

# Les Arbres Binaires de Recherche

Sandrine Vial  
`sandrine.vial@uvsq.fr`

Novembre 2020

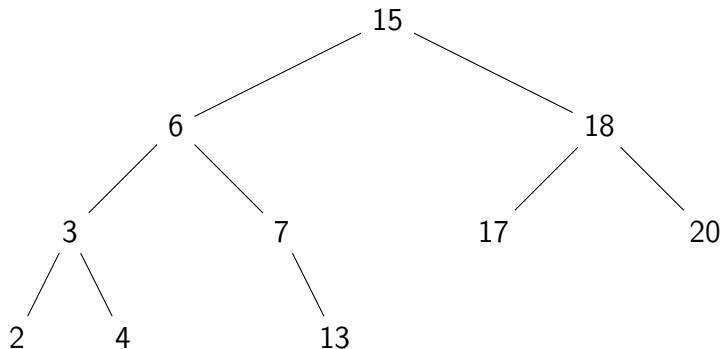
# Les arbres binaires de recherche

1. Arbre binaire enraciné
2. Chaque nœud d'un ABR est associé à une valeur. Les valeurs doivent être comparables entre elles.
3. Propriété supplémentaire

# Propriétés

1. La valeur d'un nœud est **plus grande** que toutes les valeurs de **son sous-arbre gauche**.
2. La valeur d'un nœud est **plus petite** que toutes les valeurs de **son sous-arbre droit**.

## Exemple d'ABR



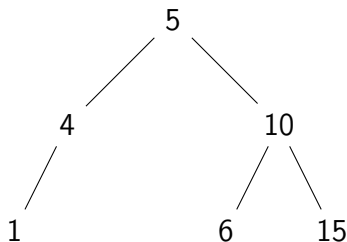
# Structure de données

## Mise en œuvre

```
Enregistrement Nœud {  
    val      : entier;  
    sag : ↑ Nœud;  
    sad : ↑ Nœud;  
    parent : ↑ Nœud;  
}
```

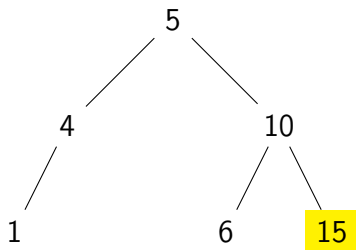
# Affichage des valeurs des nœuds

- Affichage dans l'ordre croissant des clés :  
Parcours en profondeur infixe



# Recherche de la valeur maximum

- ▶ La valeur maximum se trouve dans **le sous-arbre droit de la racine**.
- ▶ Il faut suivre **le bord droit** de l'arbre pour arriver à la valeur maximum.



# Recherche de la valeur maximum

---

## Algorithme 1 Element Maximum dans un ABR

---

ABR\_Max( $r$  : Nœud) : Entier

▷ *Entrée* :  $r$  (la racine d'un arbre)

▷ *Sortie* : l'élément maximum de l'ABR enraciné en  $r$

Debut

    tant que  $x.sad \neq \text{NIL}$  faire

$x \leftarrow x.sad$ ;

    fin tant que

    retourner  $x.val$ ;

Fin

Complexité au pire :  $O(\text{hauteur de l'ABR})$



# Recherche de la valeur minimum

---

## Algorithme 2 Element Minimum dans un ABR

---

ABR\_Min( $r$  : Nœud) : Entier

▷ *Entrée* :  $r$  (la racine d'un arbre)

▷ *Sortie* : l'élément maximum de l'ABR enraciné en  $r$

Debut

    tant que  $x.sag \neq \text{NIL}$  faire

$x \leftarrow x.sag$ ;

    fin tant que

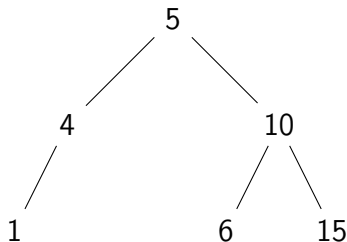
    retourner  $x.val$ ;

Fin

Complexité au pire :  $O(\text{hauteur de l'ABR})$

# Recherche d'un élément dans un ABR

- Parcours de l'arbre depuis la racine jusqu'à l'élément cherché en choisissant en chaque noeud dans quel sous-arbre on va chercher en fonction de la valeur cherchée.



# Recherche d'un élément dans un ABR

---

## Algorithme 3 Recherche d'un élément dans un ABR

---

Recherche( $r$  : Nœud,  $c$  : Entier) : Booleen

▷ *Entrée* :  $r$  (la racine d'un arbre),  $c$  (l'élément recherché)

▷ *Sortie* : renvoie vrai si  $c$  est dans l'arbre enraciné en  $r$ , faux sinon

Debut

  si  $r \neq \text{NIL}$

    si  $r.\text{val} = c$  retourner Vrai;

    sinon si  $r.\text{val} > c$  retourner Recherche( $r.\text{sag}, c$ );

    sinon retourner Recherche( $r.\text{sad}, c$ );

  fin si

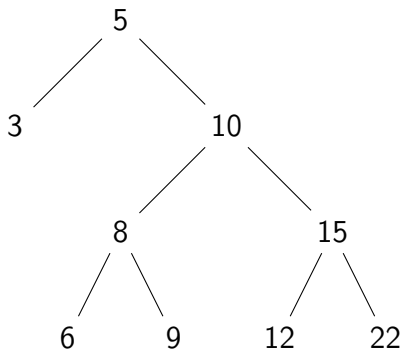
  retourner Faux;

