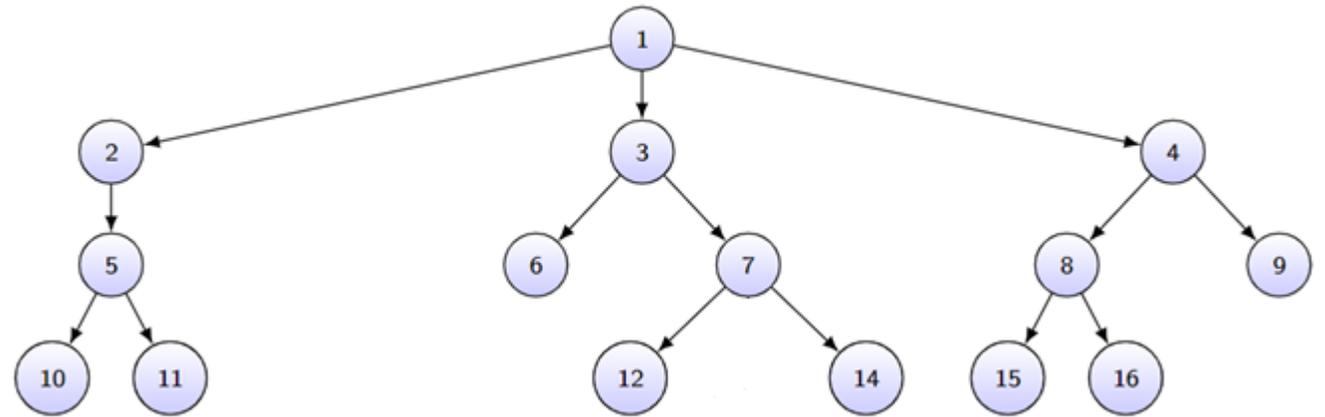


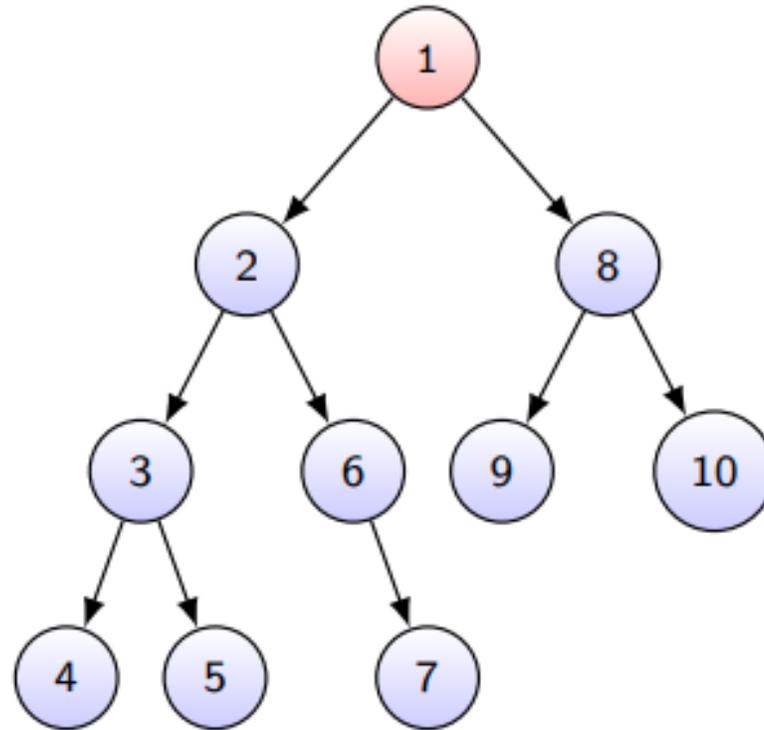
INFORMATIQUE 3

IV. ARBRES BINAIRES DE RECHERCHE



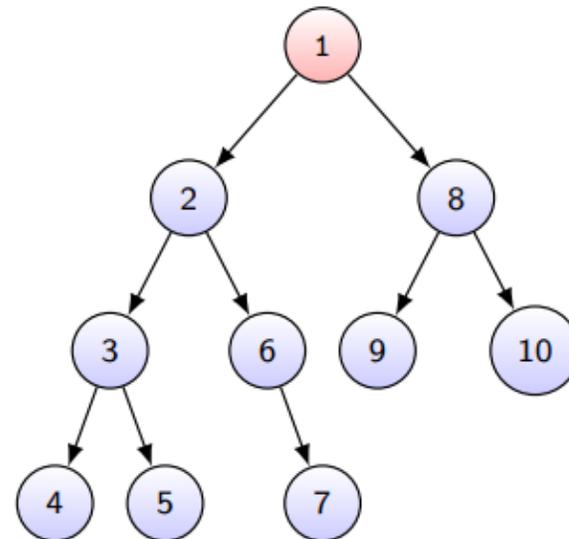
Rappel : arbre binaire

- Une arbre binaire de recherche est un arbre dont les nœuds ont au maximum 2 fils.



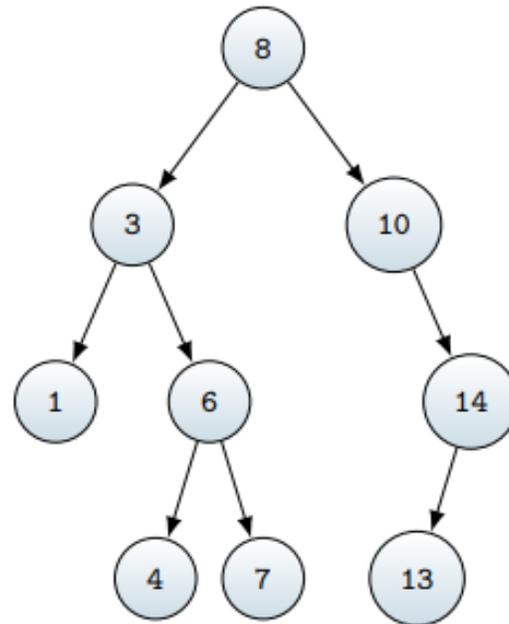
Principe de l'ABR

- Une arbre binaire est un arbre dont les nœuds ont au maximum 2 fils.
- Un arbre peut permettre de stocker des données. Si ces données ne sont pas organisées, la recherche d'élément n'est pas plus rapide que pour une liste.
- Dans un arbre quelconque les opérations de recherches sont en $O(n)$: tous les éléments doivent être parcourus.
- Dans un tableau, on peut trier les données pour faciliter la recherche. Comment faire pour un arbre ?



ABR : définition

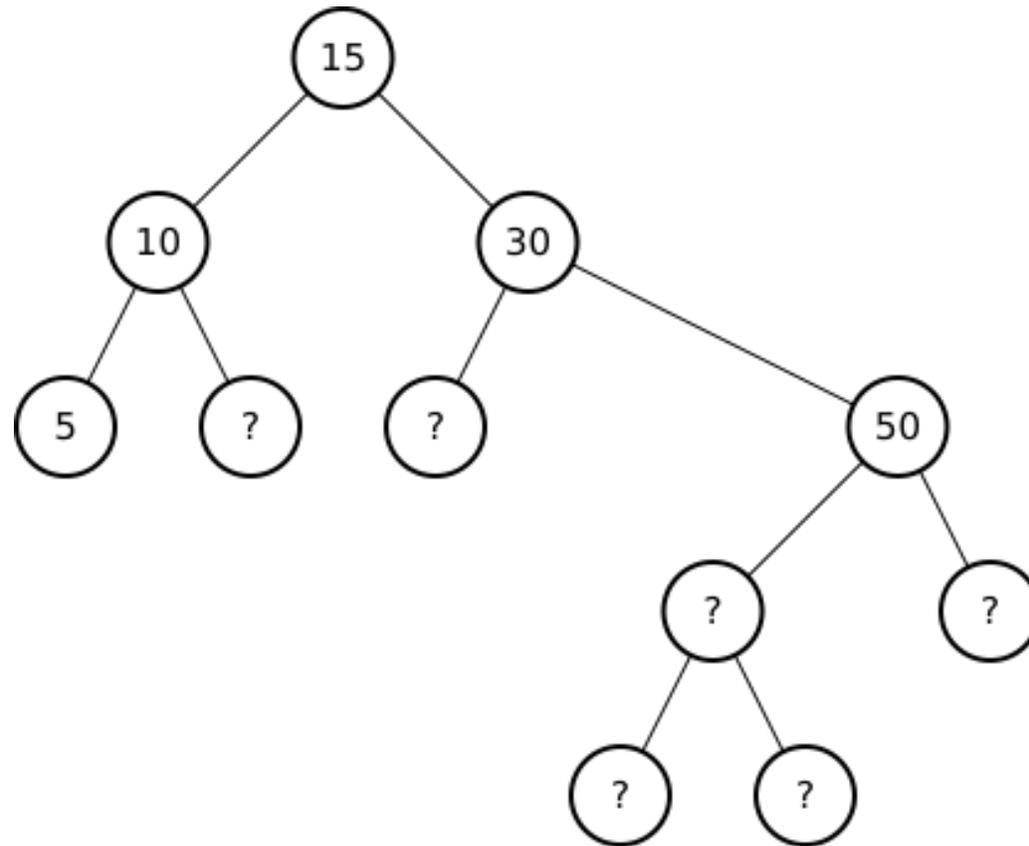
- Un arbre binaire a est appelé **Arbre Binaire de Recherche** (ABR) si et seulement si :
 - les valeurs du sous-arbre de gauche sont $<$ racine(a)
 - les valeurs du sous-arbre de droite sont $>$ racine(a)
 - tout sous-arbre de a est un ABR



- Remarque : selon la mise en œuvre de l'ABR, on interdit ou non des nœuds contenant des éléments de valeur égale.
On travaillera dans notre cours avec des arbres à **éléments uniques**.

