UNIVERSITÉ PARIS 1
PANTHÉON SORBONNE

INF6
ALGORITHMIQUE AVANCÉE

Manuele Kirsch Pinheiro
Carine Souveyet

Contenu prévisionnel

- Piles et files
- Listes
- Récursivité
  - Récursivité dans le calcul
  - Récursivité structurelle
- Arbres binaires
  - Parcours en profondeur et en largeur
- Généralisation de la notion d'arbre
  - Insertion et suppression de nœuds
- **Arbre de recherche**
  - **Recherche & manipulation**
  - **Rééquilibrage**

UNIVERSITÉ PARIS 1
PANTHÉON SORBONNE

ARBRES DE RECHERCHE

UNIVERSITÉ PARIS 1 PANTHÉON SORBONNE

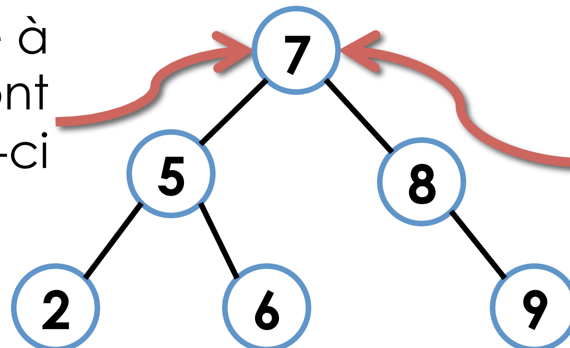
Arbres binaires de recherche (*Binary Search Trees*)

- Un arbre de recherche est un arbre binaire où les nœuds sont organisés suivant une **relation d'ordre**

  Soit a un arbre de recherche
  $\forall a \in$ ArbreRecherche,
  $\forall e \in$ SousArbreGauche(a) et $\forall e' \in$ SousArbreDroit(a)
  alors

  clé (e) ≤ clé (racine (a)) < clé (e')

Toutes les valeurs de clé à gauche de la racine sont inférieures à celle-ci
clé (e) ≤ clé (racine)

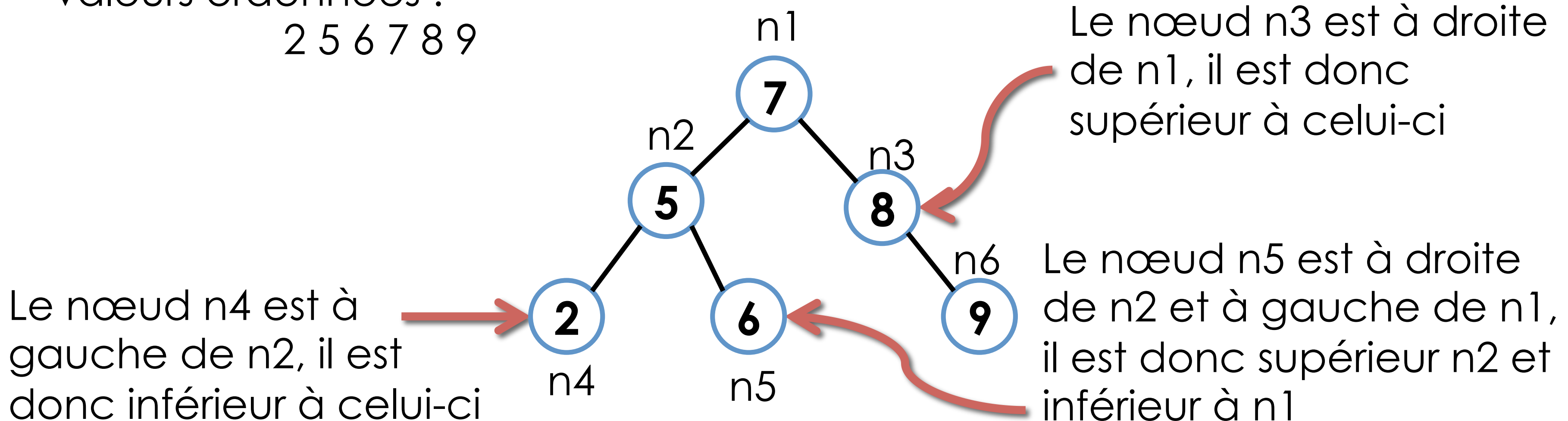


Toutes les valeurs de clé à droite de la racine sont supérieures à celle-ci
clé (e') > clé (racine)

UNIVERSITÉ PARIS 1 PANTHÉON SORBONNE

Définitions

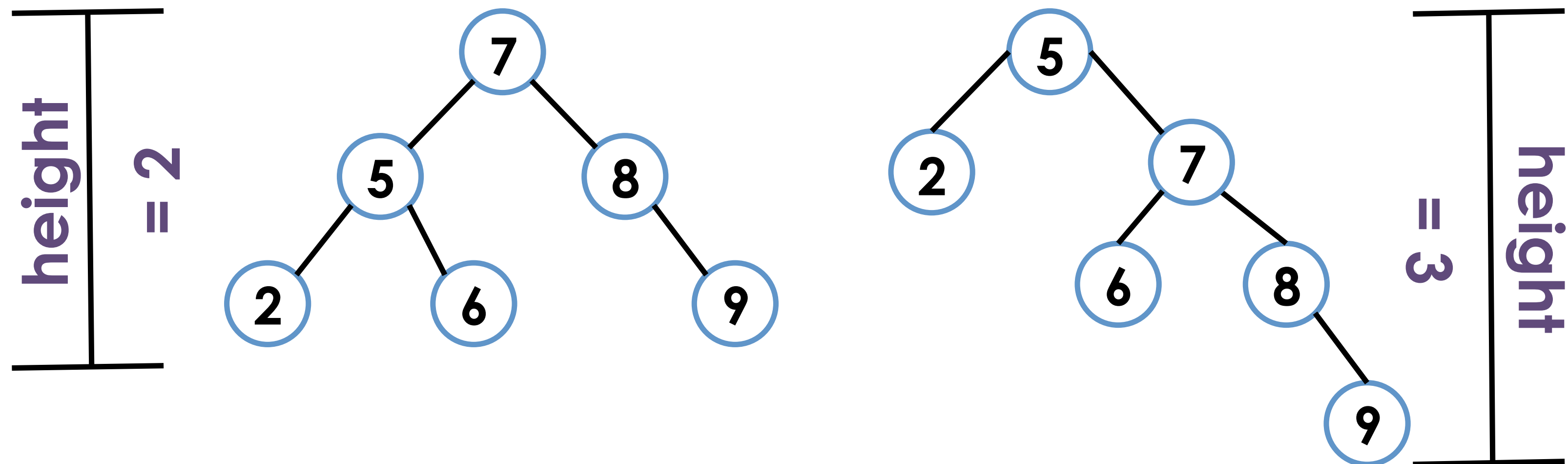
- Dans un arbre binaire de recherche :
  - **Toutes** les clés du sous arbre **gauche** sont **inférieurs ou égales** à la clé du nœud
  - **Toutes** les clés du sous arbre **droit** sont **strictement supérieurs** à la clé du nœud