Fin

Complexité au pire :  $O(\text{hauteur de l'ABR})$

# Successeur

- ▶  $x$  un nœud. On cherche  $y$  tel que :
  - ▶  $y.val > x.val$
  - ▶ et tel que pour tout nœud  $z$  :
    - ▶  $z \neq x$  et  $z \neq y$
    - ▶ on n'ait pas  $y.val > z.val > x.val$



# Successeur

- ▶ Le successeur d'un nœud  $x$  est le nœud possédant la plus petite valeur dans le sous-arbre droit de  $x$  s'il en possède un.
- ▶ sinon c'est le nœud qui est le 1er ancêtre de  $x$  dont la racine du sous-arbre gauche est aussi un ancêtre de  $x$  ou  $x$  lui-même.

---

## Algorithme 4 Successeur dans un ABR

---

```
ABR_Successeur(r : Nœud) : Noeud
▷ Entrée : r (la racine d'un arbre)
▷ Sortie : renvoie le nœud dont la valeur est immédiatement supérieure à celle de r
▷ Variables locales :
    y : Noeud ;
    Debut
        si r.Droit  $\neq$  NIL
            retourner ABR_Min(r.sad) ;
        fin si
        y  $\leftarrow$  r.Pere ;
        tant que y  $\neq$  NIL et r = y.sad
            r  $\leftarrow$  y ;
            y  $\leftarrow$  y.parent ;
        fin tant que
        retourner y ;
    Fin
```

Complexité au pire :  $O(\text{hauteur de l'ABR})$

# Insertion d'un nœud dans un ABR

- ▶ On veut garder les propriétés de l'ABR.
- ▶ Deux approches :
  - ▶ Insertion aux feuilles de l'ABR
  - ▶ Insertion à la racine de l'ABR

# Insertion aux feuilles

## Principe

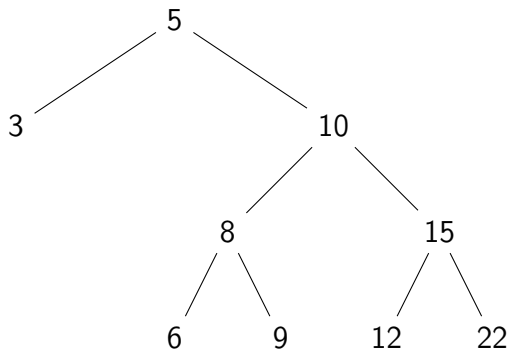
- ▶ Recherche de la valeur de l'élément que l'on cherche à insérer :
  - ▶ Si la valeur existe déjà : rien à faire
  - ▶ Sinon la recherche s'est arrêtée sur un arbre vide, qu'il suffit de remplacer par l'élément à insérer

## Conséquences

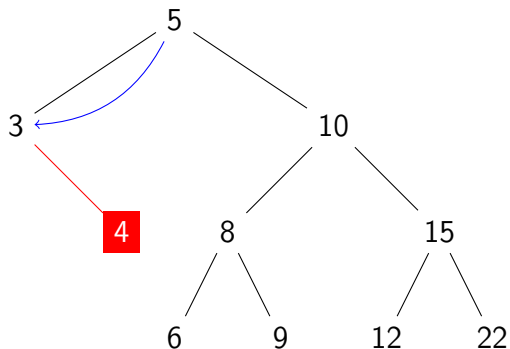
- ▶ Hauteur de l'arbre peut être modifiée.
- ▶ Complexité dans le pire cas :  $O(h)$  avec  $h$  la hauteur de l'arbre.



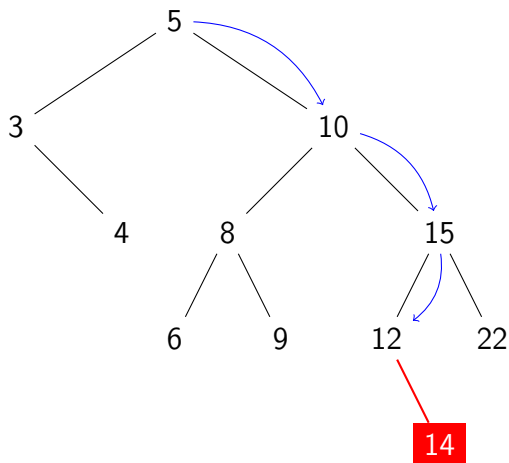
# Example



# Example



# Example



# Insertion aux feuilles

---

## Algorithme 5 Insertion d'un nœud aux feuilles

---

ABR-Inserer( $r$  : Nœud,  $z$  : Noeud) : Noeud

▷ *Entrée* :  $r$  (la racine d'un ABR),  $z$  (un nouveau à insérer)

