yncréa

ISEN école d'ingénieurs
ALL IS DIGITAL!

Algorithmique: Advanced Data Structures

Chapitre 11 :

- **LES ARBRES**

yncréa

ISEN école d'ingénieurs
ALL IS DIGITAL!

LES ARBRES

Trees

yncréa

ISEN école d'ingénieurs
ALL IS DIGITAL!

Sommaire

Les Arbres

- Ajout d'éléments dans un ABR
- Suppression d'un élément dans un ABR

- Parcours en Largeur
- Parcours en Hauteur

yncréa

ISEN école d'ingénieurs
ALL IS DIGITAL!

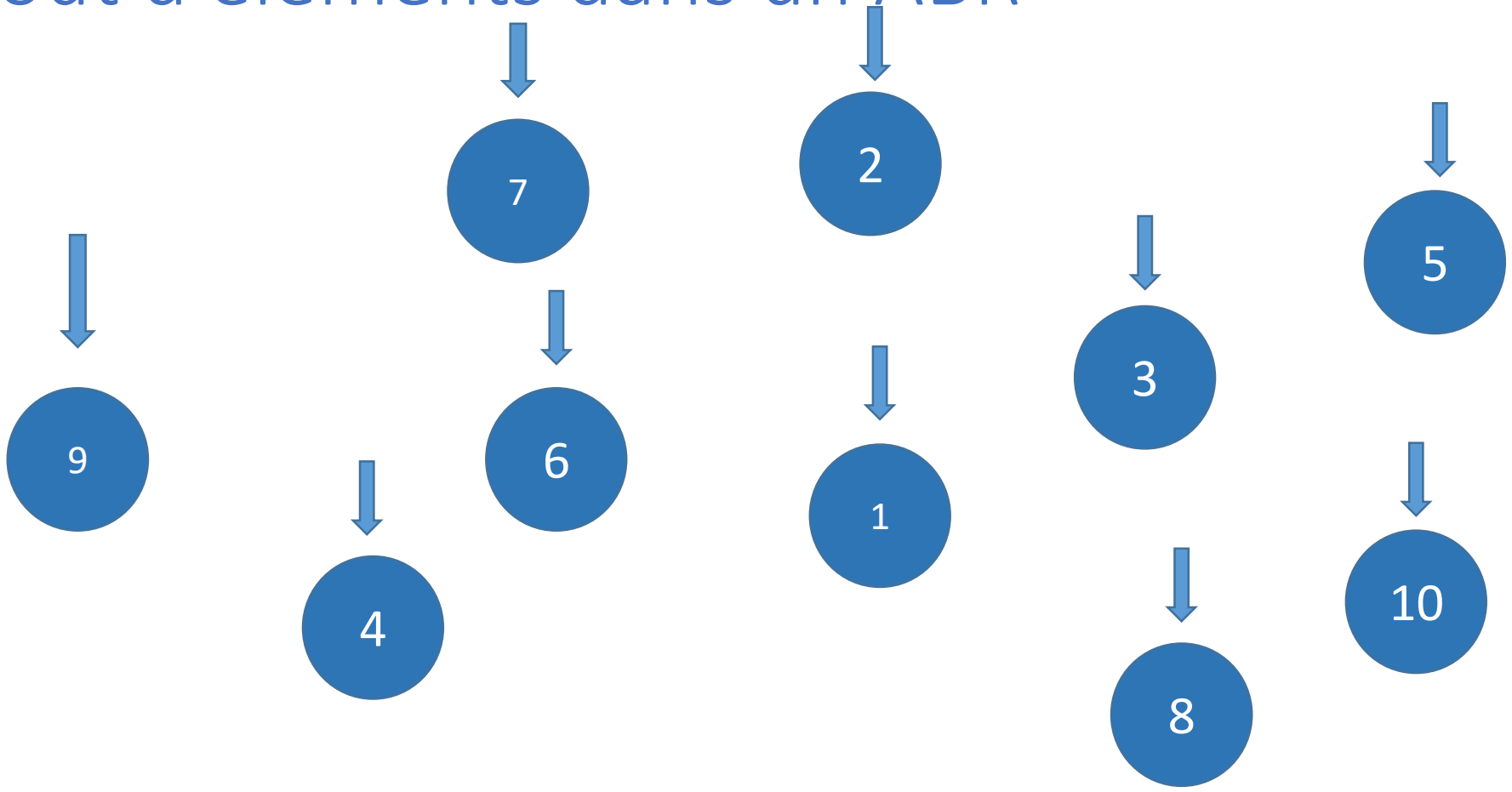
Sommaire

Les Arbres

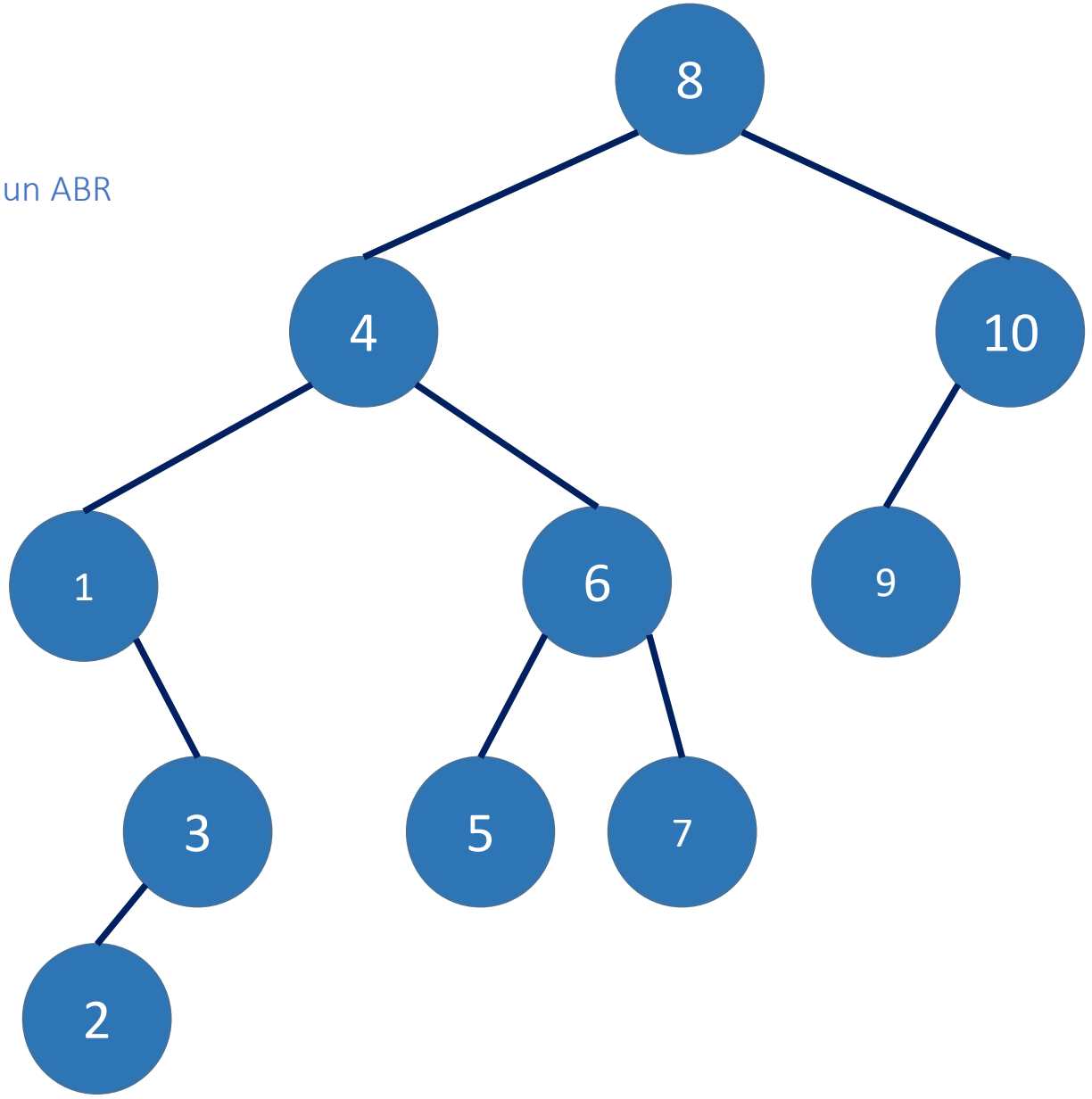
- **Ajout d'éléments dans un ABR**
- Suppression d'un élément dans un ABR

