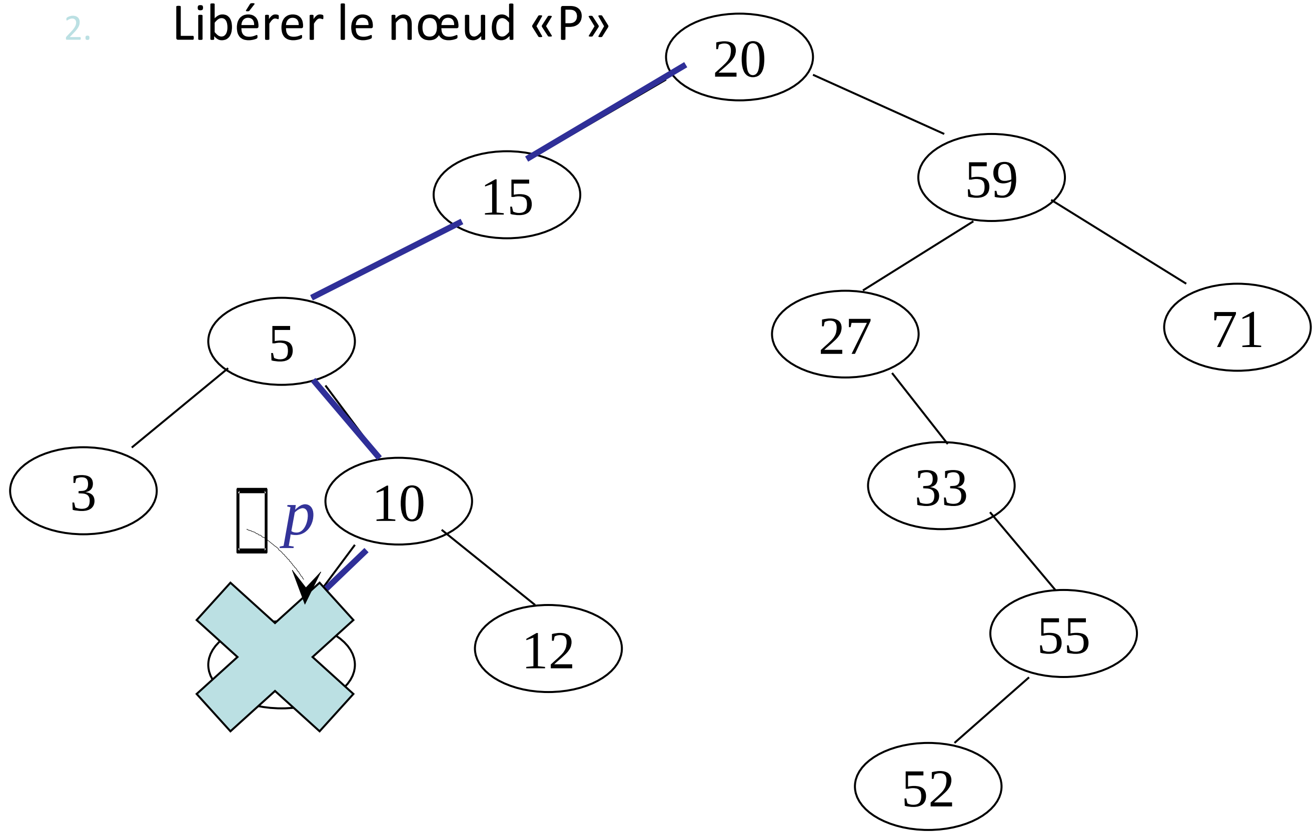


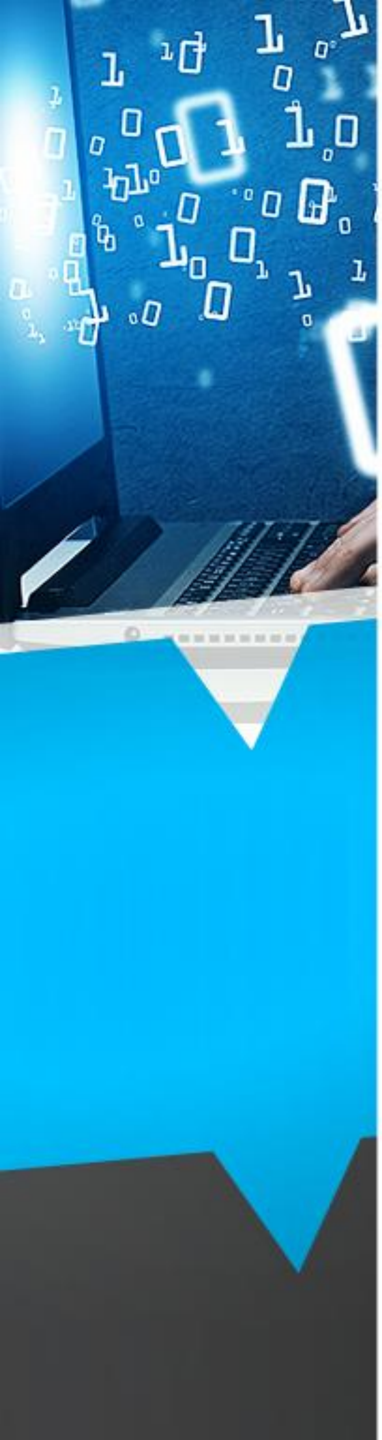
ARBRES BINAIRES DE RECHERCHE: Opération courante