UNIVERSITÉ PARIS 1 PANTHÉON SORBONNE

Définitions

- Un même ensemble de valeurs peut être représenté par différents arbres, aux profondeurs (*height*) distinctes



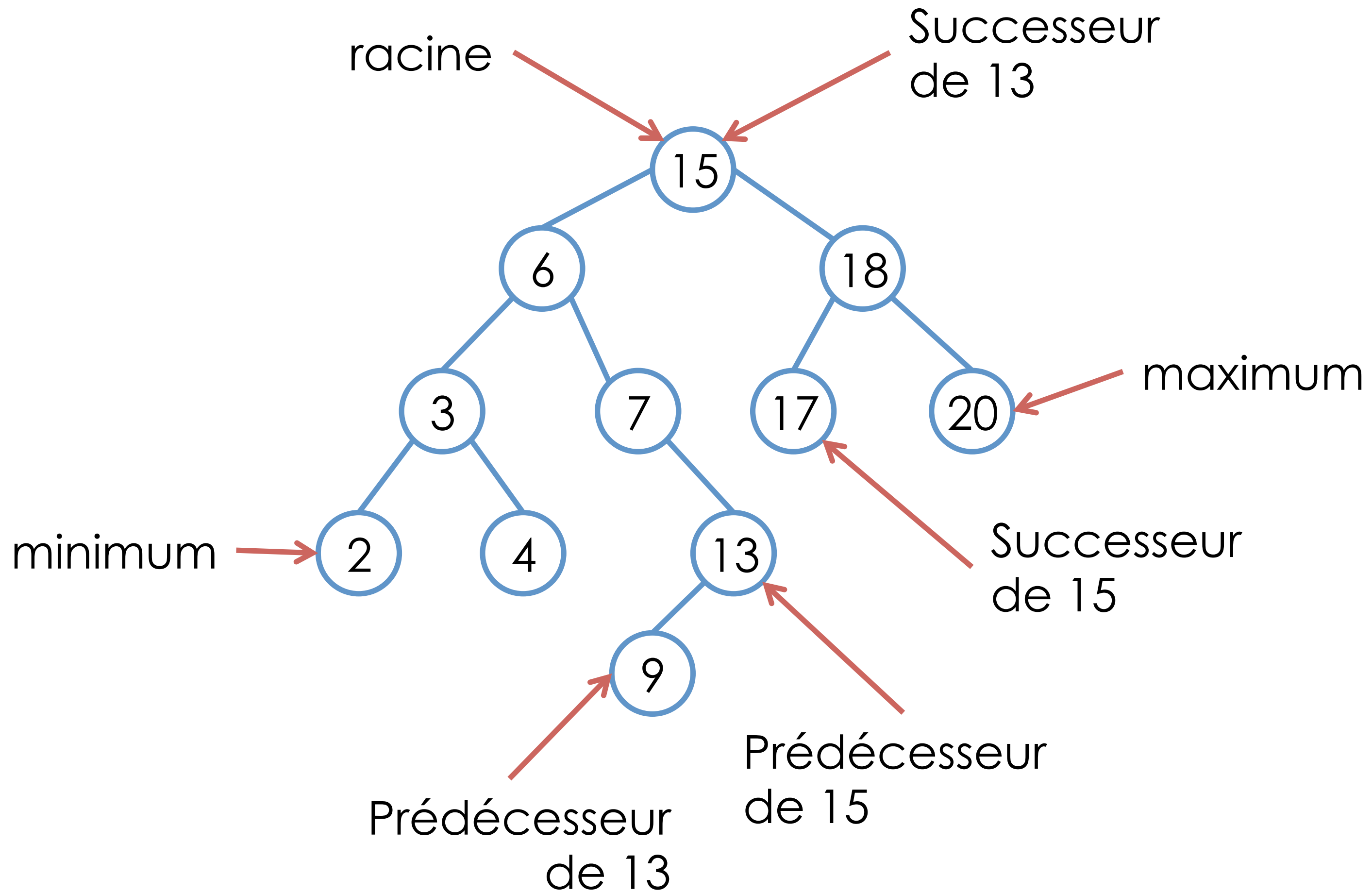
+ profondeur ⇒ **-** recherche efficace

Opérations

- Différentes opérations peuvent être réalisées dans un Arbre Binaire de Recherche (ABR)
- **Opérations de recherche**
  - ***Search*** → recherche une valeur dans l'arbre
  - ***Minimum*** → trouver la plus petite valeur contenue dans l'arbre
  - ***Maximum*** → trouver la valeur maximale contenue dans l'arbre
  - ***Successeur*** → trouver le prochain nœud (ou la prochaine valeur) d'un nœud / valeur
  - ***Prédécesseur*** → trouver le nœud (ou valeur) précédent un autre
- **Opérations de manipulation**
  - ***Insertion*** → insérer une nouvelle valeur dans l'arbre
  - ***Suppression*** → supprimer une valeur de l'arbre
  - ***Rééquilibrage*** → rééquilibre l'arbre de manière à contrôler sa profondeur