- Parcours en Largeur
- Parcours en Hauteur

Ajout d'éléments dans un ABR



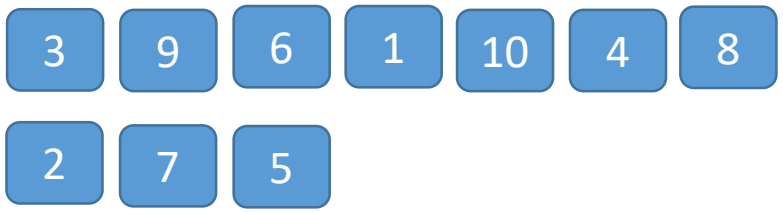
Ajout d'éléments dans un ABR

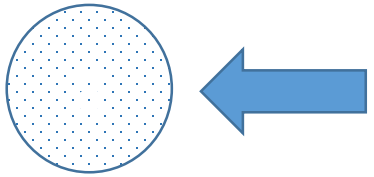


Profondeur = 4

Nombre de nœuds = 10

Ajout d'éléments dans un ABR

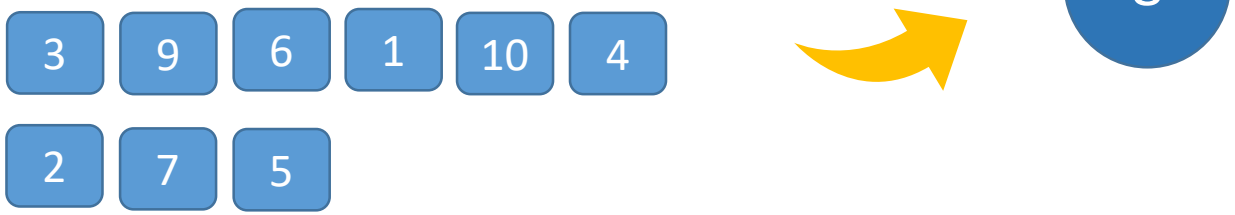


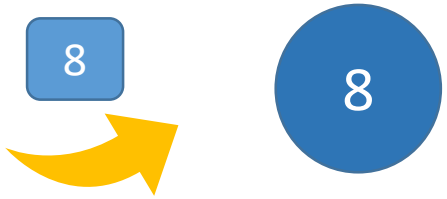


Curseur

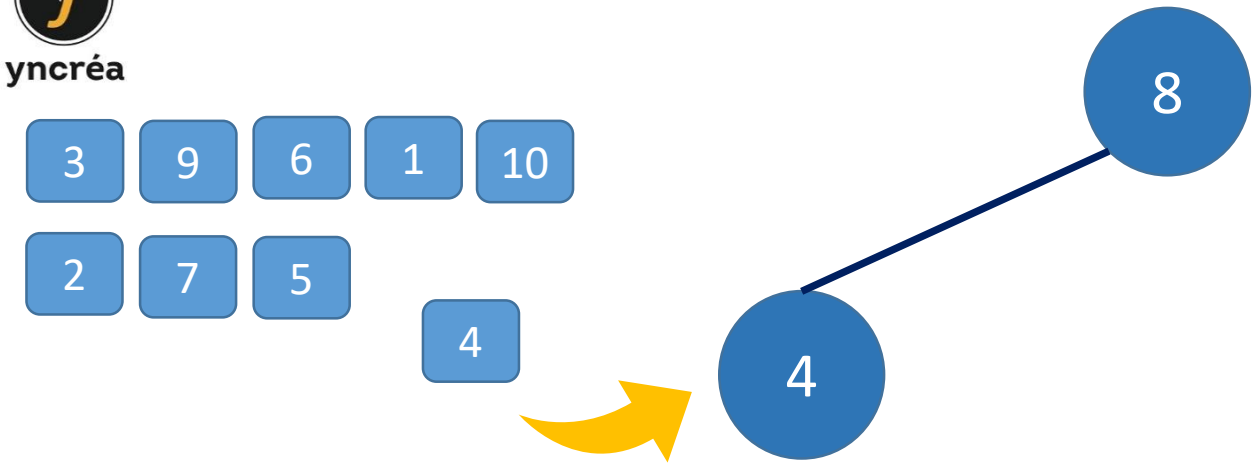
```
C <- ObtenirUnCurseurSurLaRacine( monArbre )

Tant Que ( NON (estEnfin( C )) ET ( valeur <> obtenirValeurNoeud( C ) )
Faire
        Si ( valeur < obtenirValeurNoeud( C ) )
        Alors
                descendre( C , gauche )
        Sinon
                descendre( C, droite )
        FinSi
Fait

Si  estEnfin( C )
Alors
   insérerValeurDansArbreA( C , valeur)
FinSi
```