- **Suppression: Cas1. Suppression d'une feuille :**

Exemple: supprimer le nœud P qui contient la valeur 8

1. Rechercher(**8**)
2. Libérer le nœud «P»



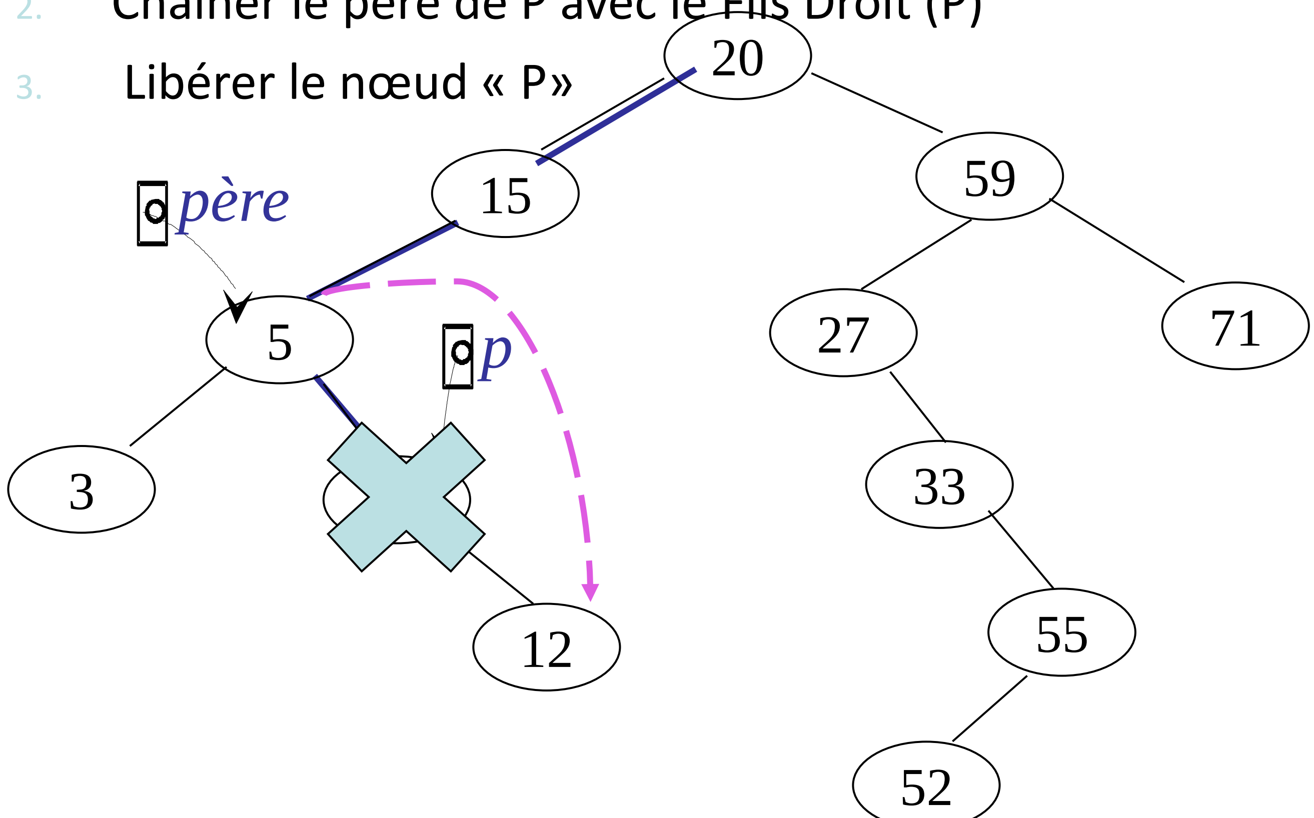


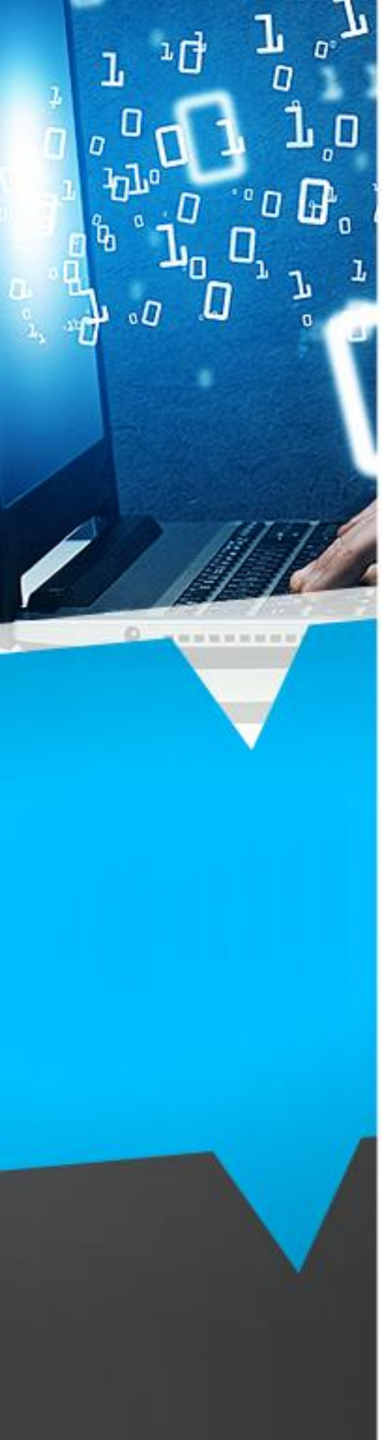
ARBRES BINAIRES DE RECHERCHE: Opération courante