UNIVERSITÉ PARIS 1 PANTHÉON SORBONNE

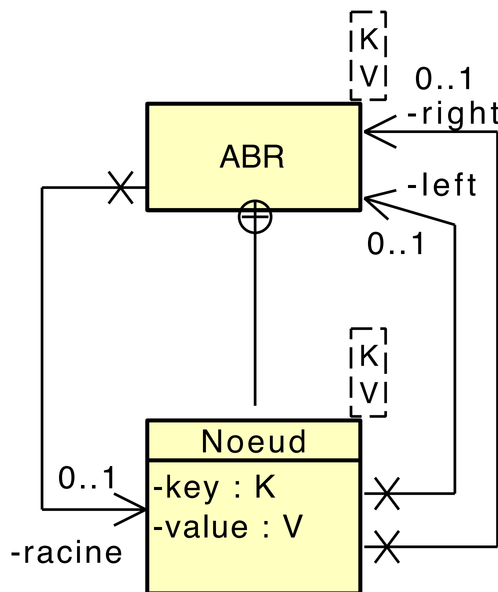
Opérations



UNIVERSITÉ PARIS 1 PANTHÉON SORBONNE

Implémentations

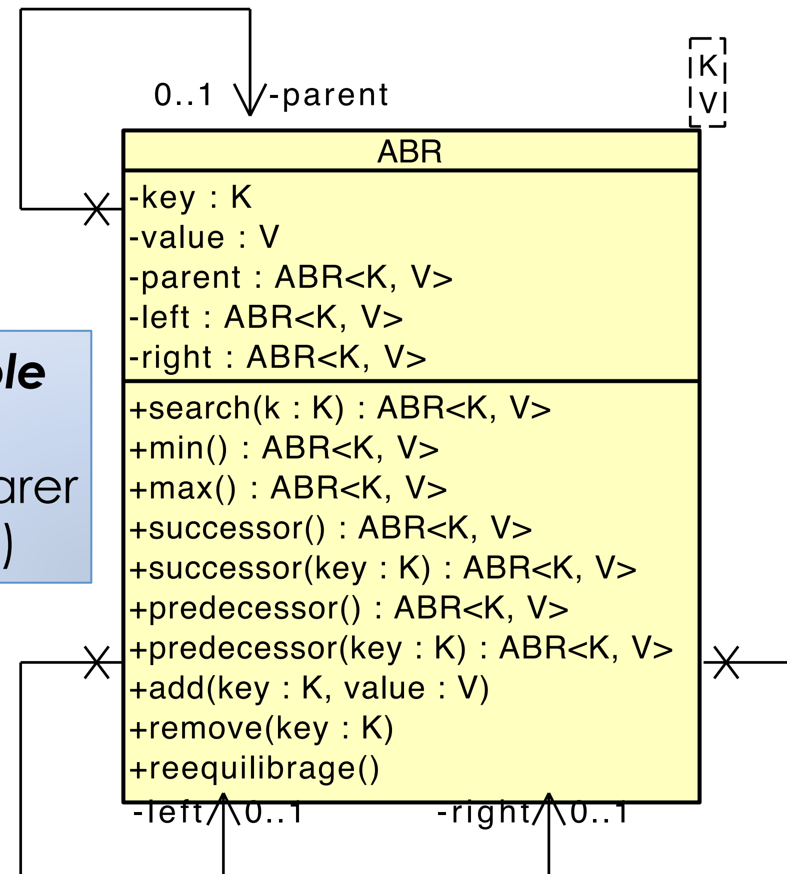
- Différentes implémentations sont possibles
  - avec ou sans la notion de **nœud**
  - avec ou sans le **parent**
  - ! **Attention** : *l'absence d'un lien vers le parent a un fort impact sur certaines opérations*



K extends Comparable

↓

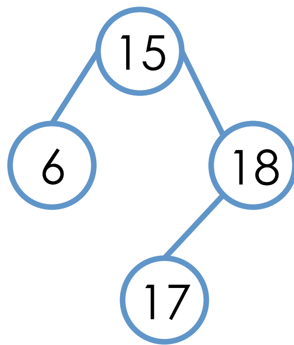
afin de pouvoir comparer les éléments (< ou >)

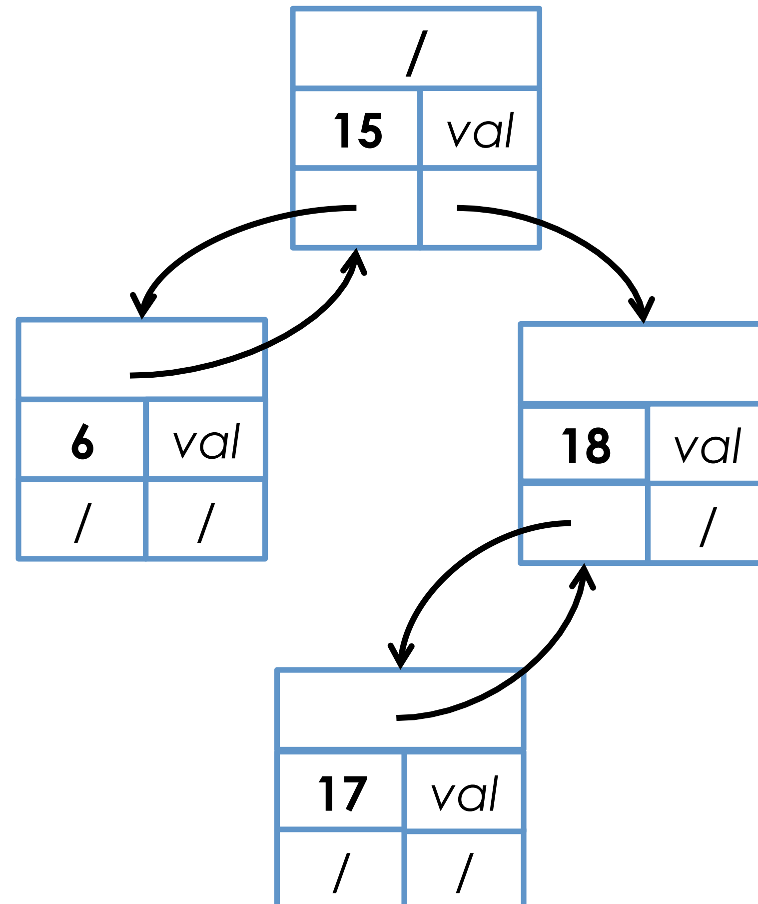


UNIVERSITÉ PARIS 1 PANTHÉON SORBONNE

Implémentations

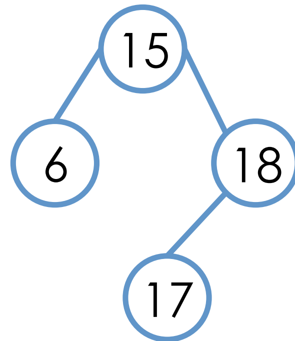
- Avec parent et sans la notion de nœud





Implémentations

- Sans parent et avec la notion de nœud

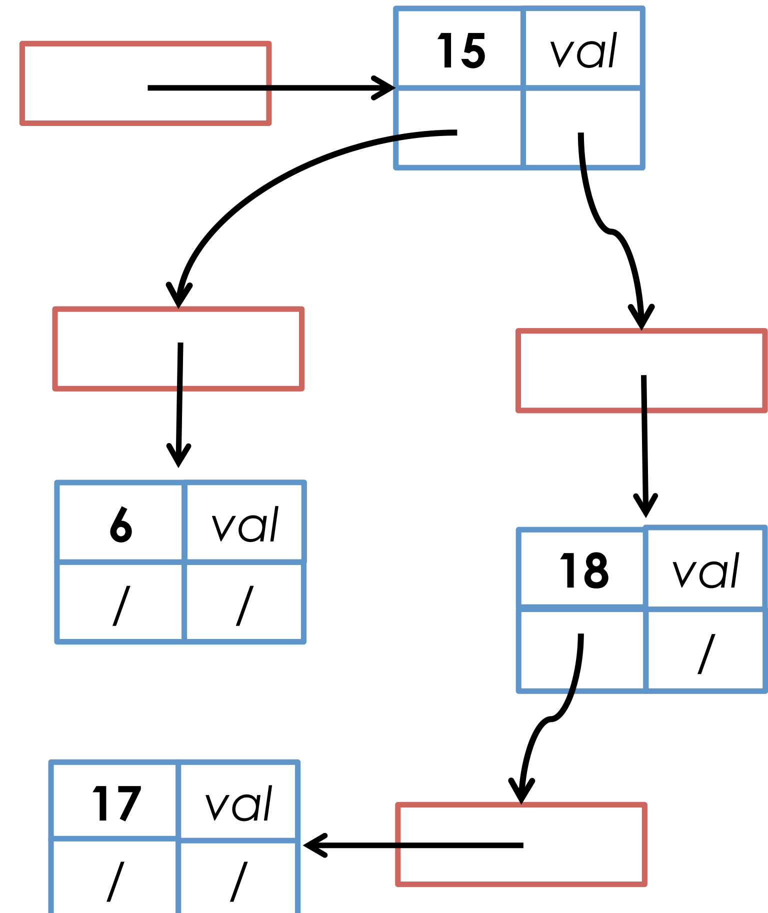


ABR

racine

Nœud

key	*val*
left	right



Recherche dans un ABR

- **Search**
  - **Objectif** : retrouver le **nœud** (sous-arbre) contenant une certaine valeur ***K***
    - On dit que **chaque nœud** contient une valeur de **clé** qui détermine **son ordre dans l'arbre**
    - On cherche alors le **nœud** contenant la **clé *K***