AjouterValeurDans ArbreABR

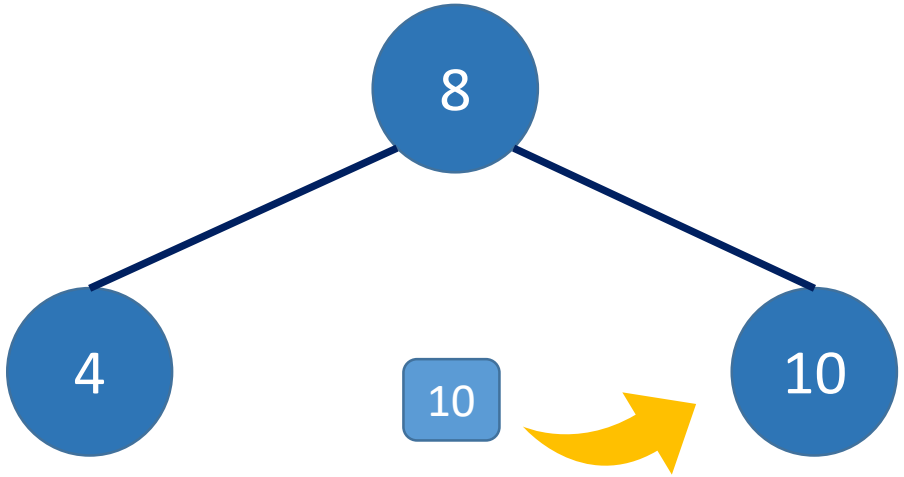

3
9
6
1
10
4
2
7
5

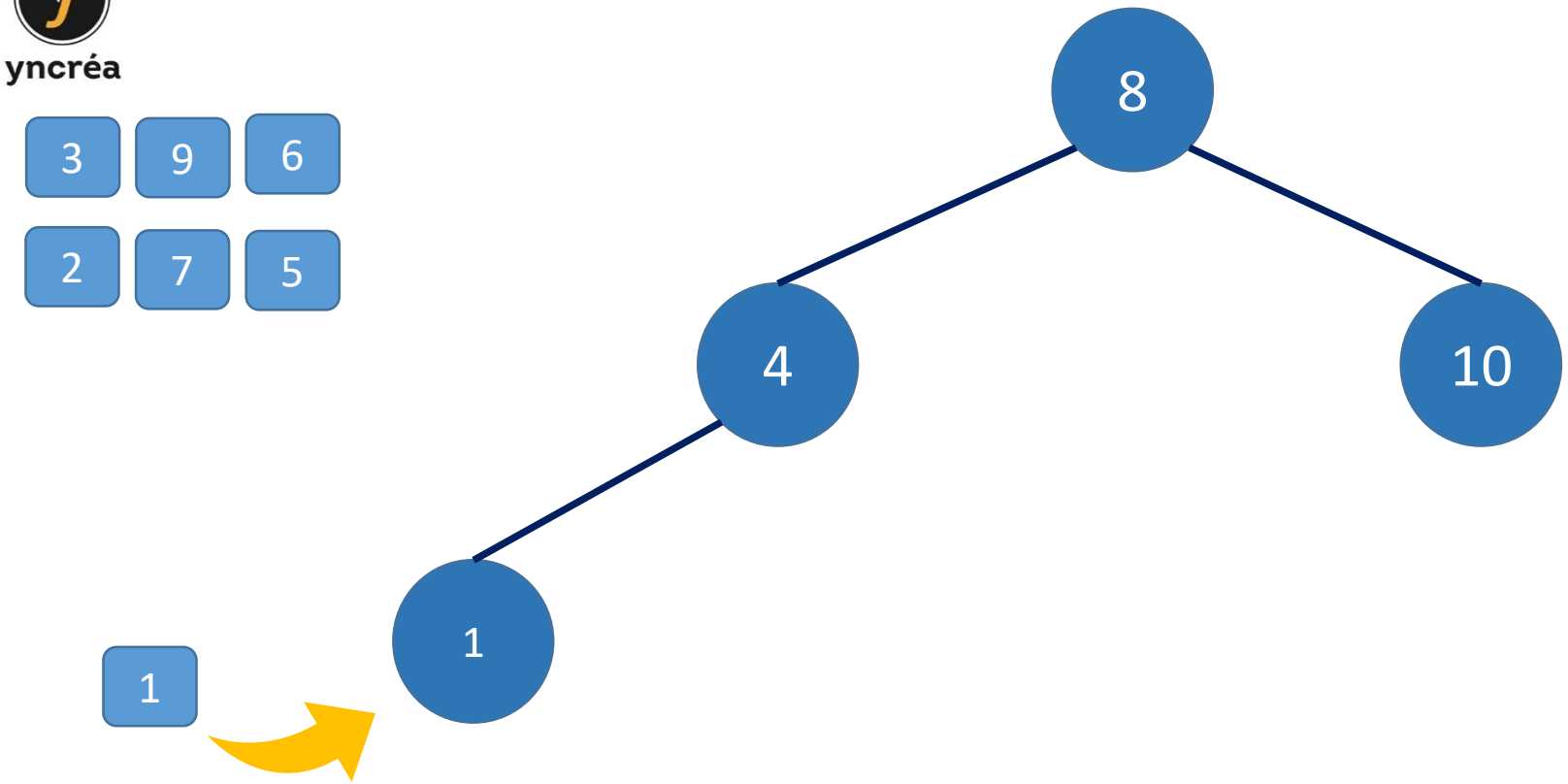

8

8


3
9
6
1
10
2
7
5
4
8
4

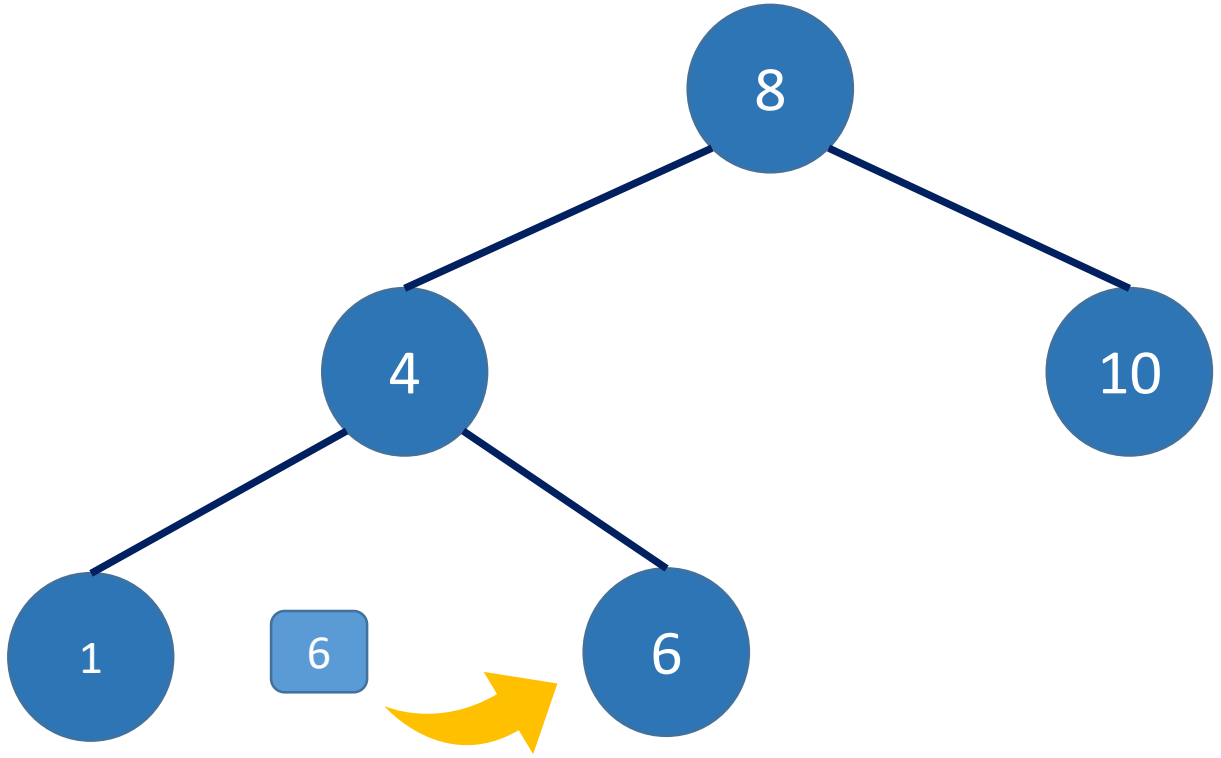

3
9
6
1
2
7