- **Suppression: Cas2. Suppression d'un nœud avec un enfant**

Exemple: supprimer le nœud P qui contient la valeur 10

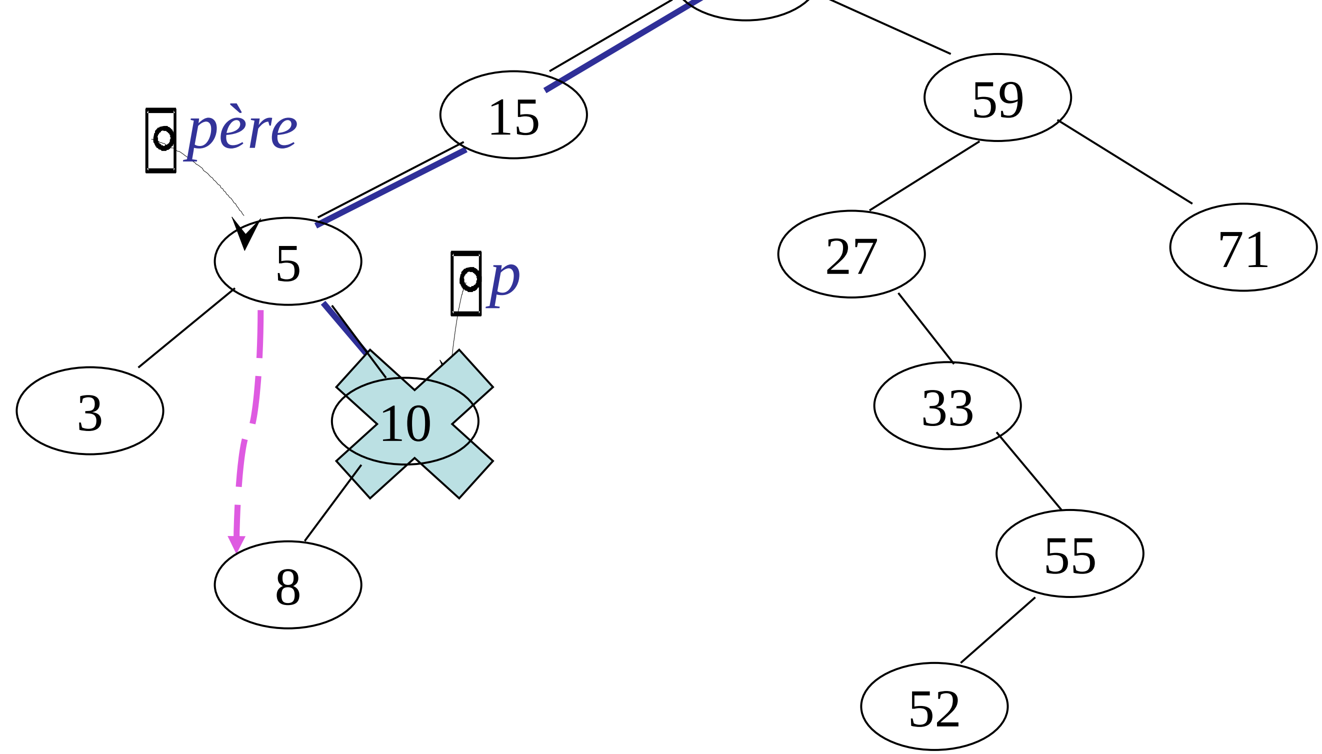
1. Rechercher(**10**)
2. Chainer le père de P avec le Fils Droit (P)
3. Libérer le nœud « P»





ARBRES BINAIRES DE RECHERCHE: Opération courante