Search : Arbre x Key → Nœud

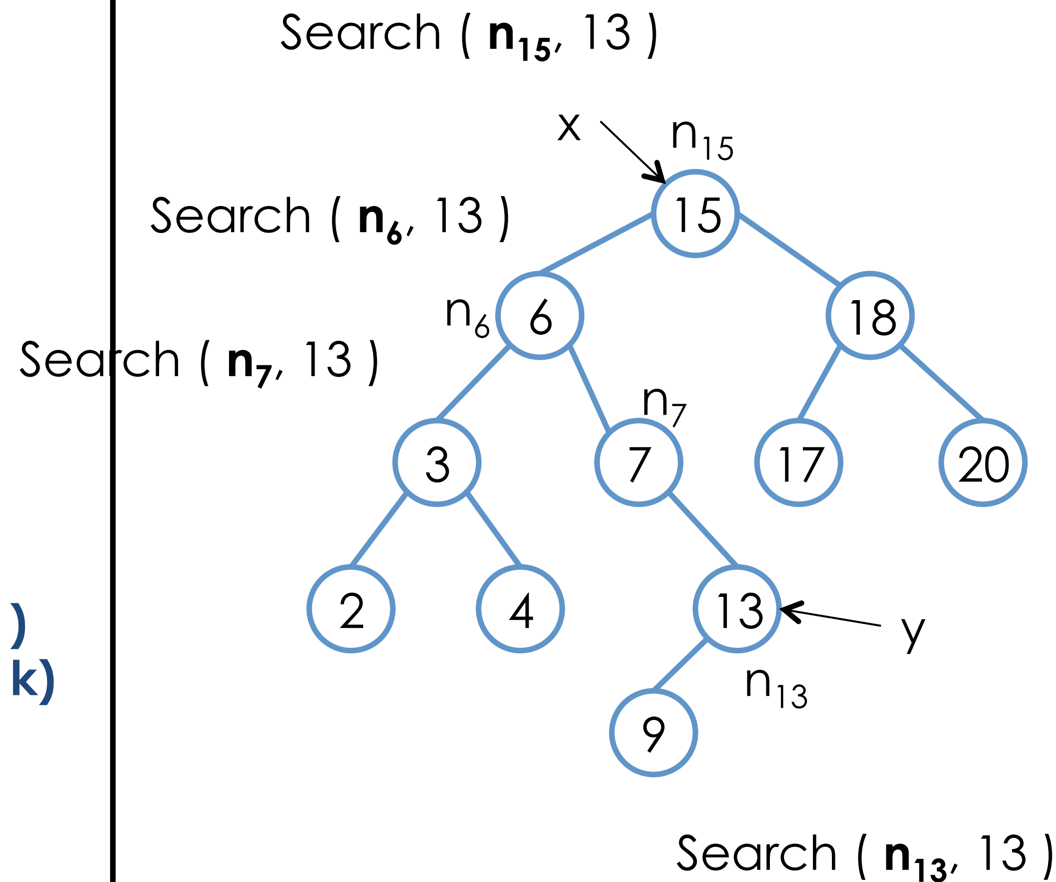
  - **Algorithme** :
    - On vérifie, pour chaque nœud **x**, s'il ne contient pas la valeur de **clé k** recherchée. Si ce n'est pas le cas, on étend la recherche aux sous arbres
      - Si **k < clé (x)** alors le nœud recherché est à la sous **arbre gauche**
      - Si **k > clé (x)** alors le nœud se trouve à la sous **arbre droite**

UNIVERSITÉ PARIS 1
PANTHÉON SORBONNE

Recherche

- On recherche à partir d'un nœud **x** (***this***)

```
Search
   Entrée :
       ABR  x
       Key k
 Sortie :
       ABR y

   Si x == null OU k == x.key
   alors
       y = x
   Sinon
       Si k < x.key
           alors  y = Search ( x.left, k )
           sinon y = Search ( x.right, k)
       fin si
   fin si
   retourner  y
fin
```

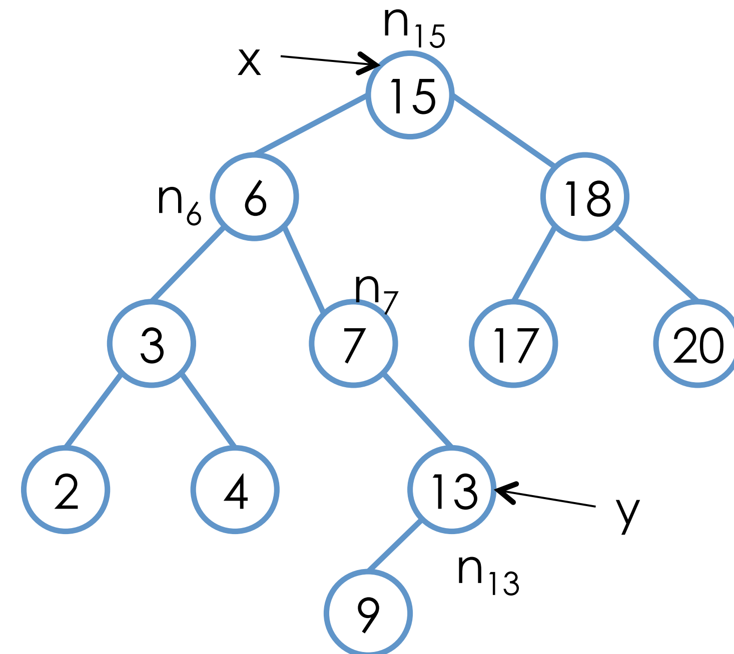


Recherche

- Ou de manière non récursive...