5


8
4
10
10

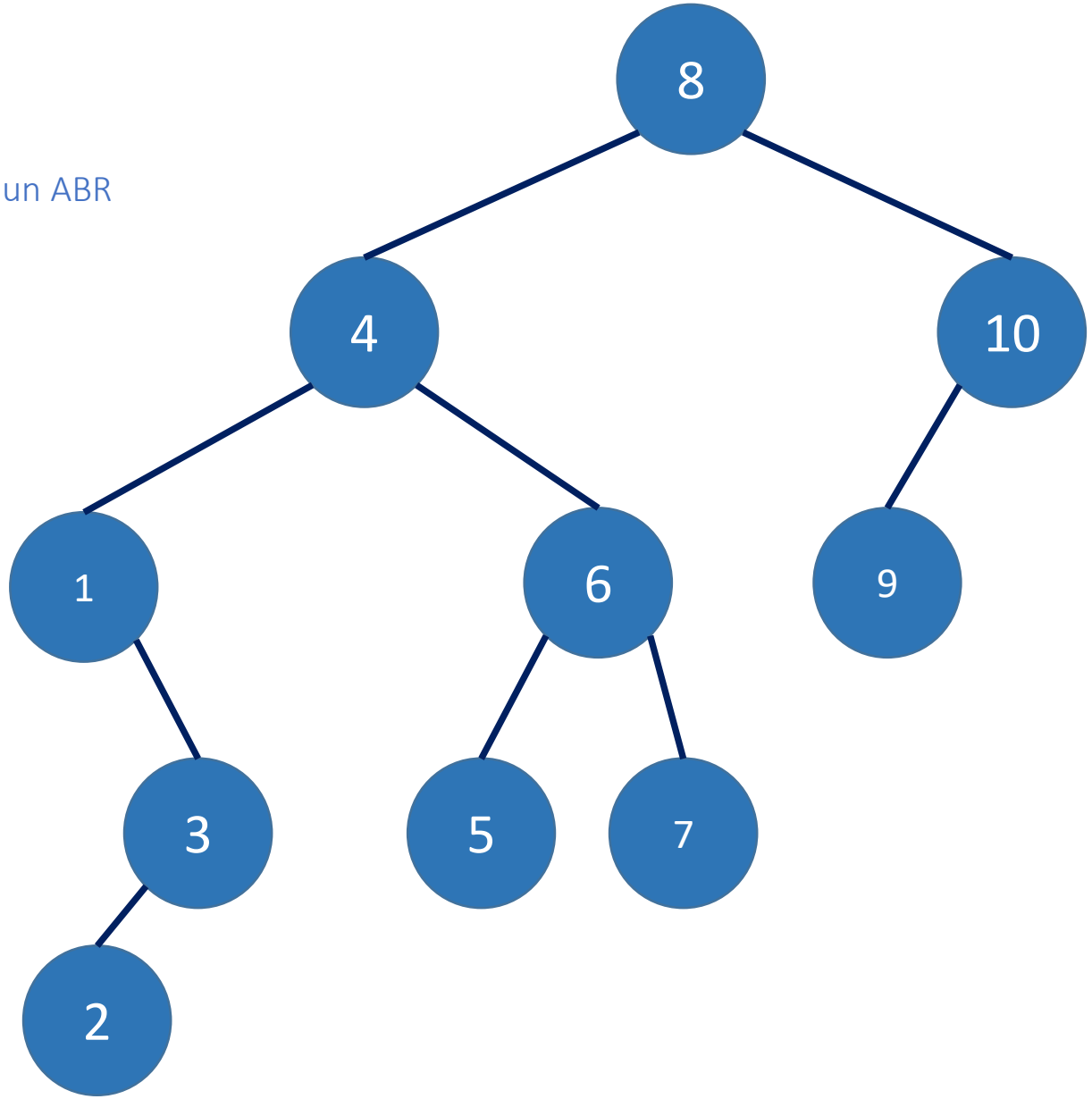

3
9
6
2
7
5
8
4
10
1
1

3 9

2 7 5



Ajout d'éléments dans un ABR



yncréa

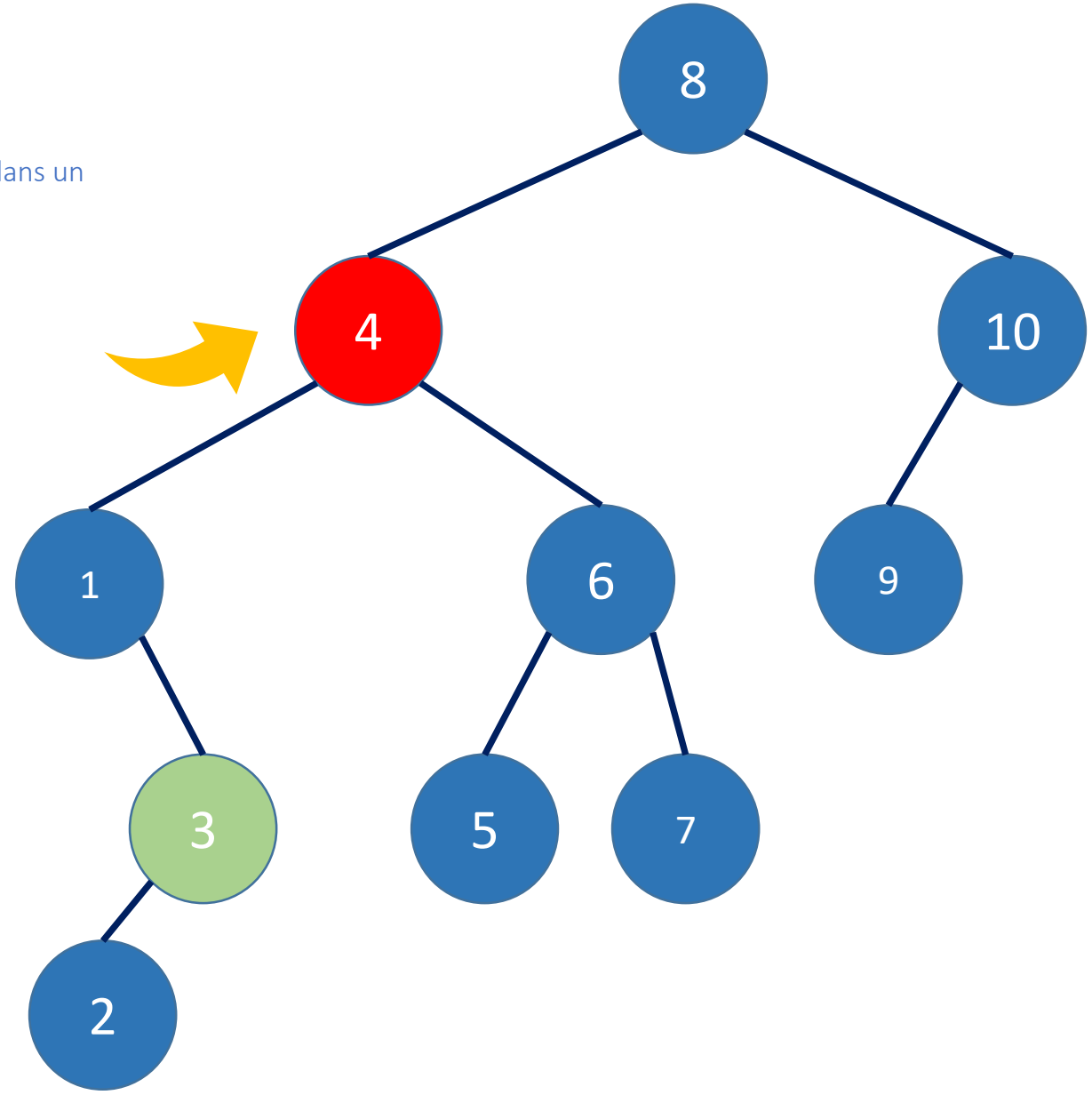
ISEN école d'ingénieurs
ALL IS DIGITAL!