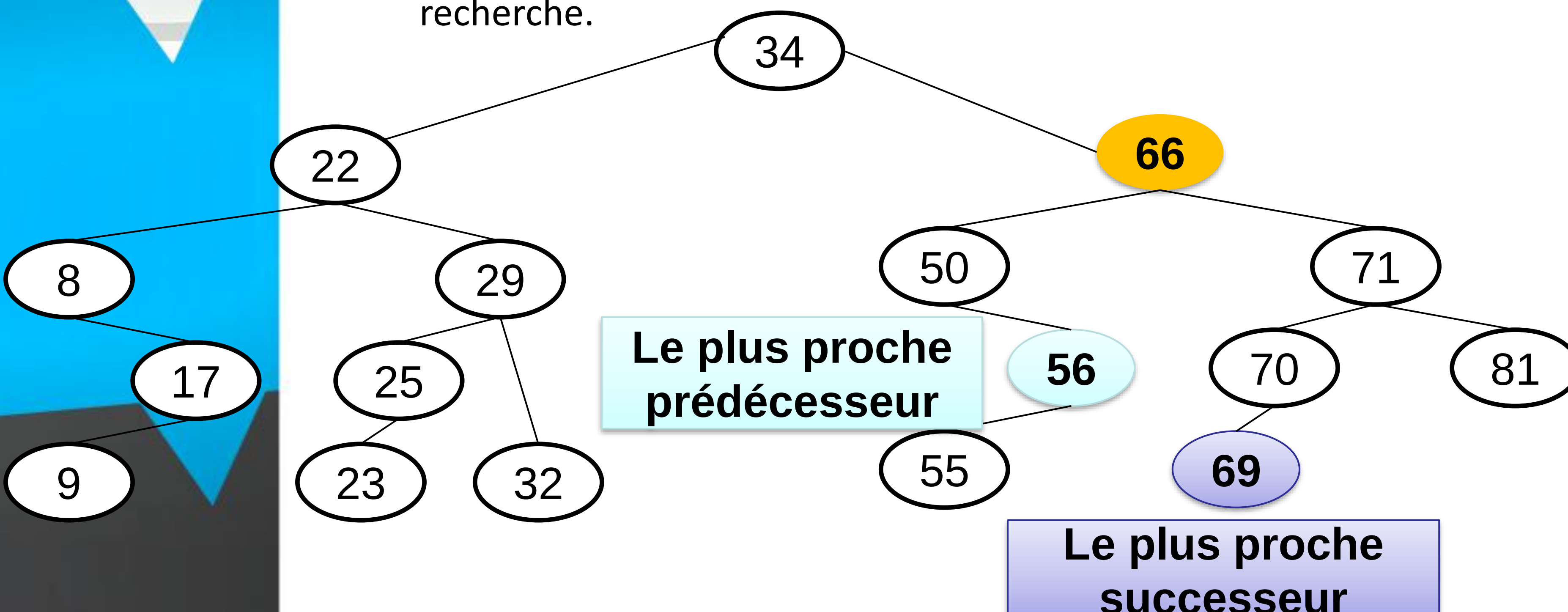
- **Suppression: Cas2. Suppression d'un nœud avec un enfant**
- **Exemple:** supprimer le nœud P qui contient la valeur 10
    1. Rechercher(**10**)
    2. Chainer le père de P avec le Fils Droit (P)
    3. Libérer le nœud « P»