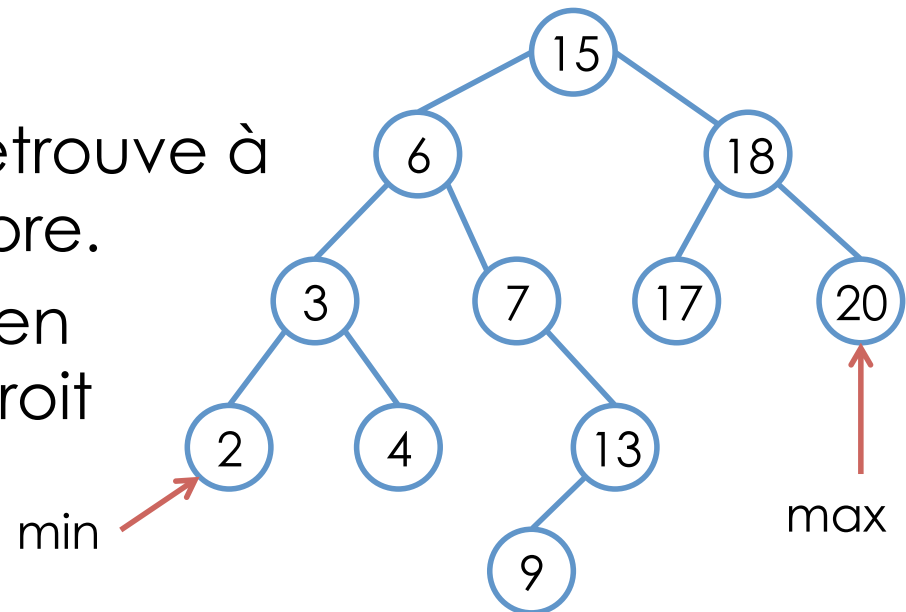
Search
Entrée :
ABR x
Key k
Sortie :
ABR y

Tant que x ≠ null ET k ≠ x.key
faire
Si k < x.key
alors **x = x.left**
sinon **x = x.right**
fin si
fin tant
y = x
retourner **y**
fin



Min / Max

- **Minimum**
  - La valeur **minimale** d'un ABR se trouve toujours à la **feuille à l'extrémité gauche** de l'arbre.
  - Pour la trouver, il suffit d'explorer le sous arbre gauche à partir de la racine
- **Maximum**
  - La valeur **maximale** se retrouve à **l'extrémité droite** de l'arbre.
  - Elle est donc accessible en explorant le sous arbre droit

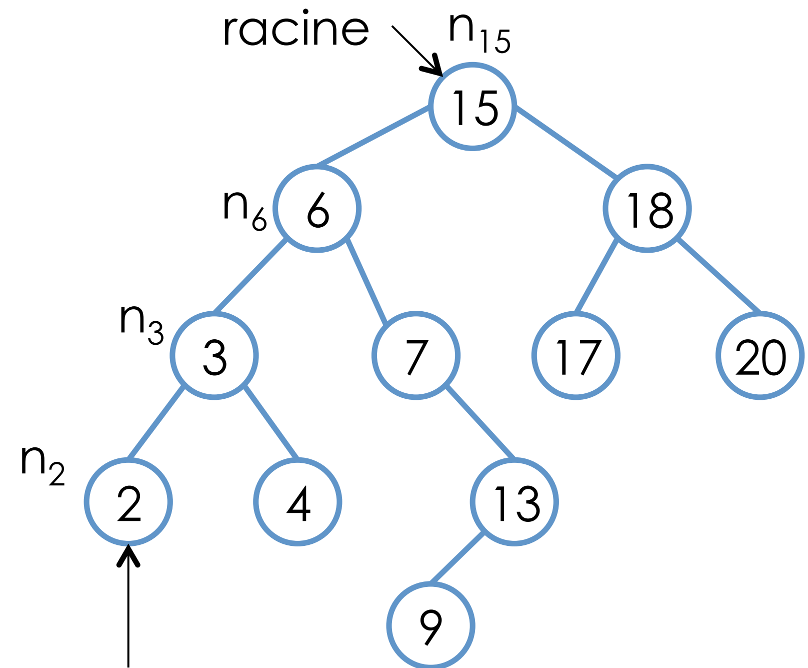


UNIVERSITÉ PARIS 1 PANTHÉON SORBONNE

Min / Max

- On recherche le min à partir de la **racine** (***this***)

```
Min
   Entrée :
       ABR racine
 Sortie :
       ABR x

   ABR x = racine
   Tant que x ≠ null ET x.left ≠ null
    faire
        x = x.left
    fin tant

    retourner x
fin
```



Exercice : on peut aussi le faire de manière récursive…

UNIVERSITÉ PARIS 1 PANTHÉON SORBONNE

Min / Max

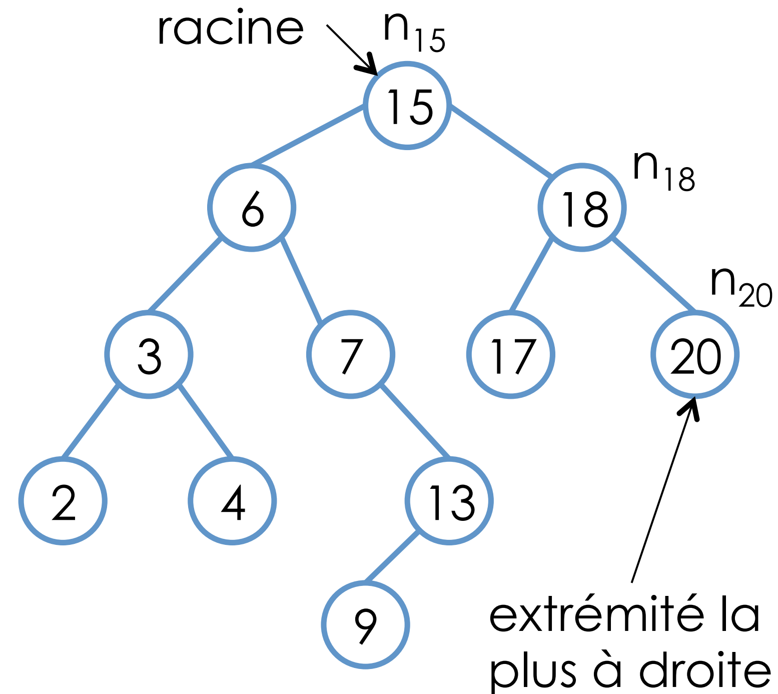
- On recherche le max à partir de la **racine** (***this***)

Max
Entrée :
ABR racine
Sortie :
ABR x

ABR x = racine
Tant que x ≠ null ET x.right ≠ null
faire
x = x.right
fin tant

retourner **x**
fin

$\mathbf{n_{15}}$.Max ()
≈
Max ($\mathbf{n_{15}}$)