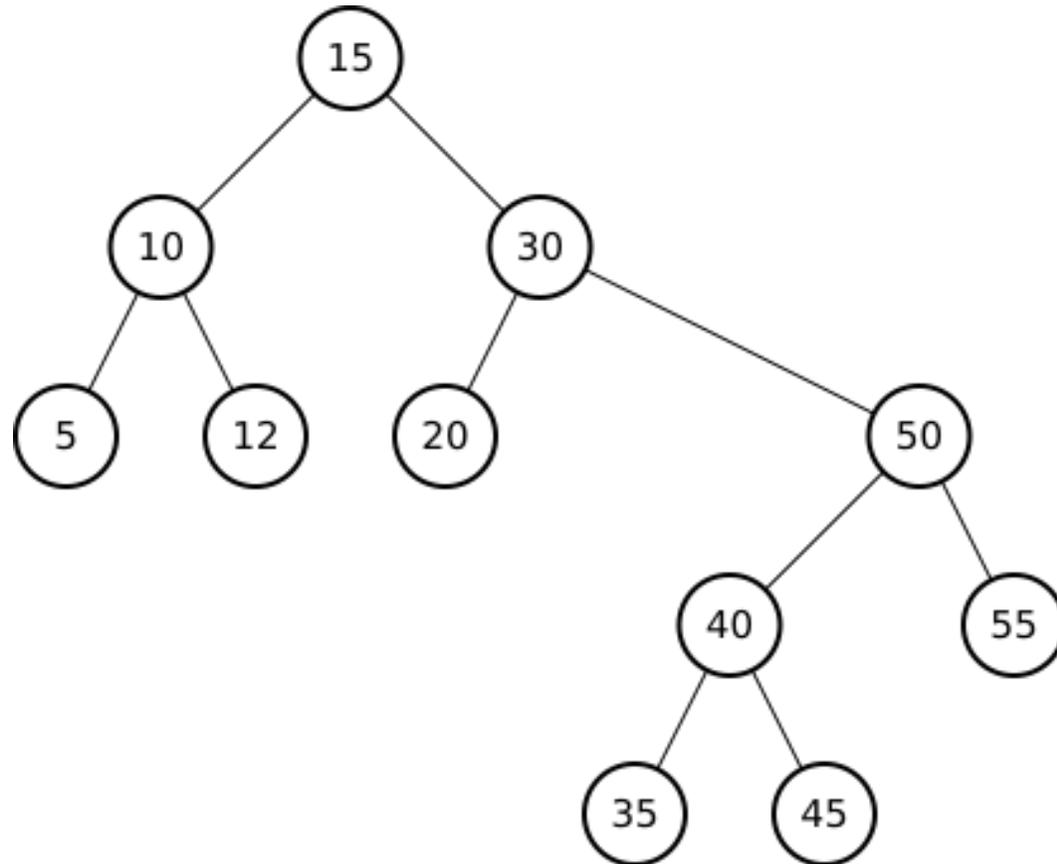
ABR : définition

- Compléter cet arbre :



ABR : définition

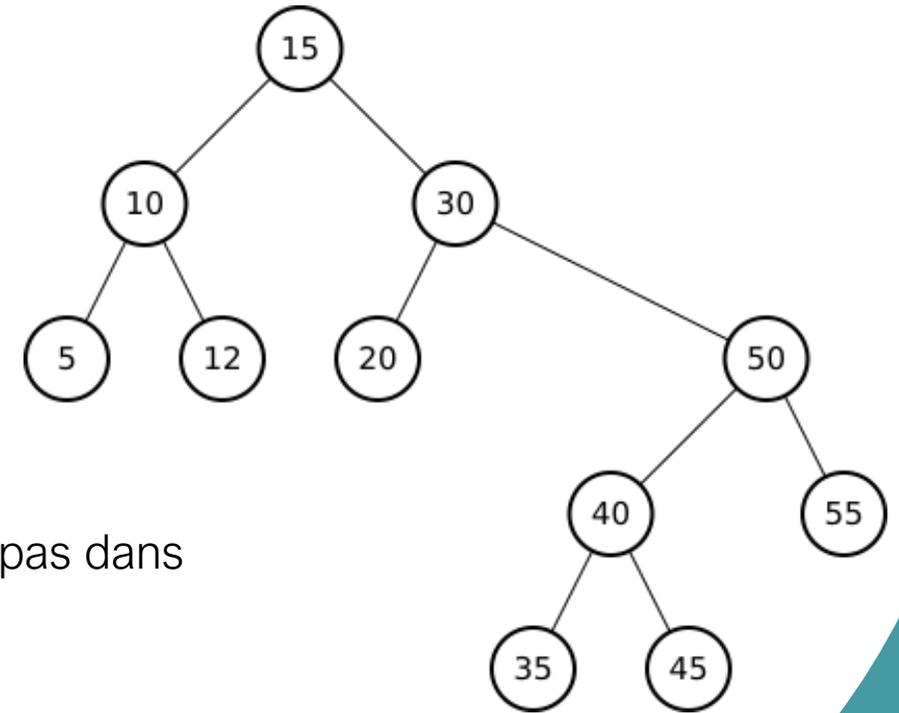
- Compléter cet arbre : une possibilité



ABR : opération de recherche

- Les ABR facilitent grandement les opérations de recherche : on sait quel sous-arbre explorer en fonction de la valeur du nœud sur lequel on se trouve.

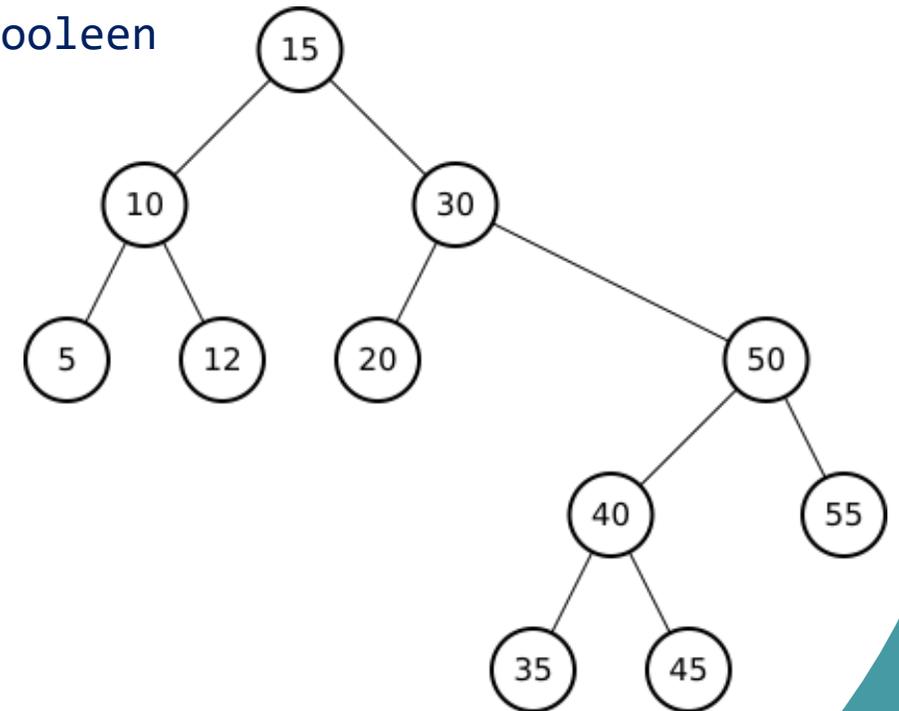
- Principe:
 - Comparaison avec la racine du nœud :
 - s'il est inférieur, recherche dans le sous-arbre gauche
 - s'il est supérieur, recherche dans le sous-arbre droit
 - Fin lorsque :
 - racine égale à la valeur cherchée
 - le sous-arbre gauche (ou droit) n'existe pas : la valeur n'est pas dans l'arbre



ABR : opération de recherche

- Algorithme

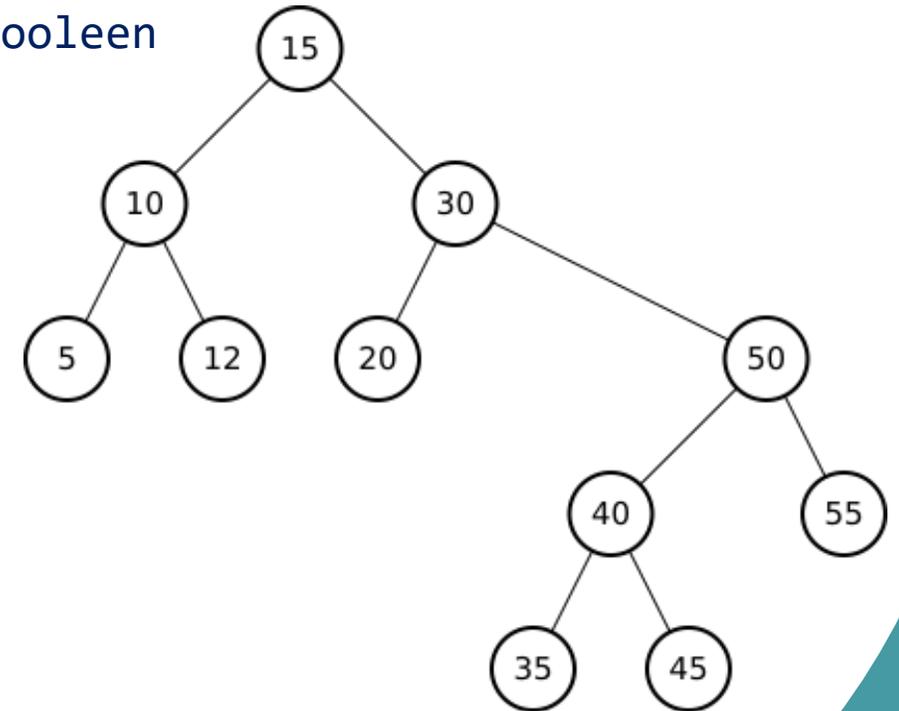
```
FONCTION recherche(a: pointeur sur Arbre, e: Element) : boolean
DEBUT
  SI (a EST EGAL A NULL) Alors
    ???
  SINON SI (element(a) EST EGAL A e) Alors
    ???
  SINON SI (e EST INF. STRICT. A element(a)) ALORS
    ???
  SINON
    ???
  FIN SI
Fin
```