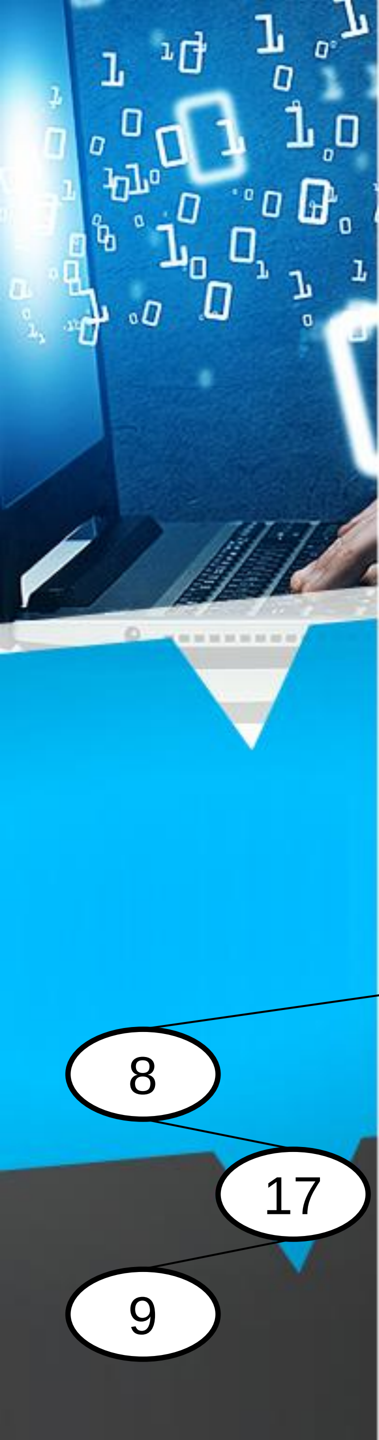


ARBRES BINAIRES DE RECHERCHE:

Opération courante

- **Suppression: Cas3. Suppression d'une feuille :**
  - **Etape 1:** On échange le nœud à supprimer avec **son successeur le plus proche** (le nœud le plus à gauche du sous-arbre droit) ou **son plus proche prédécesseur** (le nœud le plus à droite du sous-arbre gauche). Cela permet de garder une structure d'arbre binaire de recherche.