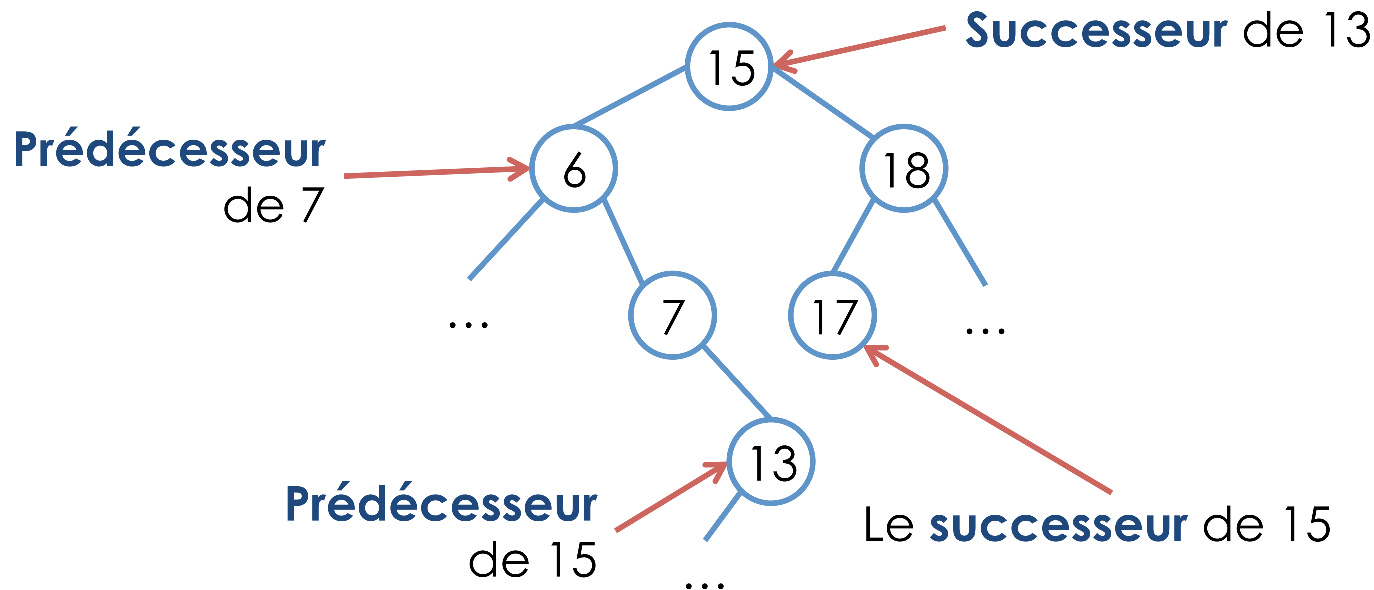


Exercice : on peut aussi le faire de manière récursive…

UNIVERSITÉ PARIS 1 PANTHÉON SORBONNE

Successeur / Prédécesseur

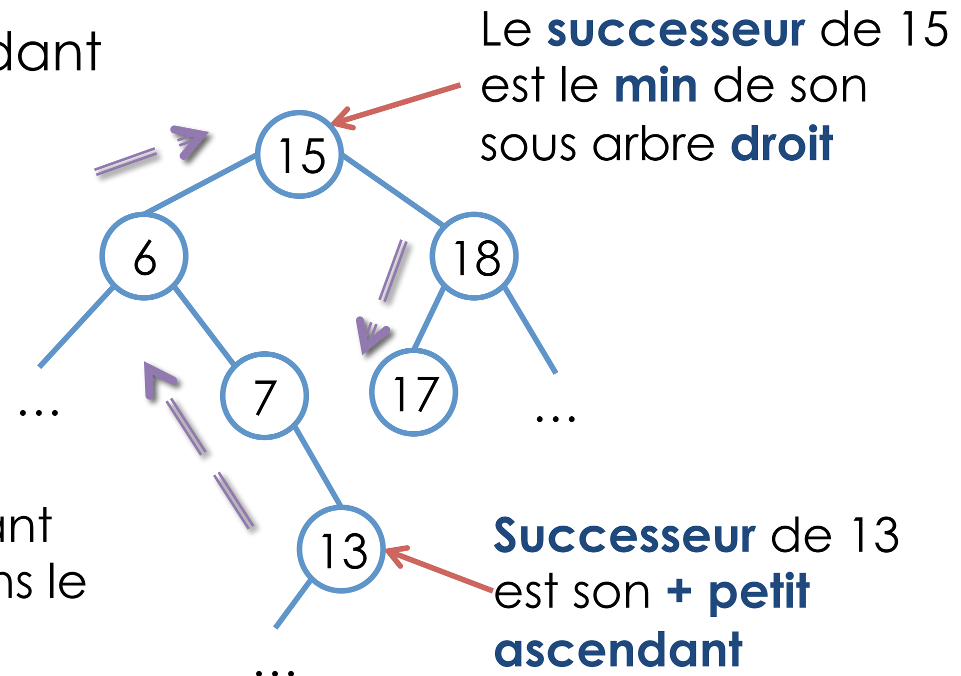
- Le **successeur** d'un **nœud x** est le **nœud y** contenant la **plus petite valeur supérieur à x**
- Le **prédécesseur** d'un **nœud x** est le **nœud y** contenant la **plus grand valeur inférieur à x**



UNIVERSITÉ PARIS 1 PANTHÉON SORBONNE

Successeur / Prédécesseur

- **Successeur :** deux cas possibles
  - x contient un sous arbre droit
    - son successeur est le **min** de son **sous arbre droit** (feuille à l'**extrémité gauche** de ce sous arbre)
  - x ne contient pas de sous arbre droit
    - le successeur est un ascendant
    - le + petit ancestral de x



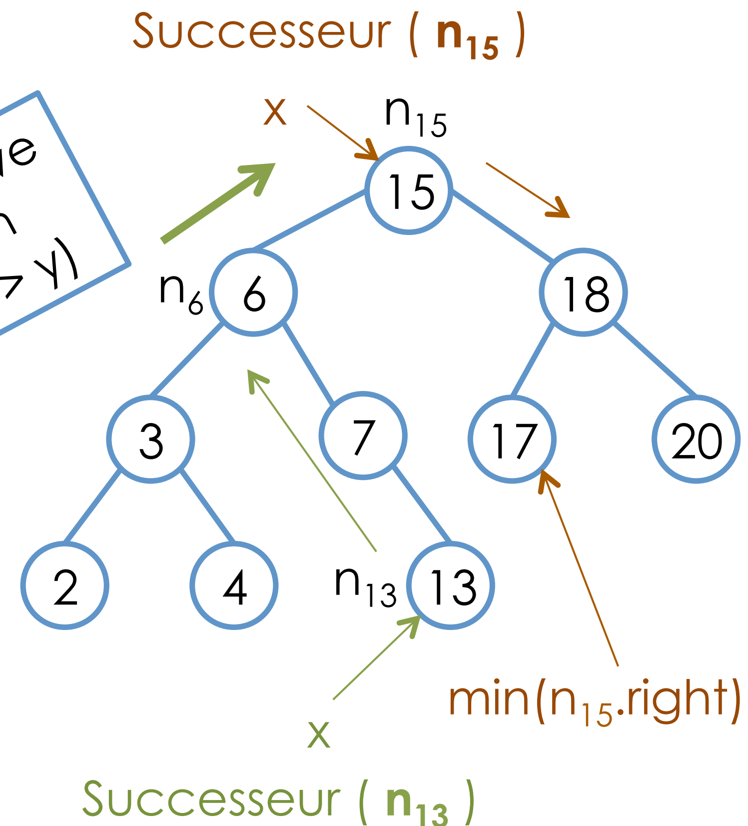
+ petit ascendant ➔ le 1er ascendant dont la branche est à **gauche** (dans le **sous arbre gauche de son parent**)