ABR : opération de recherche

- Algorithme

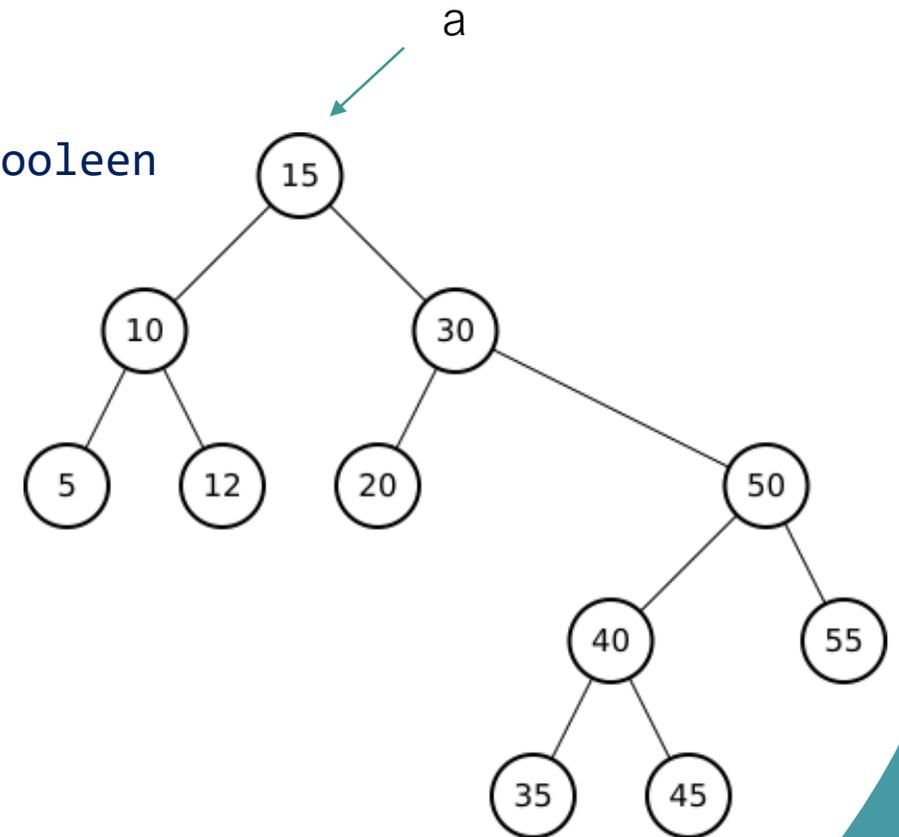
```
FONCTION recherche(a: pointeur sur Arbre, e: Element) : boolean
DEBUT
  SI (a EST EGAL A NULL) Alors
    RETOURNER FAUX
  SINON SI (element(a) EST EGAL A e) Alors
    RETOURNER VRAI
  SINON SI (e EST INF. STRICT. A element(a)) ALORS
    RETOURNER recherche(fg(a),e)
  SINON
    RETOURNER recherche(fd(a),e)
FIN SI
Fin
```



ABR : opération de recherche

- Algorithme : exemple : recherche (a,40)

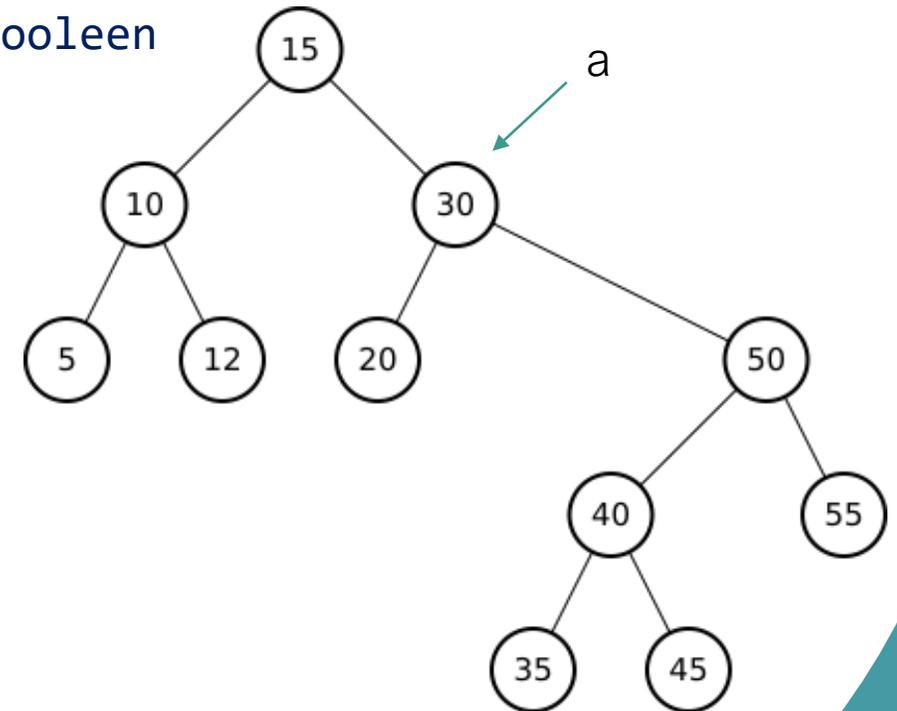
```
FONCTION recherche(a: pointeur sur Arbre, e: Element) : boolean
DEBUT
  SI (a EST EGAL A NULL) Alors
    RETOURNER FAUX
  SINON SI (element(a) EST EGAL A e) Alors
    RETOURNER VRAI
  SINON SI (e EST INF. STRICT. A element(a)) ALORS
    RETOURNER recherche(fg(a),e)
  SINON
    RETOURNER recherche(fd(a),e)
FIN SI
Fin
```



ABR : opération de recherche

- Algorithme : exemple : recherche (a,40)

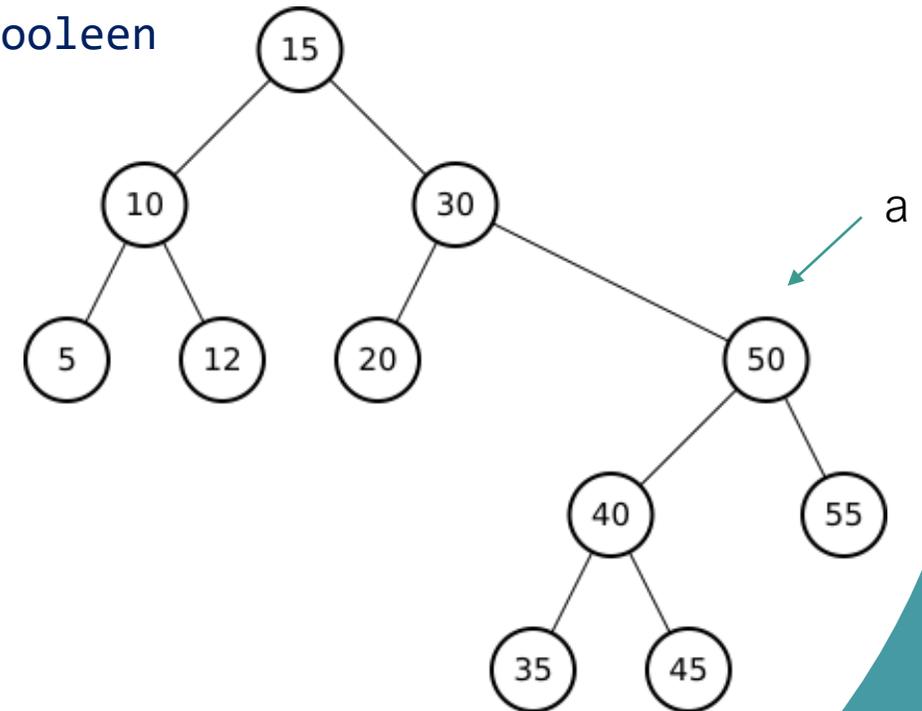
```
FONCTION recherche(a: pointeur sur Arbre, e: Element) : boolean
DEBUT
  SI (a EST EGAL A NULL) Alors
    RETOURNER FAUX
  SINON SI (element(a) EST EGAL A e) Alors
    RETOURNER VRAI
  SINON SI (e EST INF. STRICT. A element(a)) ALORS
    RETOURNER recherche(fg(a),e)
  SINON
    RETOURNER recherche(fd(a),e)
FIN SI
Fin
```



ABR : opération de recherche

- Algorithme : exemple : recherche (a,40)

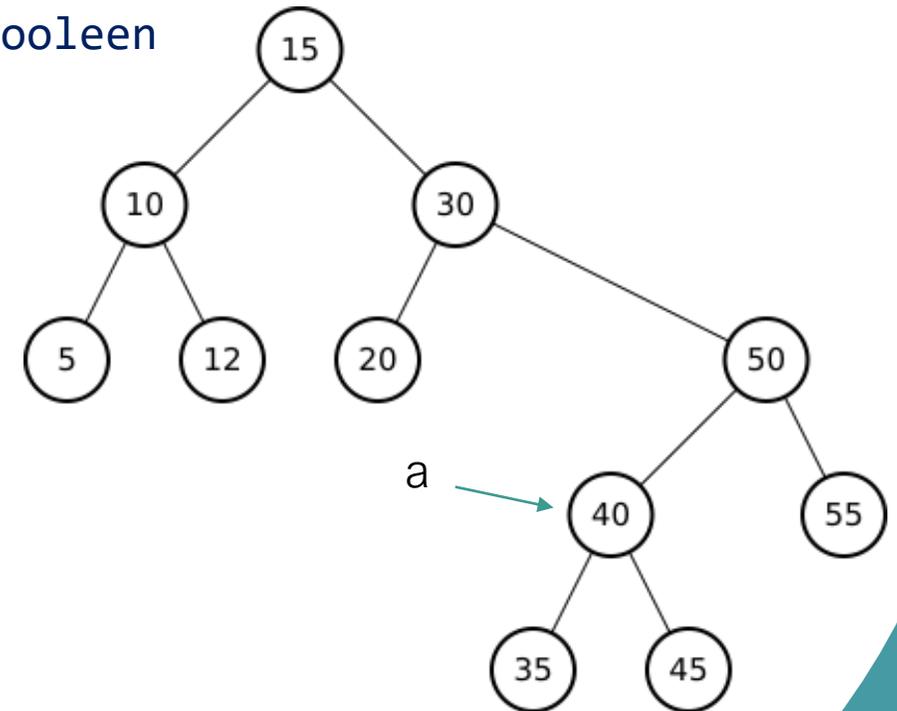
```
FONCTION recherche(a: pointeur sur Arbre, e: Element) : boolean
DEBUT
  SI (a EST EGAL A NULL) Alors
    RETOURNER FAUX
  SINON SI (element(a) EST EGAL A e) Alors
    RETOURNER VRAI
  SINON SI (e EST INF. STRICT. A element(a)) ALORS
    RETOURNER recherche(fg(a),e)
  SINON
    RETOURNER recherche(fd(a),e)
FIN SI
Fin
```