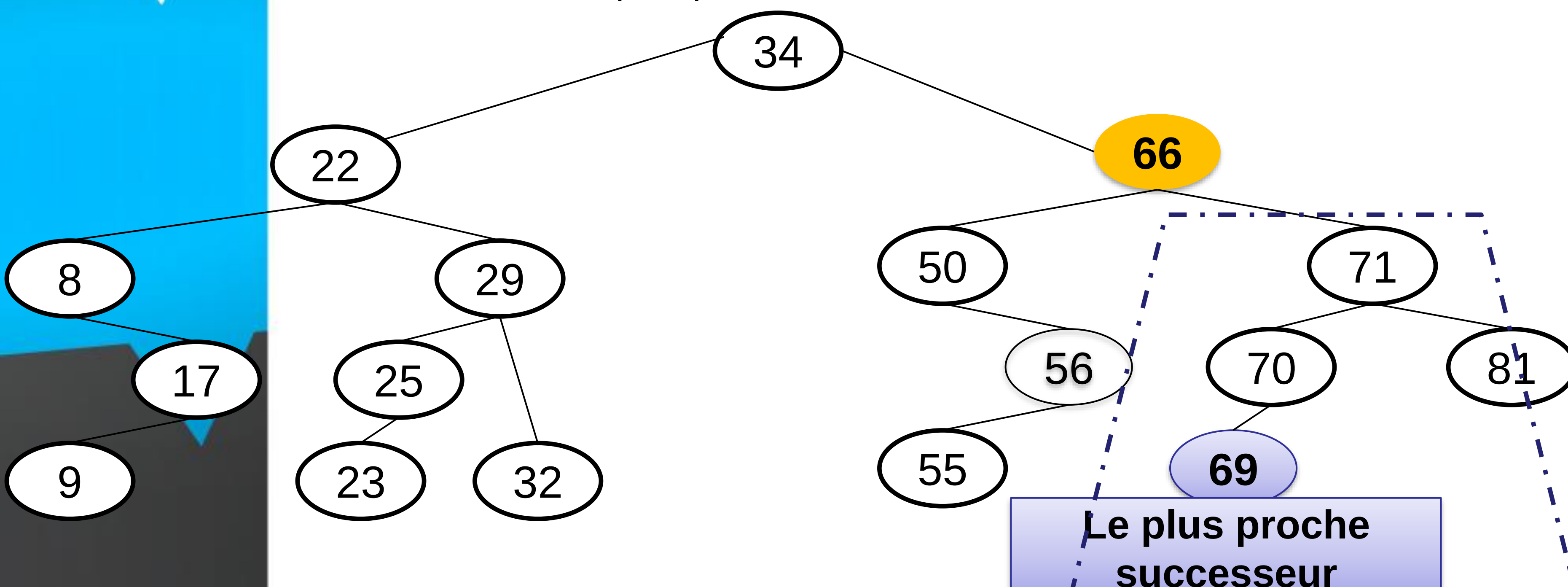
ARBRES BINAIRES DE RECHERCHE:

Opération courante

- **Suppression: Cas3. Suppression d'une feuille :**

  - **Etape 1 → Cas A:** On échange le nœud à supprimer avec **son successeur le plus proche** (le nœud le plus à gauche ou le plus petit du sous-arbre)