Successeur / Prédécesseur

- On recherche le successeur à partir d'un nœud **x** (***this***)

```
Successeur
   Entrée :
       ABR  x
   Sortie :
       ABR y

   Si x ≠ null ET x.right ≠ null
   alors
       y = Min (x.right)
   sinon
       y = x.parent
       Tant que y ≠ null ET x = y.right
       faire
           x = y
           y = y.parent
       fin tant
   retourner  y
fin
```

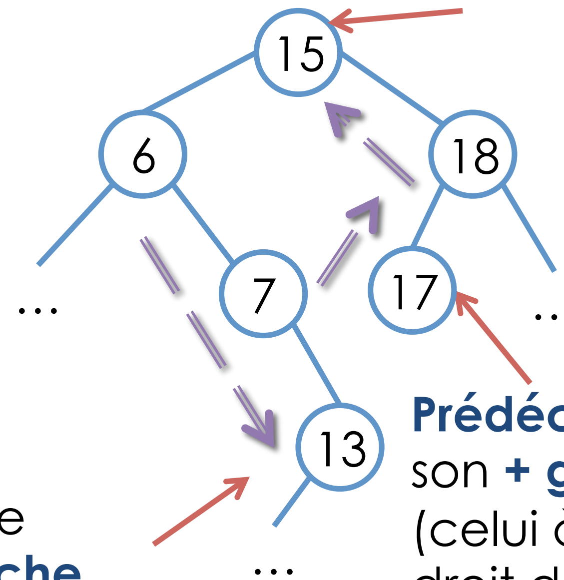
tant que x se trouve à la droite de son parent (donc x > y)



UNIVERSITÉ PARIS 1 PANTHÉON SORBONNE

Successeur / Prédécesseur

- **Prédécesseur :** idem successeur, en inversant droit et gauche
  - x contient un sous arbre **gauche**
    - son prédécesseur est le **max** de son **sous arbre gauche** (feuille à l'**extrémité droit** de ce sous arbre)
  - x ne contient pas de sous arbre **gauche**
    - le prédécesseur est un ascendant
    - le **1er ancestral** de x appartenant au **sous arbre droit** de son parent



Le **prédécesseur** de 15 est le **max** de son sous arbre **gauche**

Prédécesseur de 17 est son **+ grand ascendant** (celui à la sous arbre droit de son père)

UNIVERSITÉ PARIS 1 PANTHÉON SORBONNE

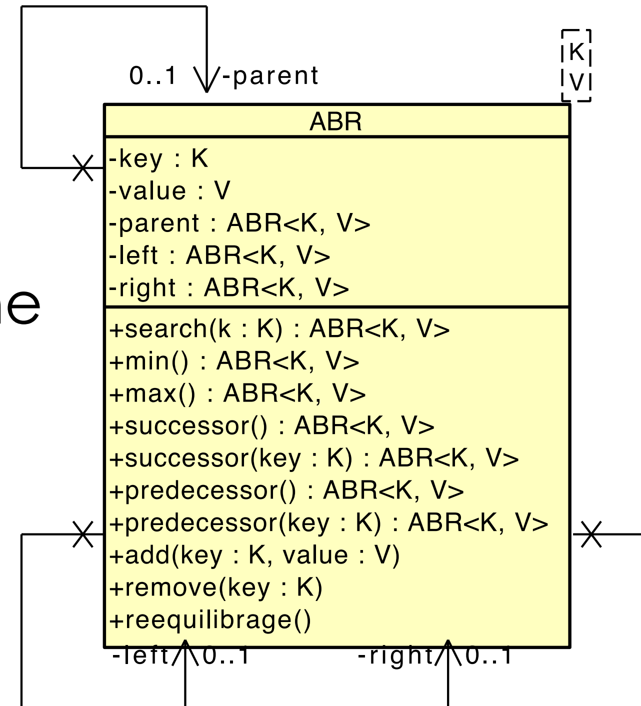
Successeur / Prédécesseur

- On recherche le successeur à partir d'un nœud **x** (***this***)

```
Prédécesseur
   Entrée :
       ABR  x
 Sortie :
       ABR y

   Si x ≠ null ET x.left ≠ null
   alors
       y = Max (x.left)
   sinon
      y = x.parent
      Tant que y ≠ null ET x = y.leftt
      faire
          x = y
          y = y.parent
      fin tant
    retourner  y
fin
```

Successeur / Prédécesseur

- On peut également demander le successeur ou prédécesseur d'une **valeur de clé** (et pas d'un nœud)
  - **successor()** → successeur d'un nœud (au vu de sa clé)
  - **successor(key: K)** → successeur d'une valeur de clé à partir de la racine



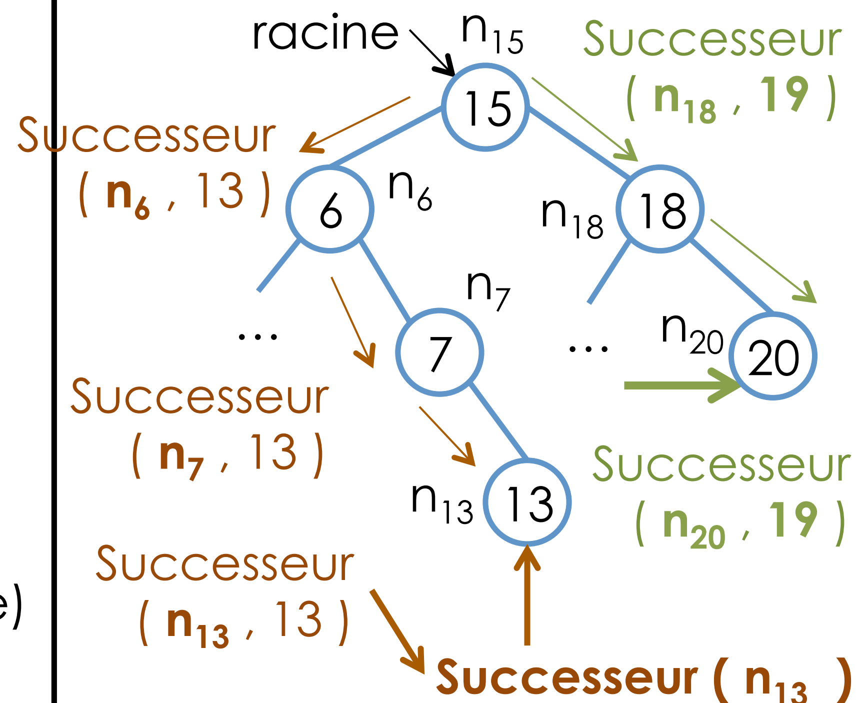
- Attention :
  - lors qu'on n'a pas le parent, successor () considère le nœud comme racine (impossible de remonter l'arbre au-delà de ce nœud)