▷ *Sortie* : renvoie la racine de l'ABR dans lequel le nœud  $z$  a été inséré

Debut

    si  $r = \text{NIL}$

        retourner  $z$  ;

    sinon

        si  $z.\text{val} \leq r.\text{val}$

$r.\text{sag} \leftarrow \text{ABR-Inserer}(r.\text{sag}, z)$  ;

            retourner  $r$  ;

        sinon

$r.\text{sad} \leftarrow \text{ABR-Inserer}(r.\text{sad}, z)$  ;

            retourner  $r$  ;

    fin si

fin si

Fin

Complexité au pire :  $O(\text{hauteur de l'ABR})$

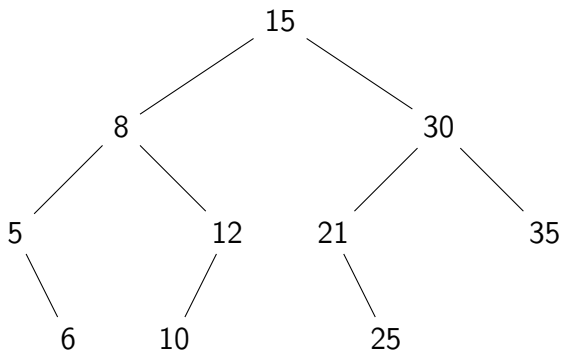
# Insertion à la racine

## Principe

- ▶ Le nœud à insérer devient la nouvelle racine.
- ▶ Il faut trouver les nœuds à mettre dans le sous-arbre droit et ceux à mettre dans le sous-arbre gauche
- ▶ On va faire cela par étape en destructurant l'arbre de départ.

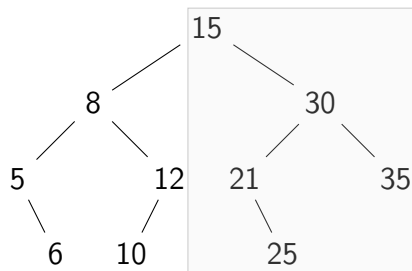
# Insertion à la racine

On va ajouter 11 à l'arbre suivant :

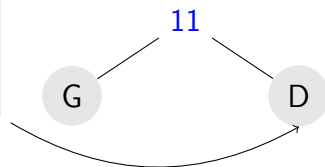


# Insertion à la racine

Arbre initial

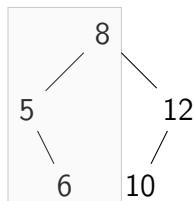


Arbre en construction

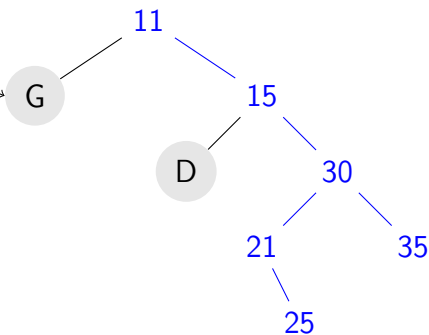


# Insertion à la racine

Arbre initial



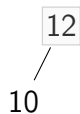
Arbre en construction



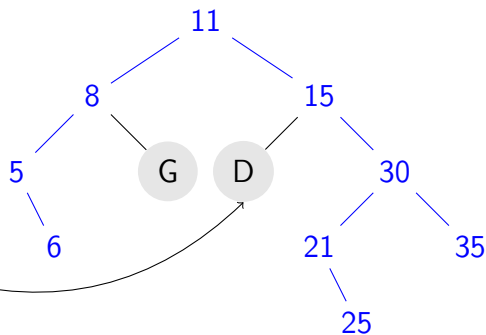


# Insertion à la racine

Arbre initial



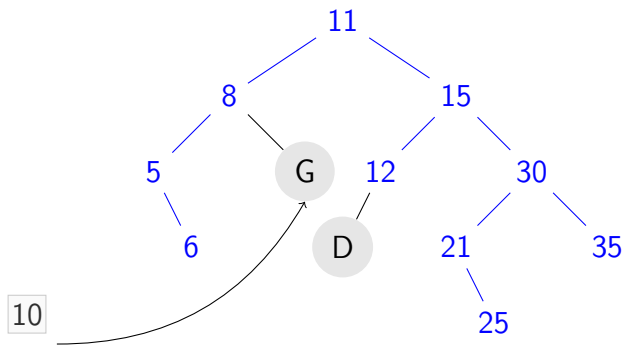
Arbre en construction



# Insertion à la racine

Arbre initial

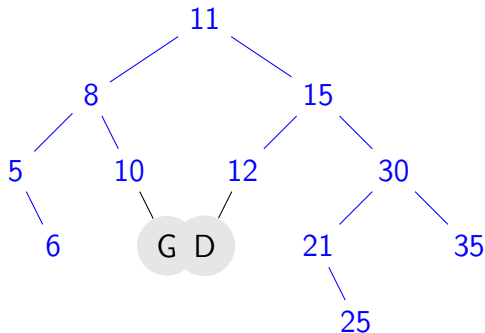
Arbre en construction



# Insertion à la racine

Arbre initial

Arbre en construction



Complexité au pire :  $O(\text{hauteur de l'ABR})$