ABR : opération de recherche

- Algorithme : exemple : recherche (a,40)

```
FONCTION recherche(a: pointeur sur Arbre, e: Element) : boolean
DEBUT
  SI (a EST EGAL A NULL) Alors
    RETOURNER FAUX
  SINON SI (element(a) EST EGAL A e) Alors
    RETOURNER VRAI
  SINON SI (e EST INF. STRICT. A element(a)) ALORS
    RETOURNER recherche(fg(a),e)
  SINON
    RETOURNER recherche(fd(a),e)
FIN SI
Fin
```

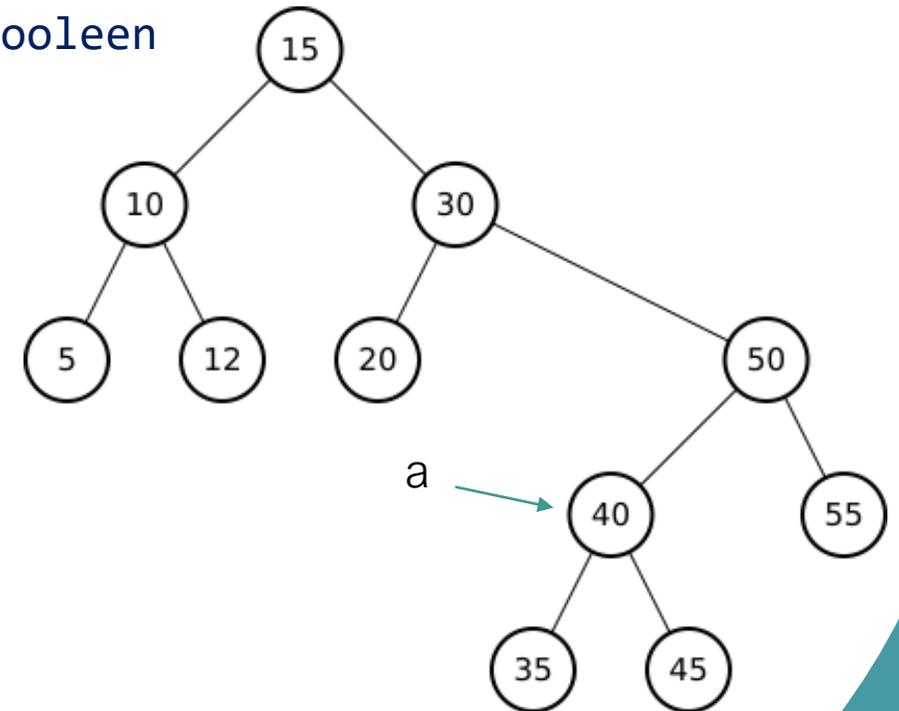


ABR : opération de recherche

- Algorithme : exemple : recherche (a,40)

```
FONCTION recherche(a: pointeur sur Arbre, e: Element) : boolean
DEBUT
  SI (a EST EGAL A NULL) Alors
    RETOURNER FAUX
  SINON SI (element(a) EST EGAL A e) Alors
    RETOURNER VRAI
  SINON SI (e EST INF. STRICT. A element(a)) ALORS
    RETOURNER recherche(fg(a),e)
  SINON
    RETOURNER recherche(fd(a),e)
FIN SI
Fin
```

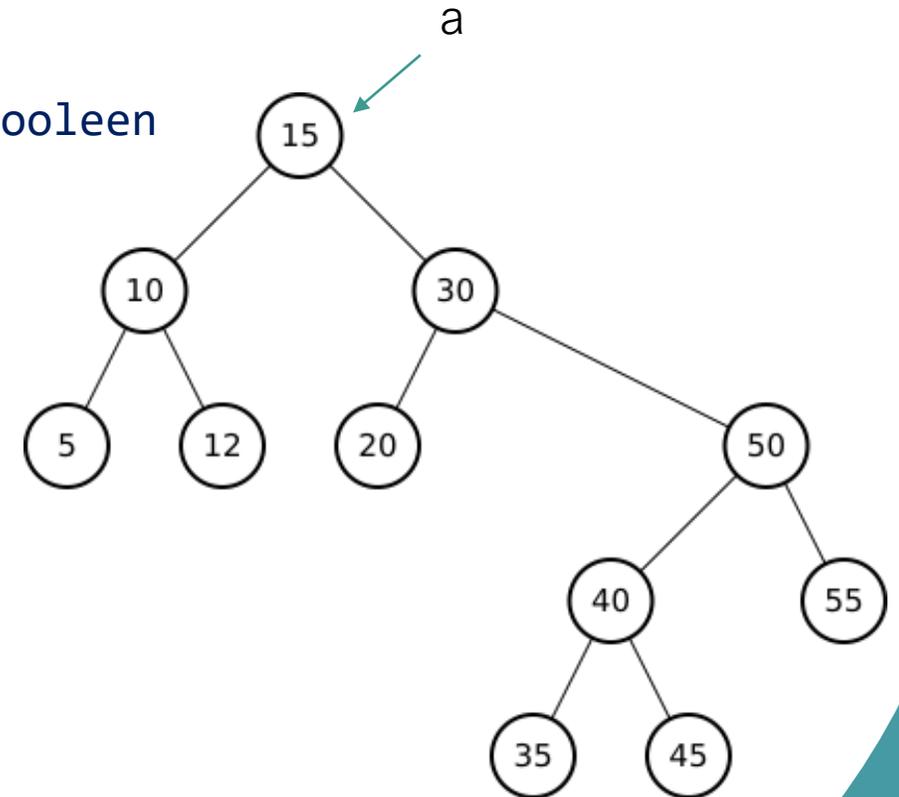
=> La fonction va retourner VRAI.



ABR : opération de recherche

- Algorithme : exemple : recherche (a,14)

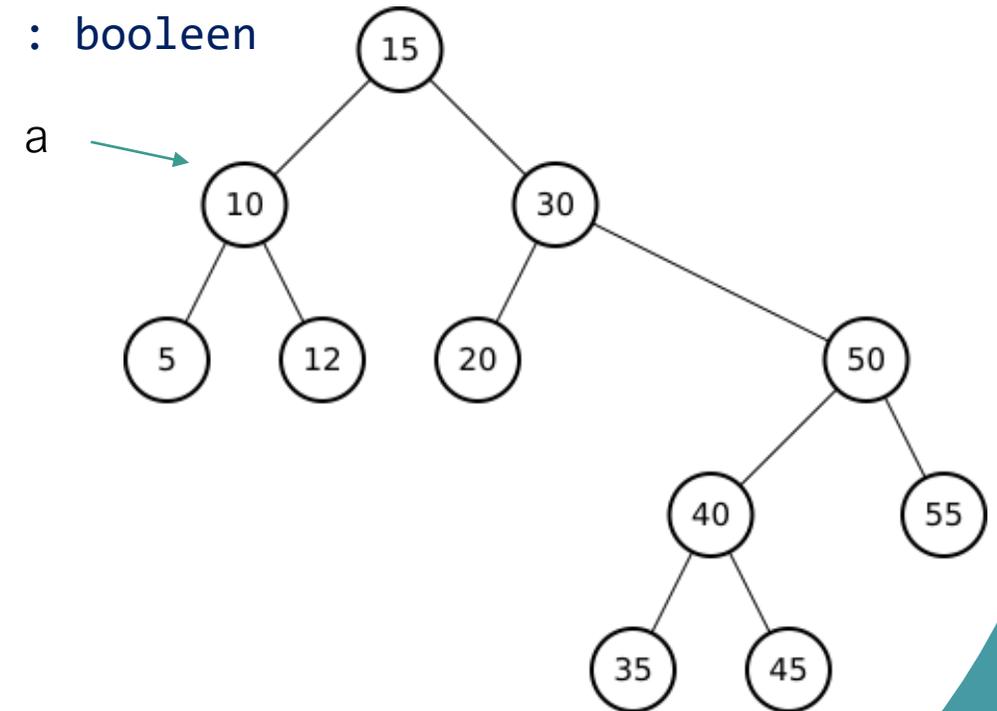
```
FONCTION recherche(a: pointeur sur Arbre, e: Element) : boolean
DEBUT
  SI (a EST EGAL A NULL) Alors
    RETOURNER FAUX
  SINON SI (element(a) EST EGAL A e) Alors
    RETOURNER VRAI
  SINON SI (e EST INF. STRICT. A element(a)) ALORS
    RETOURNER recherche(fg(a),e)
  SINON
    RETOURNER recherche(fd(a),e)
FIN SI
Fin
```



ABR : opération de recherche

- Algorithme : exemple : recherche (a,14)