Sommaire

Les Arbres

- Ajout d'éléments dans un ABR
- **Suppression d'un élément dans un ABR**

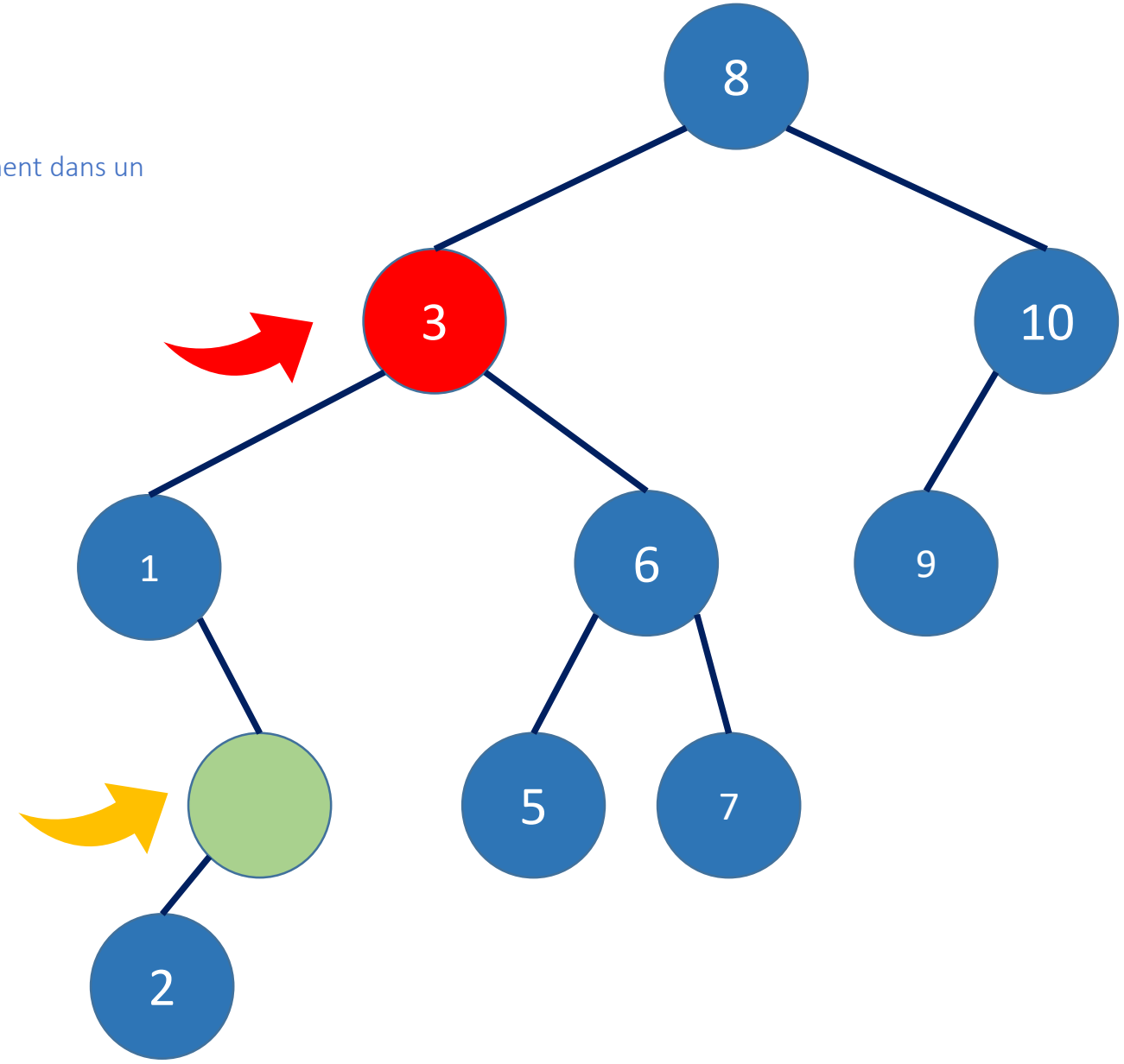
- Parcours en Largeur
- Parcours en Hauteur



Suppression d'une valeur dans l'arbre ABR :

Cette méthode utilise une astuce qui facilite les opérations.

Il s'agit dans un premier temps de remplacer la valeur à supprimer par la plus grande des valeurs du sous-arbre **gauche**.

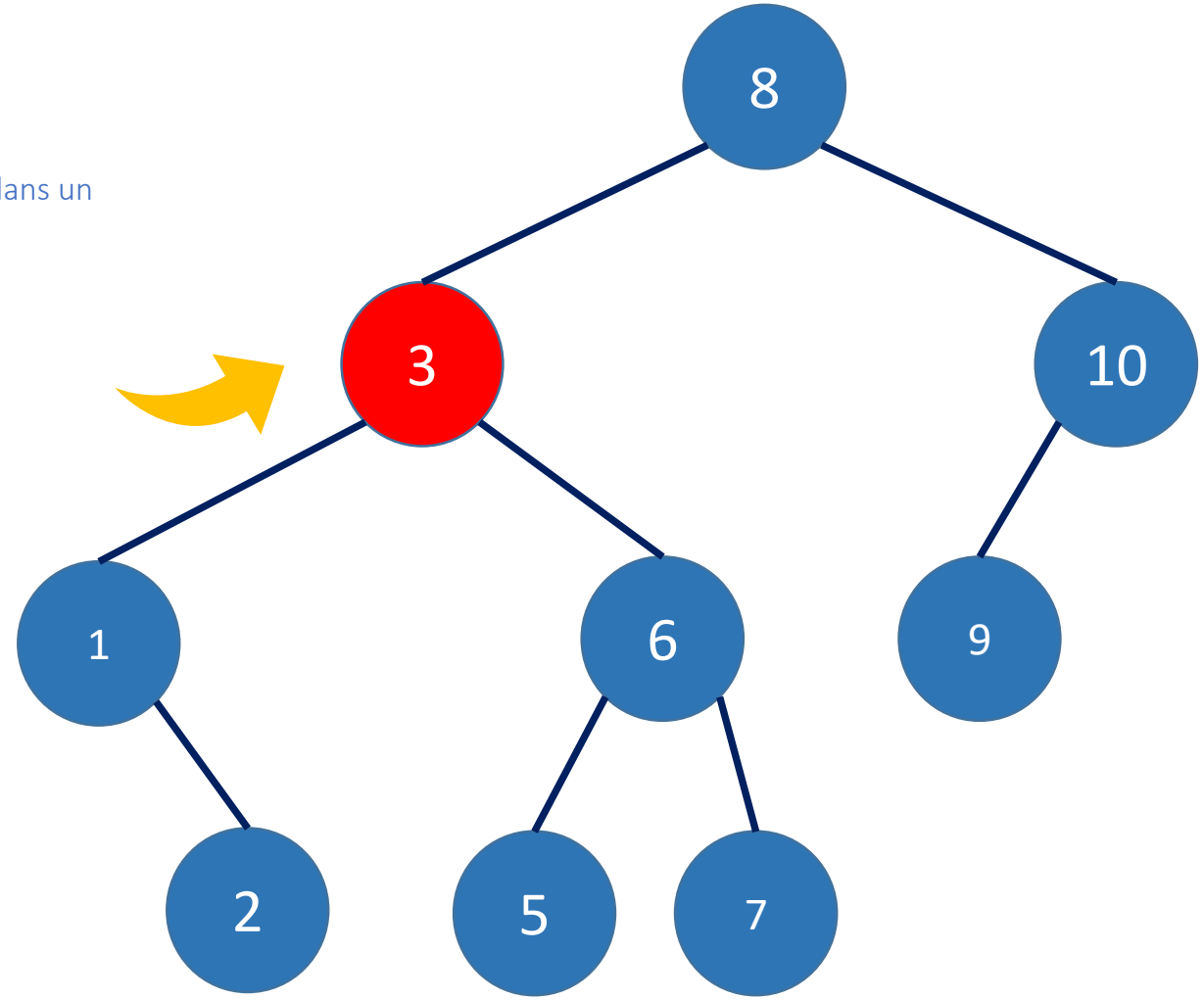


Suppression d'une valeur dans l'arbre ABR :

Cette méthode utilise une astuce qui facilite les opérations.

Il s'agit dans un premier temps de remplacer la valeur à supprimer par la plus grande des valeurs du sous-arbre **gauche**.

Puis dans un second temps, de supprimer le nœud vidé



Suppression d'une valeur dans l'arbre ABR :