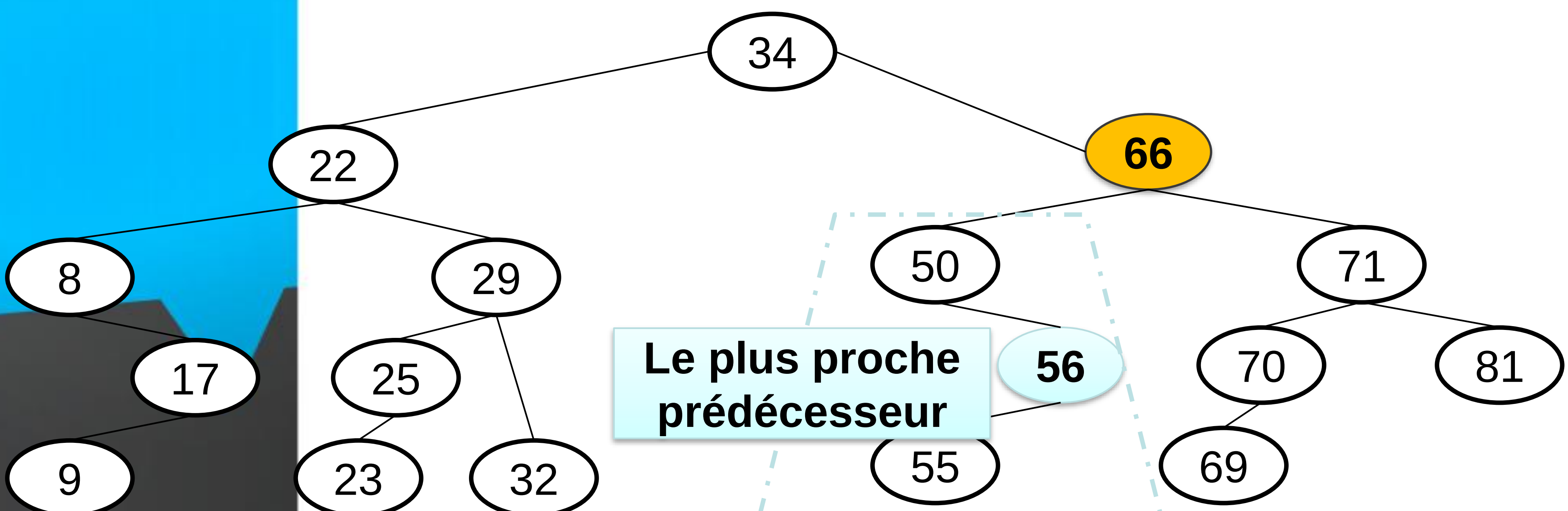
    - Racine: 71
    - La plus petite valeur : 69



ARBRES BINAIRES DE RECHERCHE:

Opération courante

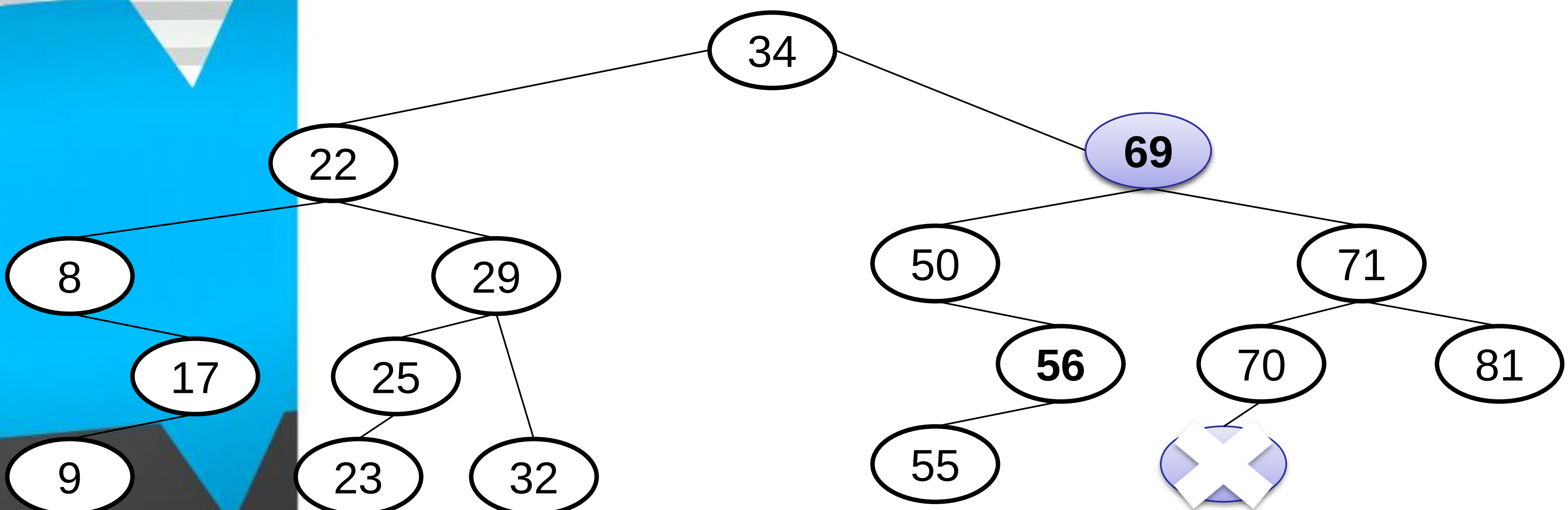
- **Suppression: Cas3. Suppression d'une feuille :**
  - **Etape 1 → Cas A:** On échange le nœud à supprimer avec **son successeur le plus proche** (le nœud le plus à gauche ou le plus petit du sous-arbre)
    - Racine: 50
    - La plus petite valeur : 56