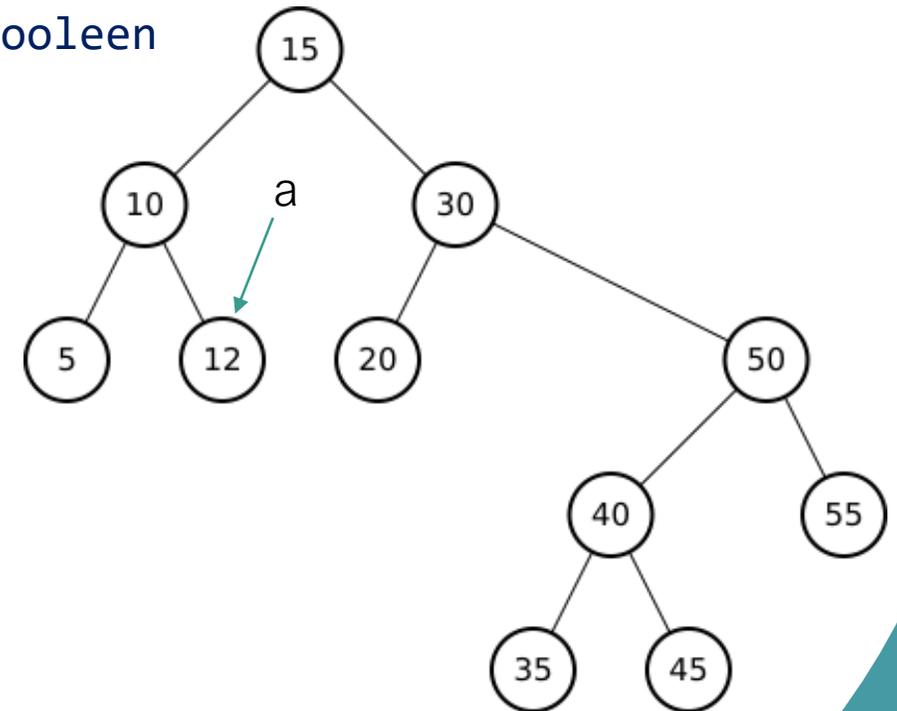
```
FONCTION recherche(a: pointeur sur Arbre, e: Element) : boolean
DEBUT
  SI (a EST EGAL A NULL) Alors
    RETOURNER FAUX
  SINON SI (element(a) EST EGAL A e) Alors
    RETOURNER VRAI
  SINON SI (e EST INF. STRICT. A element(a)) ALORS
    RETOURNER recherche(fg(a),e)
  SINON
    RETOURNER recherche(fd(a),e)
FIN SI
Fin
```



ABR : opération de recherche

- Algorithme : exemple : recherche (a,14)

```
FONCTION recherche(a: pointeur sur Arbre, e: Element) : boolean
DEBUT
  SI (a EST EGAL A NULL) Alors
    RETOURNER FAUX
  SINON SI (element(a) EST EGAL A e) Alors
    RETOURNER VRAI
  SINON SI (e EST INF. STRICT. A element(a)) ALORS
    RETOURNER recherche(fg(a),e)
  SINON
    RETOURNER recherche(fd(a),e)
FIN SI
Fin
```

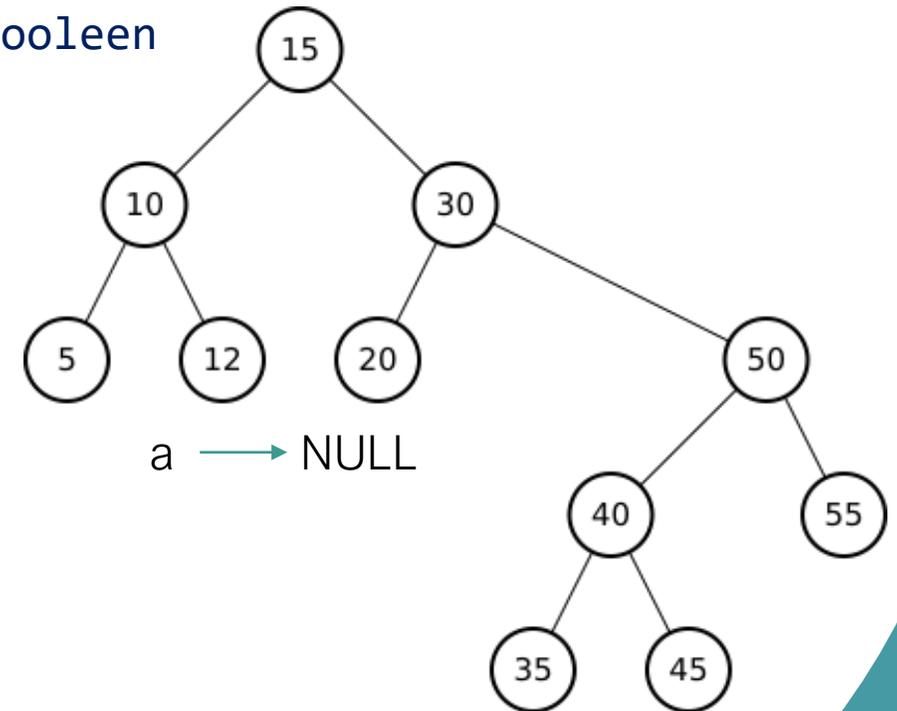


ABR : opération de recherche

- Algorithme : exemple : recherche (a,14)

```

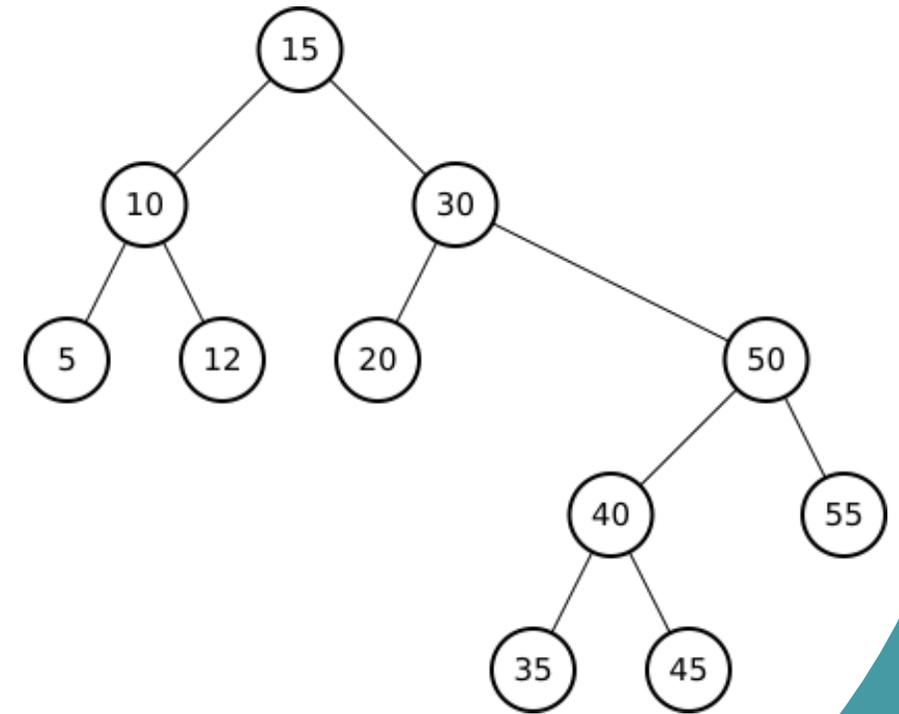
FONCTION recherche(a: pointeur sur Arbre, e: Element) : boolean
DEBUT
  SI (a EST EGAL A NULL) Alors
    RETOURNER FAUX
  SINON SI (element(a) EST EGAL A e) Alors
    RETOURNER VRAI
  SINON SI (e EST INF. STRICT. A element(a)) ALORS
    RETOURNER recherche(fg(a),e)
  SINON
    RETOURNER recherche(fd(a),e)
FIN SI
Fin
```



=> La fonction va retourner FAUX

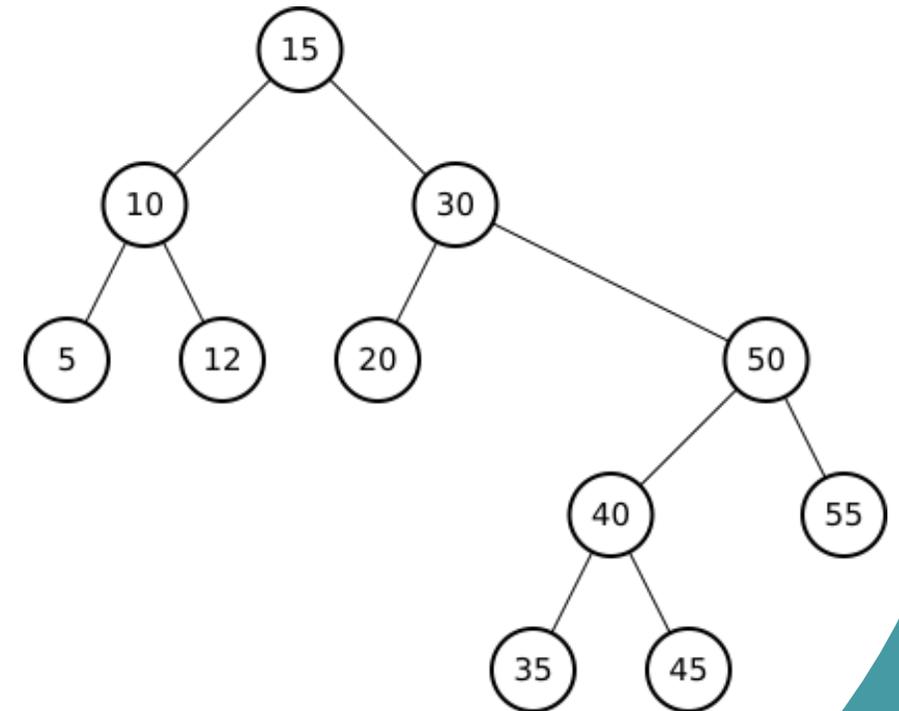
ABR : opération de recherche

- Complexité :
 - Meilleur des cas :
 - Pire cas :
 - Cas moyen :