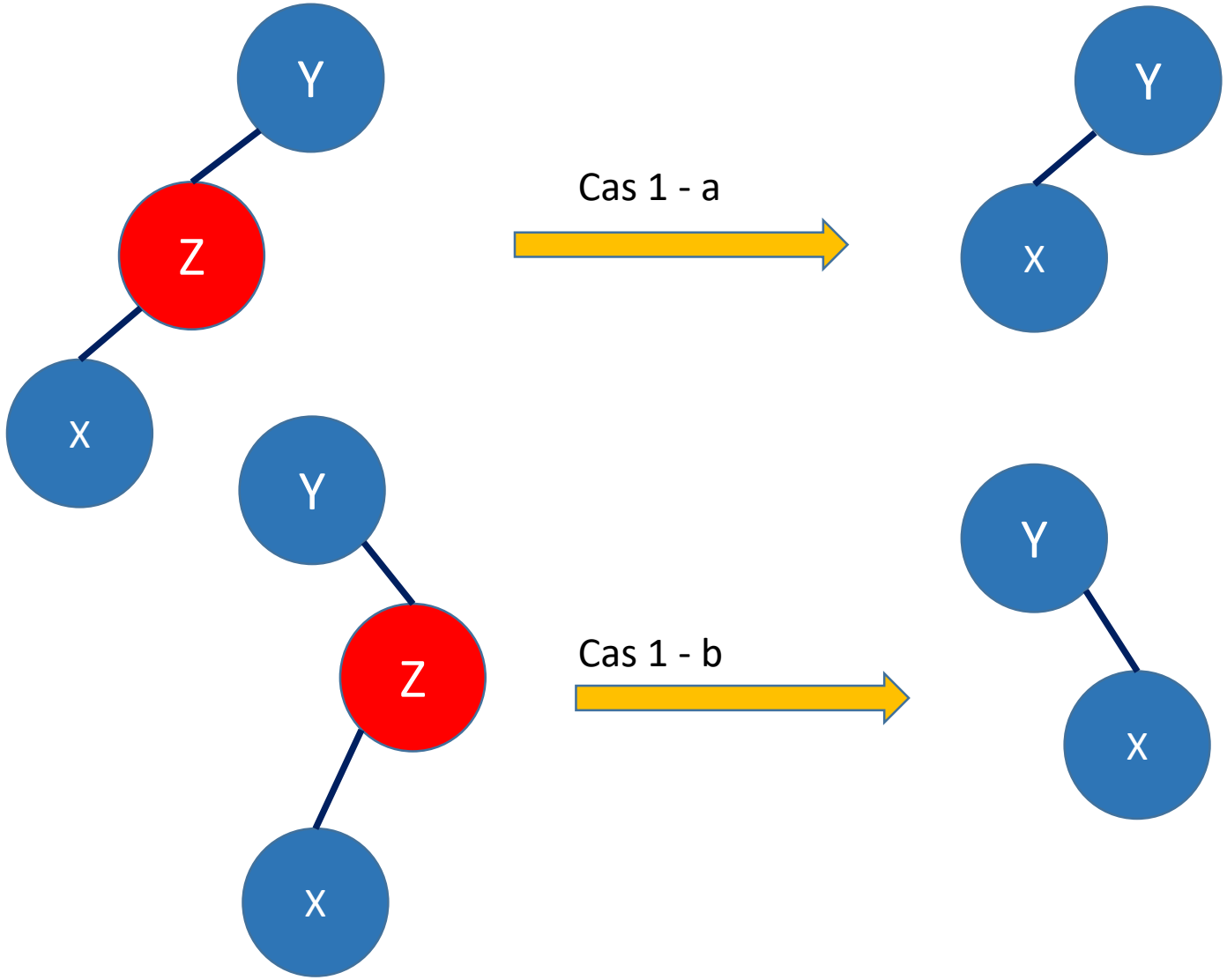
Cette méthode utilise une astuce qui facilite les opérations.

Il s'agit dans un premier temps de **remplacer la valeur** à supprimer par la plus grande des valeurs du **sous-arbre gauche**.

Puis dans un second temps, de **supprimer** le nœud vidé.

```
C <- ObtenirUnCurseurSurLaRacine( monArbre )


// Recherche le nœud qui contient la valeur à supprimer de l'arbre


Tant Que ( NON (estEnfin( C )) ET ( valeur <> obtenirValeurNoeud( C ) )
Faire
        Si ( valeur < obtenirValeurNoeud( C ) )
        Alors
                descendre( C , gauche )
        Sinon
                descendre( C, droite )
        FinSi
Fait
```

```
// Si on a trouvé la valeur
Si  NON estEnfin( C )
Alors
      // Dans le cas d'un nœud qui possède deux fils
     Si  possedeFils( C, gauche ) && possedeFils( C, droite )
     Alors
          // recherche de la plus grande valeur du sous-arbre gauche
          D <- C   // mémorise le nœud à l'embranchement à l'aide d'un nouveau curseur D
         Tant Que possedeFils( C, droite)
         Faire
                  descendre(C, droite)
         Fait
         modifierValeur(D, C) // on la place dans nœud (D)
     FinSi
     // Dans tous les cas on supprime le nœud désigné par C
     suppressionNoeud( C )
FinSi
```

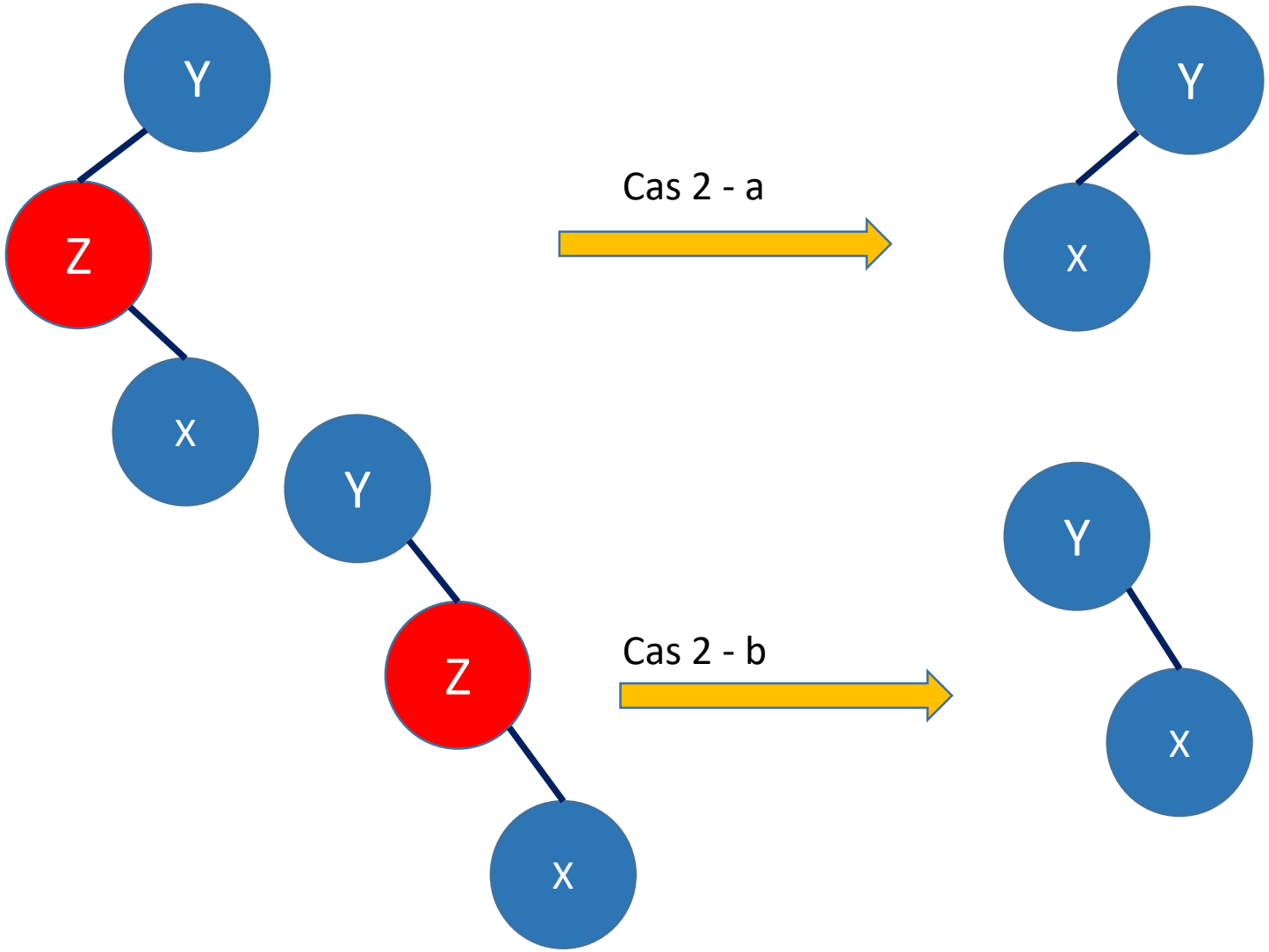


Suppression d'un nœud dans un arbre :

On distingue quatre cas simples pour lesquels le nœud à supprimer n'a qu'un fils.