ARBRES BINAIRES DE RECHERCHE:

Opération courante

- **Suppression: Cas3. Suppression d'une feuille :**
  - **Etape 2:** on applique à nouveau la procédure de suppression qui est maintenant une feuille ou un nœud avec un seul fils.
  - Ainsi, si on choisit d'échanger le nœud « 66 » avec son plus proche successeur « 69 », on obtient

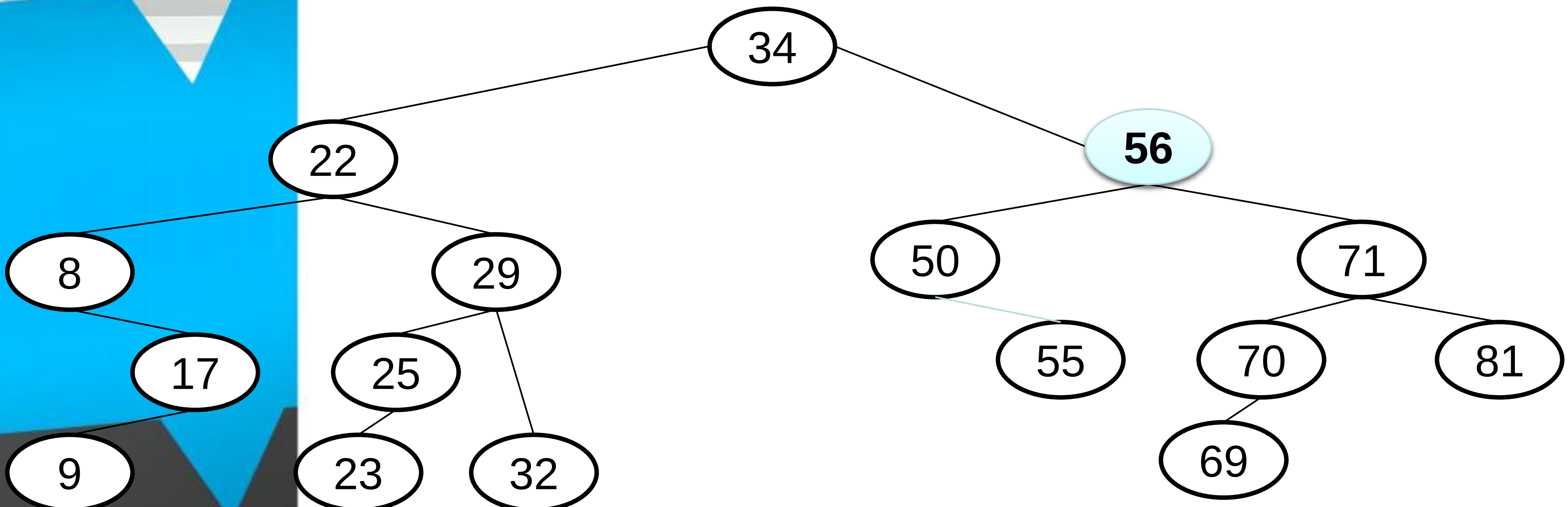


ARBRES BINAIRES DE RECHERCHE:

Opération courante

- **Suppression: Cas3. Suppression d'une feuille :**

  - Puis on applique à nouveau la procédure de suppression qui est maintenant une feuille ou un nœud avec un seul fils.
  - Ainsi, si on choisit d' échanger le nœud « 66 » avec son plus proche prédécesseur « 56 », on obtient



ARBRES BINAIRES DE RECHERCHE:

Opération courante

- **Exercice :**

1. Soit R un arbre binaire de recherche (ABR)

   a. Construire cet arbre partir des valeurs suivantes :

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

   b. Ajouter à l'arbre obtenu et dans l'ordre, les éléments suivants : 22, 62, 64, 4, 8

   c. **Supprimer de l'arbre obtenu et dans l'ordre les éléments suivants : 15, 70, 50, 35, 60, 25**