ABR : opération de recherche

- Complexité :
 - Meilleur des cas : élément trouvé de suite : $O(1)$
 - Pire cas : Tous les éléments à parcourir : $O(n)$
(question : a quoi ressemble l'arbre dans ce cas?)
 - Cas moyen : $O(\log_2(n))$

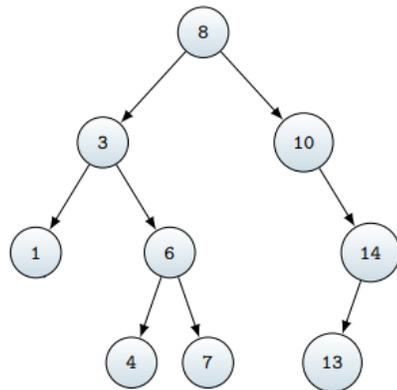


ABR : opération d'insertion

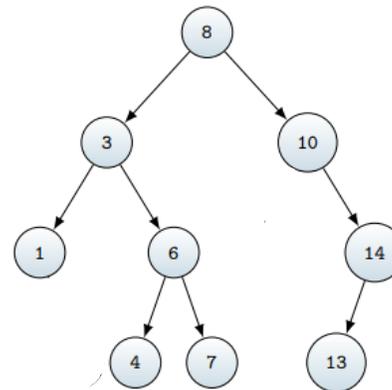
- L'insertion d'un nouvel élément dans l'arbre doit respecter les règles de l'ABR : il ne peut pas être inséré n'importe où.
- Principe:

On ajoute un nouvel élément en feuille, en respectant les propriétés d'un ABR :

- L'élément est déjà dans l'arbre, pas d'ajout
- On descend au bout de l'unique branche pour rajouter l'élément



Ajout de l'élément 2



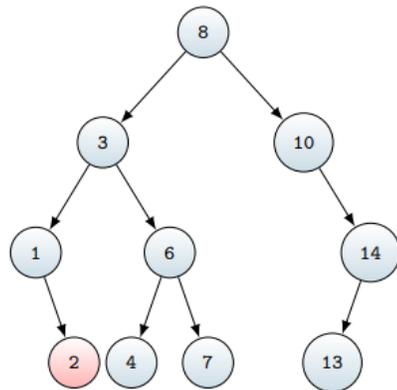
Ajout de l'élément 9

ABR : opération d'insertion

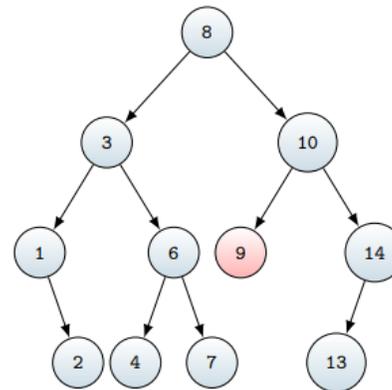
- L'insertion d'un nouvel élément dans l'arbre doit respecter les règles de l'ABR : il ne peut pas être inséré n'importe où.
- Principe:

On ajoute un nouvel élément en feuille, en respectant les propriétés d'un ABR :

- L'élément est déjà dans l'arbre, pas d'ajout
- On descend au bout de l'unique branche pour rajouter l'élément



Ajout de l'élément 2



Ajout de l'élément 9

ABR : opération d'insertion

- Algorithme :

```
FONCTION insertionABR(a: pointeur sur Arbre, e: Element) :
```

```
pointeur sur Arbre
```

```
DEBUT
```

```
  SI (a EST EGAL A NULL) ALORS
```

```
    RETOURNER creerArbre(e)
```

```
  SINON SI (e EST INF. STRICT. A element(a)) ALORS
```

```
    fg(a) ← insertionABR(fg(a), e)
```

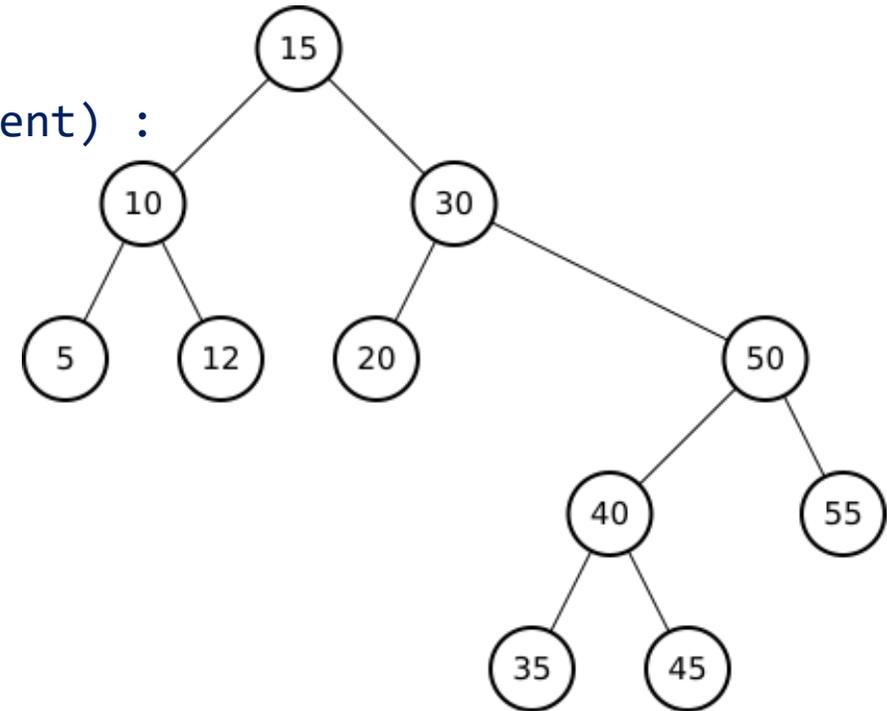
```
  SINON SI (e SUP. STRICT. A element(a)) Alors
```

```
    fd(a) ← insertionABR(fd(a), e)
```

```
  FIN SI
```

```
  RETOURNER a
```

```
FIN
```

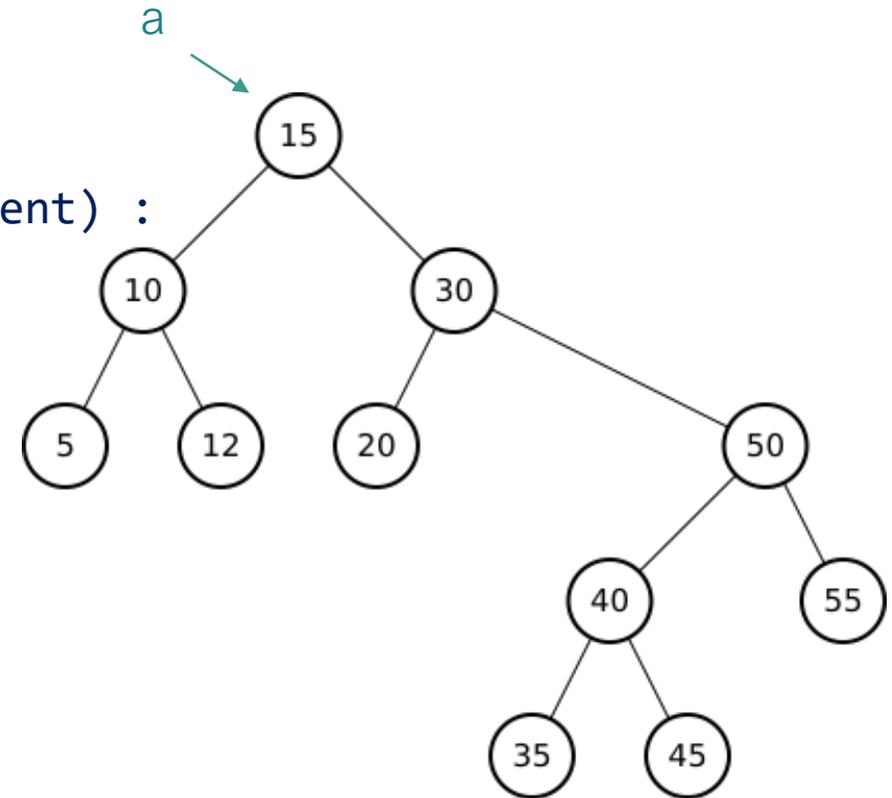


ABR : opération d'insertion

- Algorithme : Exemple : insertionABR(a, 11)

```
FONCTION insertionABR(a: pointeur sur Arbre, e: Element) :  
pointeur sur Arbre
```

```
→ DEBUT  
  SI (a EST EGAL A NULL) ALORS  
    RETOURNER creerArbre(e)  
  SINON SI (e EST INF. STRICT. A element(a)) ALORS  
    fg(a) ← insertionABR(fg(a), e)  
  SINON SI (e SUP. STRICT. A element(a)) Alors  
    fd(a) ← insertionABR(fd(a), e)  
  FIN SI  
  RETOURNER a  
FIN
```



ABR : opération d'insertion

- Algorithme : Exemple : insertionABR(a, 11)

FONCTION insertionABR(a: pointeur sur Arbre, e: Element) :

pointeur sur Arbre

DEBUT

➔ SI (a EST EGAL A NULL) ALORS
RETOURNER creerArbre(e)