Cas 1 – a et b

Le nœud à supprimer à un fils gauche

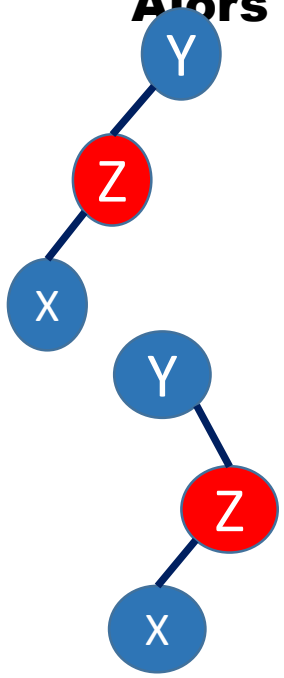


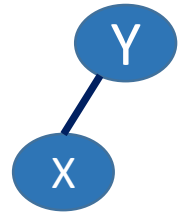
Suppression d'un nœud dans un arbre :

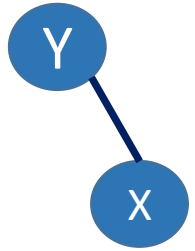
Cas 2 – a et b

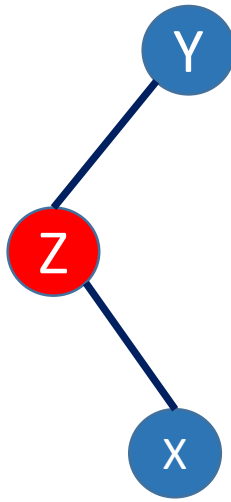
Le nœud à supprimer à un fils **droit**

```
// Etape II : Suppression du noeud:

// si le noeud à supprimer a un fils gauche : cas 1
// alors on attache ce fils gauche sur le noeud parent
// pour faire monter la branche gauche
Si Node->filsGauche <> AUCUNE_VALEUR
Alors
        Si Curs.side = gauche
        Alors                                   // par la gauche (1 – a)
                Curs.pointeur->filsGauche <- Node->filsGauche
        Sinon
                Si Curs.side = droite
                Alors                           // par la droite (1 – b )
                        Curs.pointeur->filsDroit <- Node->filsGauche
                FinSi
        FinSi
```

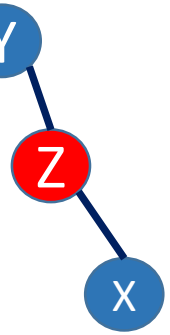


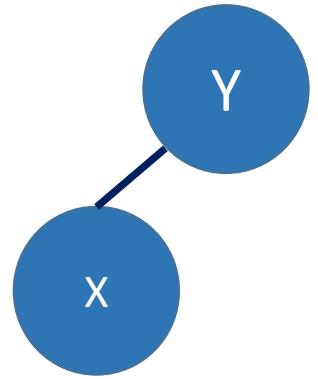


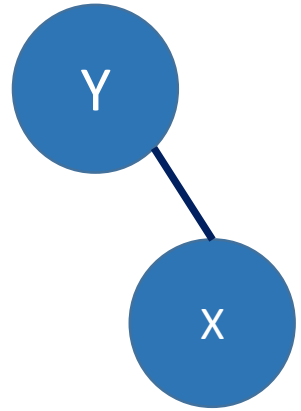





```
// Etape II : Suppression du noeud:

// sinon, le nœud à supprimer a un fils droit : cas 2
// on attache ce fils droit sur le nœud parent
// pour faire monter la branche droite
Sinon
        Si Curs.side = gauche
        Alors                                   // par la gauche (2 – a)
                Curs.pointeur->filsGauche <- Node->filsDroit
        Sinon
                Si Curs.side = droite
                Alors                           // par la droite (2 – b )
                        Curs.pointeur->filsDroit <- Node->filsDroit
                FinSi
        FinSi
FinSi
libérerMémoire( Node )
```







yncréa

ISEN école d'ingénieurs
ALL IS DIGITAL!

Sommaire

Les Arbres

- Ajout d'éléments dans un ABR
- Suppression d'un élément dans un ABR

- **Parcours en Largeur**
- Parcours en Profondeur

Parcours en largeur

Fonction **itérative** qui va calculer le nombre de nœuds de l'arbre :

- On réalise un parcours en largeur en s'aidant d'une file
- On compte les nœuds rencontrés.

- On peut faire autre chose que compter les nœuds comme par exemple afficher les nœuds. Ce qui nous amène à utiliser **un pointeur de fonction** qui va fournir en paramètre l'outil qu'on souhaite utiliser

```
// parcours itératif en largeur de l'arbre en utilisant une file
// passage d'un pointeur de fonction en paramètre pour le traitement de la valeur des
noeuds visités
int breadthFirstTraversal(BinaryTree tree, int (*doSomething)(BinaryTreeElement *oneNode))
{
#define QUEUESIZE 1000
if (doSomething == NULL) return(EXIT_FAILURE);

Queue *file = NULL;
// création d'une file pour le parcours en largeur
NewQueue(&file, QUEUESIZE);

Cursor curs;
// placement d'un curseur à la racine de l'arbre
curs = getCursorOnRoot(tree);

BinaryTreeElement * node;
// obtention du noeud à la racine
node = getNode(curs);
// mise en attente du noeud dans la file
queue(file, (void *)node);

while (!isQueueEmpty(file)) { // tant qu'il y a des noeud à traiter

deQueue(file, (void *)&node);// récupération du noeud en attente
doSomething(node);// réaliser le traitement choisi
// pousser le fils gauche puis le fils droit dans la file d'attente
if (node->leftChild!=NULL) queue(file, (void*)(node->leftChild));
if (node->rightChild!=NULL) queue(file, (void*)(node->rightChild));
}
return(EXIT_SUCCESS);
}
```

Détermination du nombres de nœuds dans un arbre binaire :

```
// Calcule le nombre de noeuds dans un arbre
int getBinaryTreeNodesCount(BinaryTree tree) {
// attention nodeCount est une variable globale.
nodeCount = 0;
breadthFirstTraversal(tree, countOneNode);
return(tree.nodeCount = nodeCount);

}
```

Fonction de comptage :

```
// ajoute la valeur 1 au nombre de noeuds
int countOneNode(BinaryTreeElement *oneNode) {
return(nodeCount++);
}
```

Affichage des valeurs contenues dans l'arbre binaire :

```
printf("Parcours itératif de l'arbre binaire en largeur, à l'aide d'une file
: \n");

breadthFirstTraversal(*Arbre, displayBinaryTreeNodeValue);
```

Fonction d'affichage de la valeur contenue dans un nœud :

```
// Affiche la valeur contenue dans le champs data d'un noeud
int displayBinaryTreeNodeValue(BinaryTreeElement *oneNode) {
if (oneNode == NULL) {
        return(-1);
}
printf("%d ", oneNode->data);

return(0);
}
```

yncréa

ISEN école d'ingénieurs
ALL IS DIGITAL!

Sommaire

Les Arbres

- Ajout d'éléments dans un ABR
- Suppression d'un élément dans un ABR

- Parcours en Largeur
- **Parcours en Profondeur**

yncréa

ISEN école d'ingénieurs
ALL IS DIGITAL!

Parcours en profondeur

Exercice :

Ecrire une fonction **récursive** qui va mesurer la profondeur de l'arbre.

yncréa

ISEN école d'ingénieurs
ALL IS DIGITAL!