Successeur

```
Successeur
  Entrée :
     Key k
     ABR racine
  Sortie :
     ABR a

  Si racine.key ≠ null Alors
     Si k < racine.key Alors
        Si racine.left ≠ null Alors
           a = successeur (k, racine.left)
        sinon  a = racine
        fin Si
     Sinon Si k > racile.key Alors
        Si racine.right ≠ null Alors
           a = successeur (k, racine.right)
        Sinon a = successeur (racine)
        Fin si
     Sinon  a = successeur (racine)
     fin Si
  Sinon a = null (ou exception car arbre vide)
   retourner  a
fin
```

il faut d'abord trouver le nœud contenant la valeur recherchée (ou la plus proche dans l'arbre)

- On recherche le successeur à partir d'un nœud **racine** (***this***)



UNIVERSITÉ PARIS 1
PANTHÉON SORBONNE

Successeur / Prédécesseur

- Si on simplifie la structure de données en **supprimant le pointeur vers le parent** :
  - On ne peut plus remonter l'arbre → appel à partir de la racine
  - On aura besoin de garder une « **trace** » des appels pour retrouver les parents
    - Structure auxiliaire : **une pile**

```
public ABR successeur  (K key )

   Si this.key == null  Alors
       retourner null (ou lancer exception)

   Pile p = nouvelle Pile
   retourner this.successeur ( key, p )

fin
```

Méthode public

créer la pile et lancer la méthode interne (**protected**)

Méthode protected

ABR successeur (K key , Pile p) parcours l'arbre de manière récursive à la recherche du successeur

PANTHÉON SORBONNE UNIVERSITÉ PARIS 1

Successeur / Prédécesseur

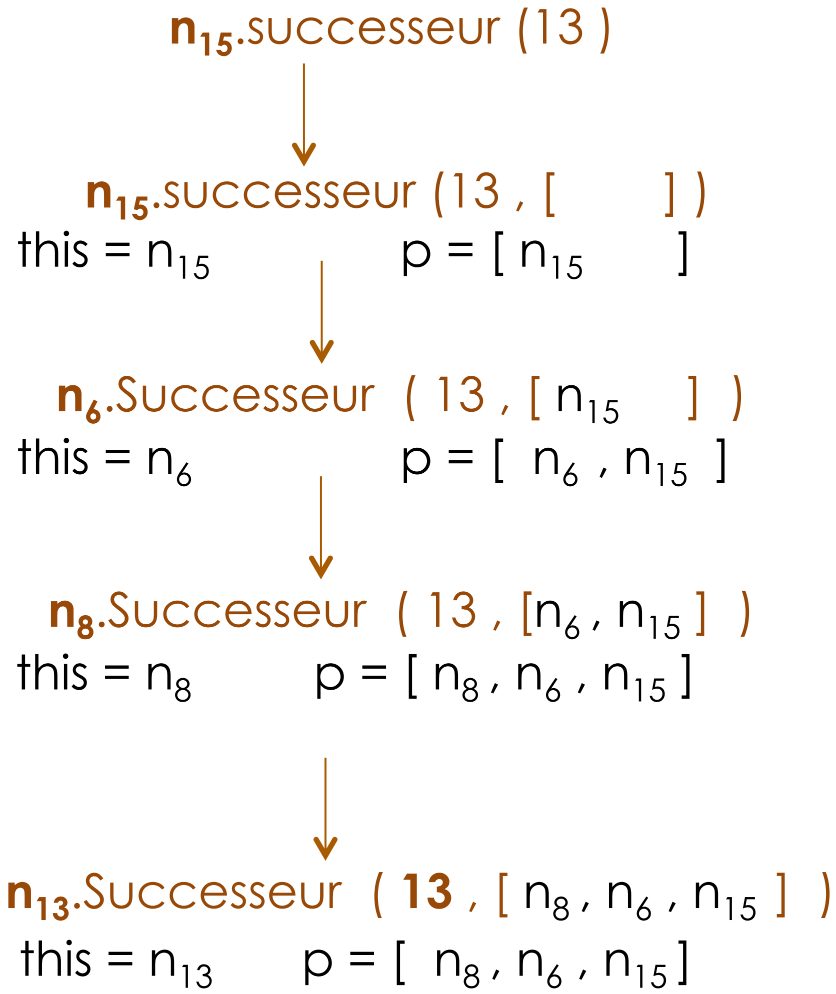
- Successeur (clé) sans parent

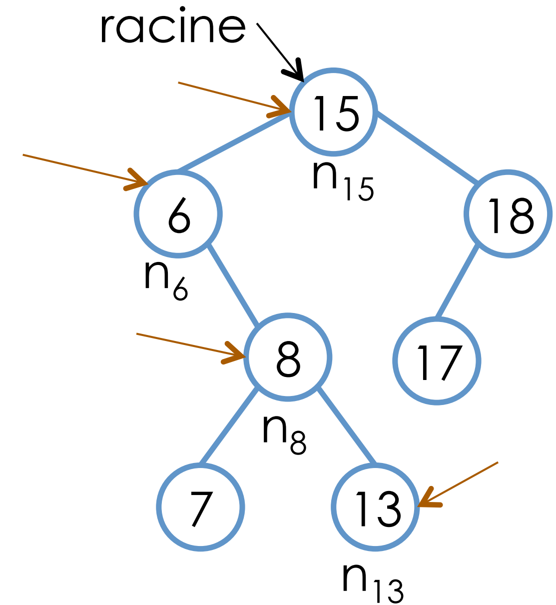
```
protected ABR successeur ( K key , Pile p )

  Si key < this.key Alors
     Si this.left ≠ null Alors
        p.empile ( this )
        retourner left.successeur ( key, p )
     sinon
        retourner this
     fin Si
  Sinon Si key > this.key Alors
     Si this.right ≠ null Alors
        p.empile ( this )
        retourner right.successeur ( k, p )
     Sinon
        retourner successeur ( this.key , p )
     Fin si
…
  Sinon
     Si this.right ≠ null Alors
        retourner this.right.min ()
     Sinon
        ABR x = this
        Tant que ! p.estVide() Faire
           ABR pere = p.depiler ()
           Si x ≠ pere.right Alors
              retourner pere
           Sinon
              x = pere
        fin Tant
        retourner null
     fin Si
  fin Si
fin
```

on ne l'a pas trouvé dans l'arbre…

Successeur / Prédécesseur

- Successeur (clé) **sans parent**





x = n_{13} p = [~~n_8 ,~~ n_6 , n_{15}]
pere = n_8
x = n_8 p = [~~n_8 , n_6 ,~~ n_{15}]
pere = n_6
x = n_6 p = [~~n_8 , n_6 , n_{15}~~]
pere = $\mathbf{n_{15}}$