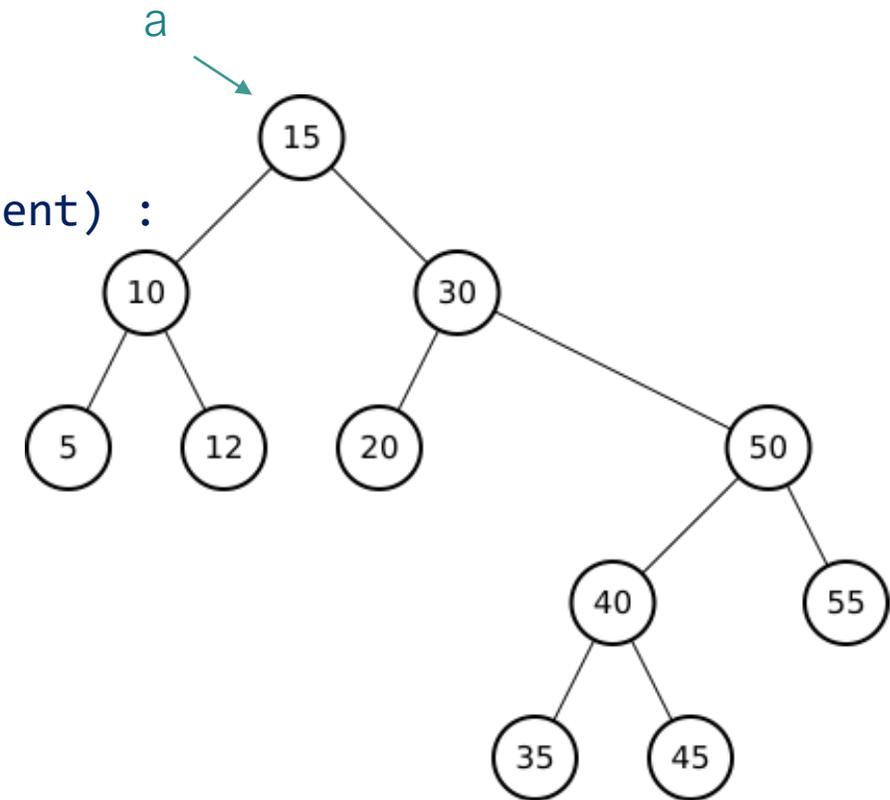
SINON SI (e EST INF. STRICT. A element(a)) ALORS
fg(a) ← insertionABR(fg(a), e)

SINON SI (e SUP. STRICT. A element(a)) Alors
fd(a) ← insertionABR(fd(a), e)

FIN SI

RETOURNER a

FIN



ABR : opération d'insertion

- Algorithme : Exemple : insertionABR(a, 11)

FONCTION insertionABR(a: pointeur sur Arbre, e: Element) :

pointeur sur Arbre

DEBUT

SI (a EST EGAL A NULL) ALORS

RETOURNER creerArbre(e)

➔ SINON SI (e EST INF. STRICT. A element(a)) ALORS

fg(a) ← insertionABR(fg(a), e)

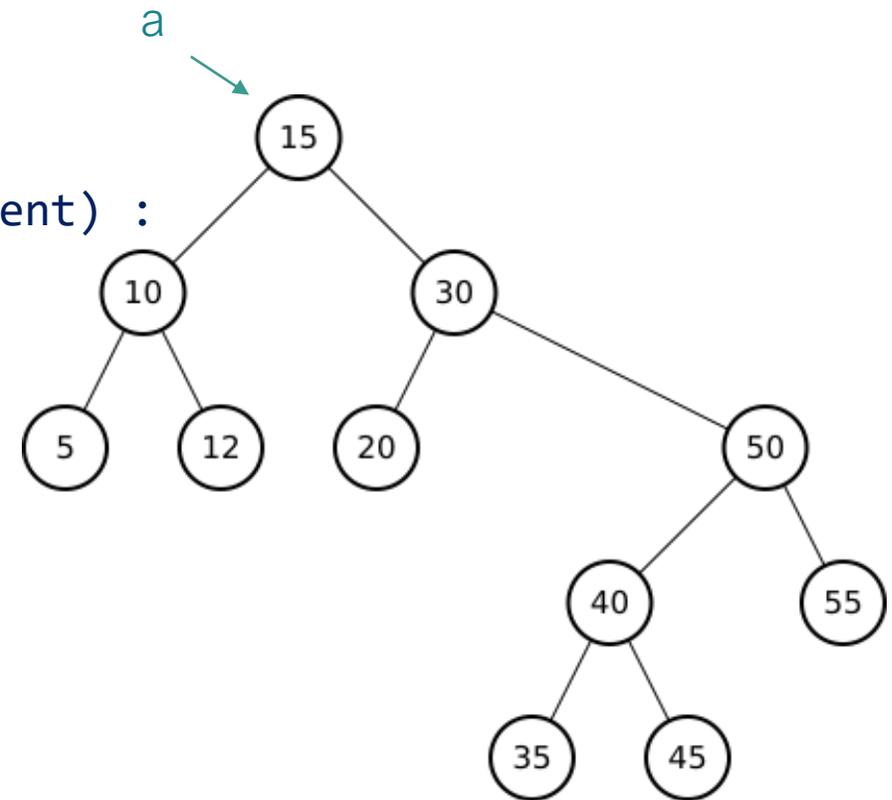
SINON SI (e SUP. STRICT. A element(a)) Alors

fd(a) ← insertionABR(fd(a), e)

FIN SI

RETOURNER a

FIN



ABR : opération d'insertion

- Algorithme : Exemple : insertionABR(a, 11)

```
FONCTION insertionABR(a: pointeur sur Arbre, e: Element) :
```

```
pointeur sur Arbre
```

```
DEBUT
```

```
SI (a EST EGAL A NULL) ALORS
```

```
    RETOURNER creerArbre(e)
```

```
SINON SI (e EST INF. STRICT. A element(a)) ALORS
```

```
    → fg(a) ← insertionABR(fg(a), e)
```

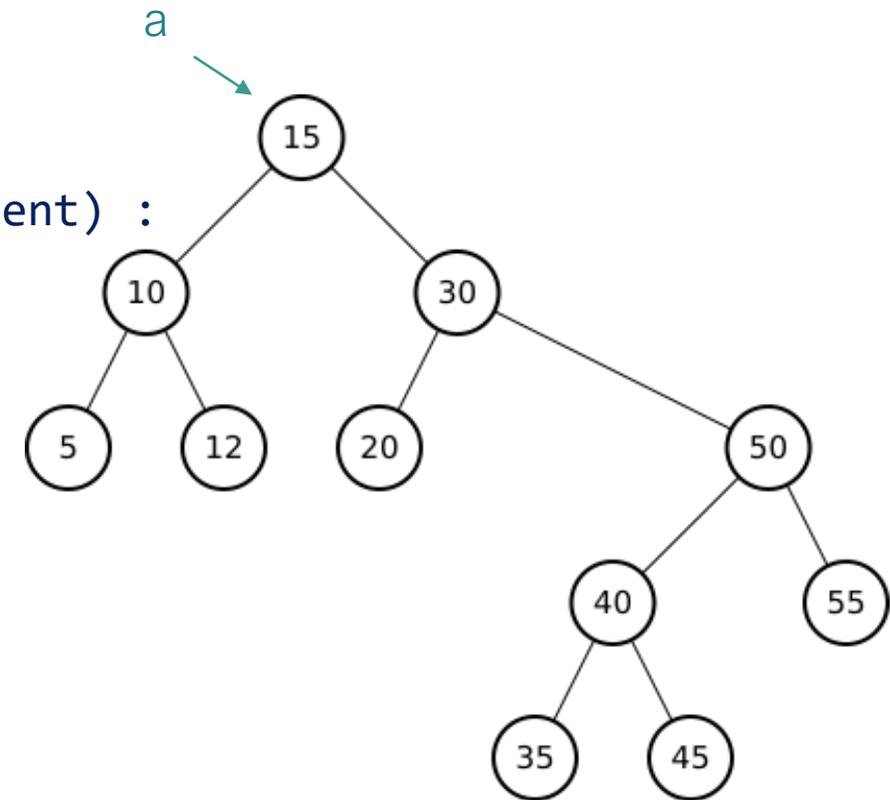
```
SINON SI (e SUP. STRICT. A element(a)) Alors
```

```
    fd(a) ← insertionABR(fd(a), e)
```

```
FIN SI
```

```
RETOURNER a
```

```
FIN
```



ABR : opération d'insertion

- Algorithme : Exemple : insertionABR(a, 11)

```
FONCTION insertionABR(a: pointeur sur Arbre, e: Element) :  
pointeur sur Arbre
```

```
→ DEBUT
```

```
SI (a EST EGAL A NULL) ALORS  
    RETOURNER creerArbre(e)
```

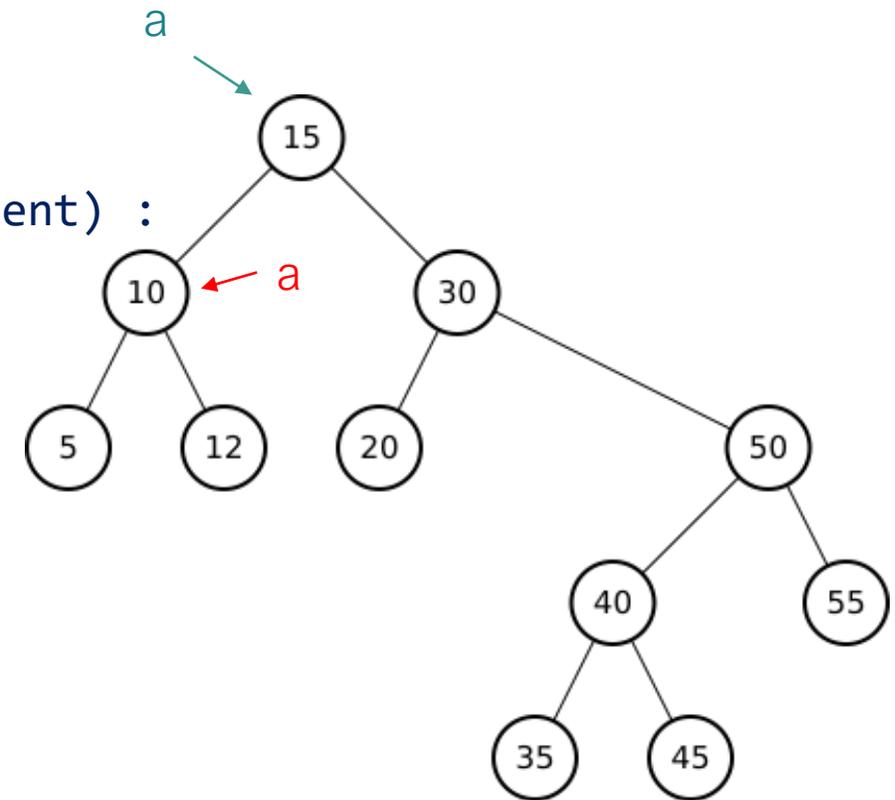
```
SINON SI (e EST INF. STRICT. A element(a)) ALORS  
    → fg(a) ← insertionABR(fg(a), e)
```

```
SINON SI (e SUP. STRICT. A element(a)) Alors  
    fd(a) ← insertionABR(fd(a), e)
```

```
FIN SI
```

```
RETOURNER a
```

```
FIN
```