Parcours en profondeur

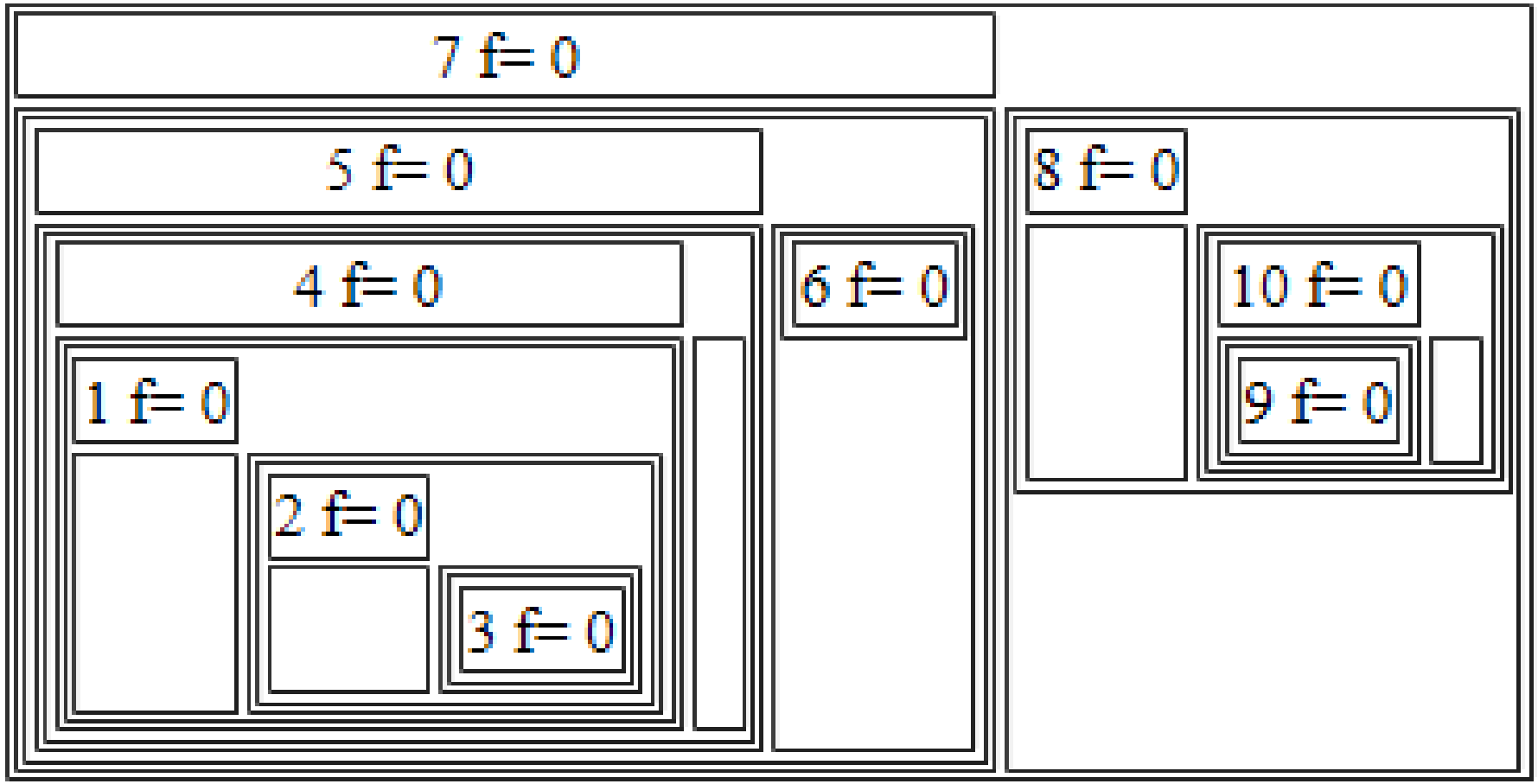
Fonction récursive qui va mesurer la profondeur de l'arbre :

- On parcours les différents chemins depuis la racine jusqu'à une feuille de l'arbre en comptant les nœuds traversés.
- La hauteur de l'arbre correspond au plus grand nombre de nœuds calculé

```
Fonction obtenirProfondeurSousArbre( C, n, max) ne renvoie rien

Déclaration paramètres
DEBUT                   C en Cursor, // curseur sur la racine du sous-arbre
                        n en entier,  // accumulateur -> nombre de nœuds parcourus
                        max en pointeur d'entier // la plus grande profondeur atteinte
FIN
Déclaration variables
DEBUT                   X en Cursor // X permet de se déplacer dans les sous-arbres
FIN
DEBUT
            Cursor X;

            // parcours du sous-arbre gauche
            SI ( C possèdeUnFils à gauche )
            ALORS
            // le curseur X prend la valeur de la racine du sous arbre
                        X = C;                                          // backtracking
                        descendreCurseur X à gauche;
                        obtenirProfondeurSousArbre(X, n + 1, max);
            FINSI

            // parcours du sous-arbre droit
            SI (C possèdeUnFils à droite )
            ALORS
                        X = C;                                          // backtracking
                        descendreCurseur X à droite;
                        obtenirProfondeurSousArbre(X, n + 1, max);
            FINSI
            // on garde le max de n comme étant le plus grand nombre de nœuds parcourus
            SI (n > *max) ALORS *max = n FSI

FIN Fonction obtenirProfondeurSousArbre
```

```
// Calcul de la profondeur d'un arbre : WRAPPER

int obtenirProfondeur(Arbre) renvoie un entier
Déclaration paramètres
DEBUT                Arbre en arbre binaire, // curseur sur la racine du sous-arbre
FIN
Déclaration variables
DEBUT      C en Curseur // Curseur pour se déplacer dans l'arbre
           n en entier,  // accumulateur -> nombre de nœuds parcourus
           profondeur en entier // profondeur atteinte

FIN
DEBUT
           int n = 0;
           int depth = 0;
           Placer curseur C à la racine de Arbre;

           SI (curseur C n'est pas en fin)  // on peut descendre
           ALORS      obtenirProfondeurSousArbre (C, n, &profondeur);
                      renvoyer profondeur;
           FINSI
           renvoyer 0; // pas de sous-niveau
FIN fonction obtenirProfondeur
```

```
1 2 3 4 5 6 7 8 9 10
7 8 5 10 4 1 6 2 9 3

la profondeur de l arbre est égale à = 5

Parcours itératif de l'arbre binaire en largeur, à l'aide d'une file :
7 5 8 4 6 10 1 9 2 3
L'arbre contient 10 noeuds.

Parcours itératif de l'arbre binaire en profondeur, à l'aide d'une pile :
7 5 4 1 2 3 6 8 10 9
Parcours récursif de l'arbre binaire en profondeur avec affichage du chemin :
---1
---+2
---++3
--4
-5
-+6
7
+8
++-9
++10

Parcours récursif de l'arbre binaire en profondeur avec génération de balise html:

<html><body><table border=1> <tr> <td align=center>7 f= 0</td> </tr> <tr valign=top> <td align=center><table border=1> <tr
> <td align=center>5 f= 0</td> </tr> <tr valign=top> <td align=center><table border=1> <tr> <td align=center>4 f= 0</td> <
/tr> <tr valign=top> <td align=center><table border=1> <tr> <td align=center>1 f= 0</td> </tr> <tr valign=top><td width=10
mm></td> <td align=center><table border=1> <tr> <td align=center>2 f= 0</td> </tr> <tr valign=top><td width=10mm></td> <td
 align=center><table border=1> <tr> <td align=center>3 f= 0</td> </tr></table> </td></tr></table> </td></tr></table> </td>
<td width=10mm></td></tr></table> </td> <td align=center><table border=1> <tr> <td align=center>6 f= 0</td> </tr></table>
</td></tr></table> </td> <td align=center><table border=1> <tr> <td align=center>8 f= 0</td> </tr> <tr valign=top><td widt
h=10mm></td> <td align=center><table border=1> <tr> <td align=center>10 f= 0</td> </tr> <tr valign=top> <td align=center><
table border=1> <tr> <td align=center>9 f= 0</td> </tr></table> </td><td width=10mm></td></tr></table> </td></tr></table>
</td></tr></table></body></html>
```


7 f= 0
5 f= 0
4 f= 0
1 f= 0
2 f= 0
3 f= 0
6 f= 0
8 f= 0
10 f= 0
9 f= 0