ABR : opération d'insertion

- Algorithme : Exemple : insertionABR(a, 11)

FONCTION insertionABR(a: pointeur sur Arbre, e: Element) :

pointeur sur Arbre

DEBUT

→ SI (a EST EGAL A NULL) ALORS
RETOURNER creerArbre(e)

SINON SI (e EST INF. STRICT. A element(a)) ALORS

→ fg(a) ← insertionABR(fg(a), e)

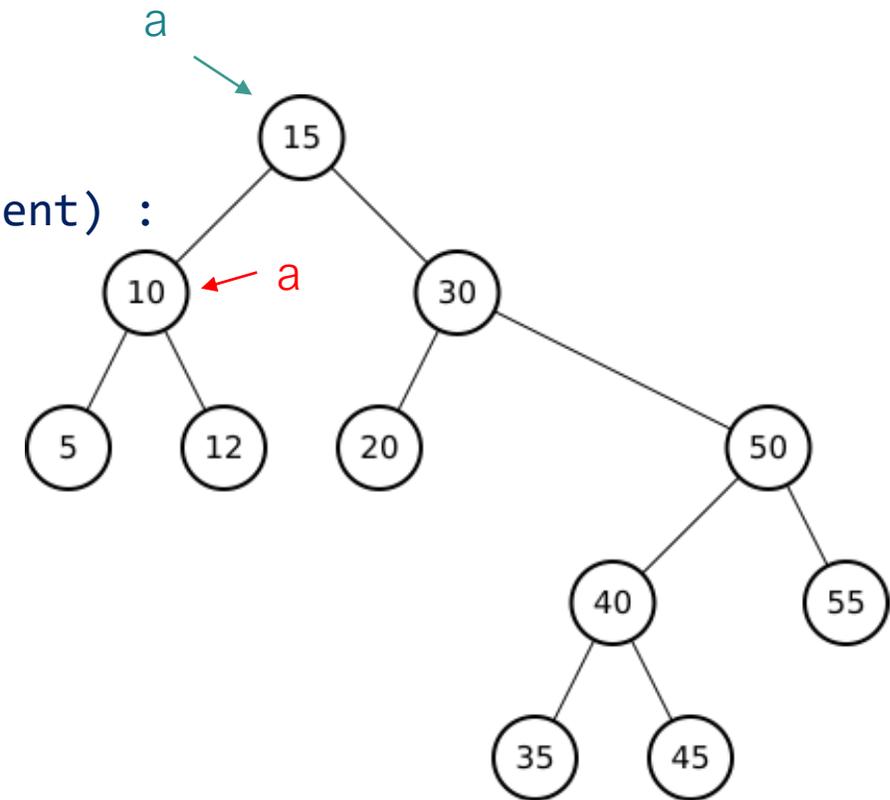
SINON SI (e SUP. STRICT. A element(a)) Alors

fd(a) ← insertionABR(fd(a), e)

FIN SI

RETOURNER a

FIN



ABR : opération d'insertion

- Algorithme : Exemple : insertionABR(a, 11)

FONCTION insertionABR(a: pointeur sur Arbre, e: Element) :

pointeur sur Arbre

DEBUT

SI (a EST EGAL A NULL) ALORS

RETOURNER creerArbre(e)

→ SINON SI (e EST INF. STRICT. A element(a)) ALORS

→ fg(a) ← insertionABR(fg(a), e)

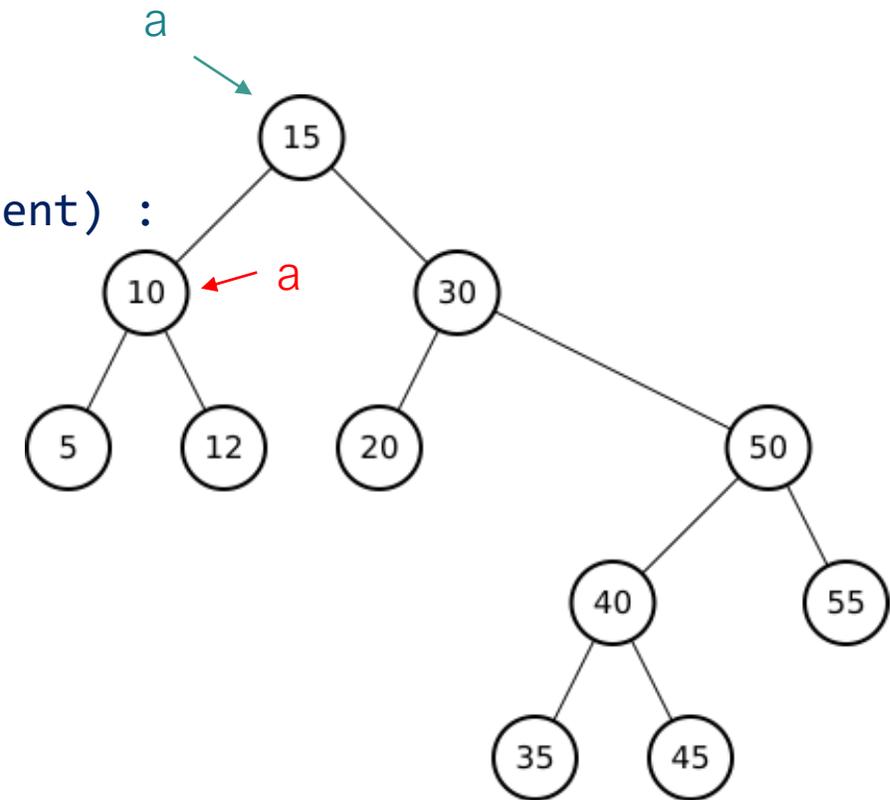
SINON SI (e SUP. STRICT. A element(a)) Alors

fd(a) ← insertionABR(fd(a), e)

FIN SI

RETOURNER a

FIN



ABR : opération d'insertion

- Algorithme : Exemple : insertionABR(a, 11)

FONCTION insertionABR(a: pointeur sur Arbre, e: Element) :

pointeur sur Arbre

DEBUT

SI (a EST EGAL A NULL) ALORS

RETOURNER creerArbre(e)

SINON SI (e EST INF. STRICT. A element(a)) ALORS

→ fg(a) ← insertionABR(fg(a), e)

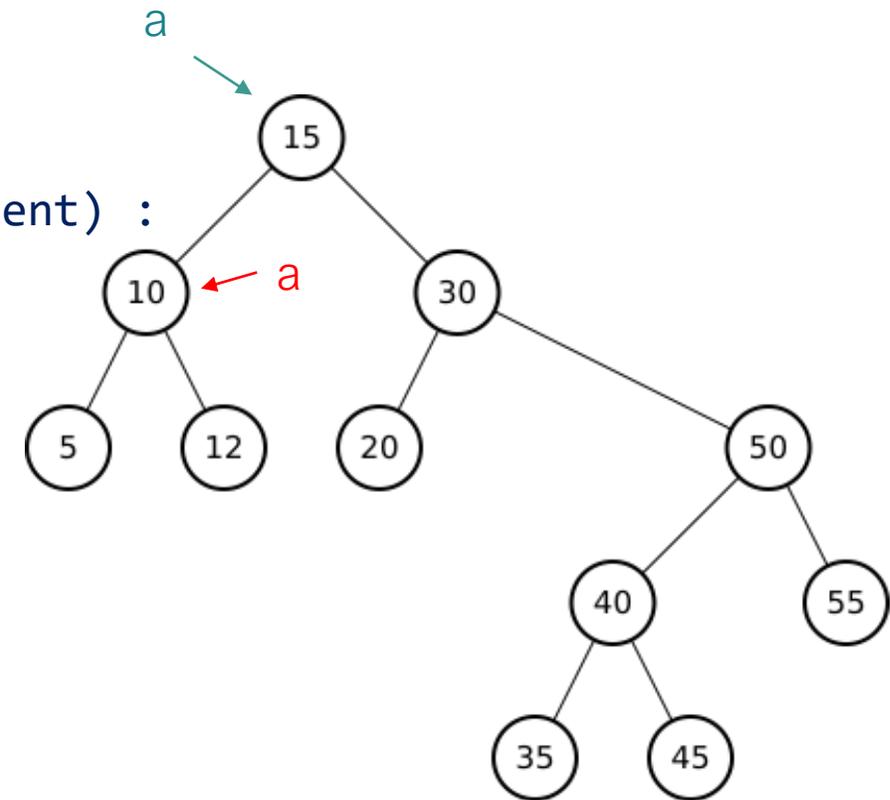
→ SINON SI (e SUP. STRICT. A element(a)) Alors

fd(a) ← insertionABR(fd(a), e)

FIN SI

RETOURNER a

FIN



ABR : opération d'insertion

- Algorithme : Exemple : insertionABR(a, 11)

FONCTION insertionABR(a: pointeur sur Arbre, e: Element) :

pointeur sur Arbre

DEBUT

SI (a EST EGAL A NULL) ALORS

RETOURNER creerArbre(e)

SINON SI (e EST INF. STRICT. A element(a)) ALORS

→ fg(a) ← insertionABR(fg(a), e)

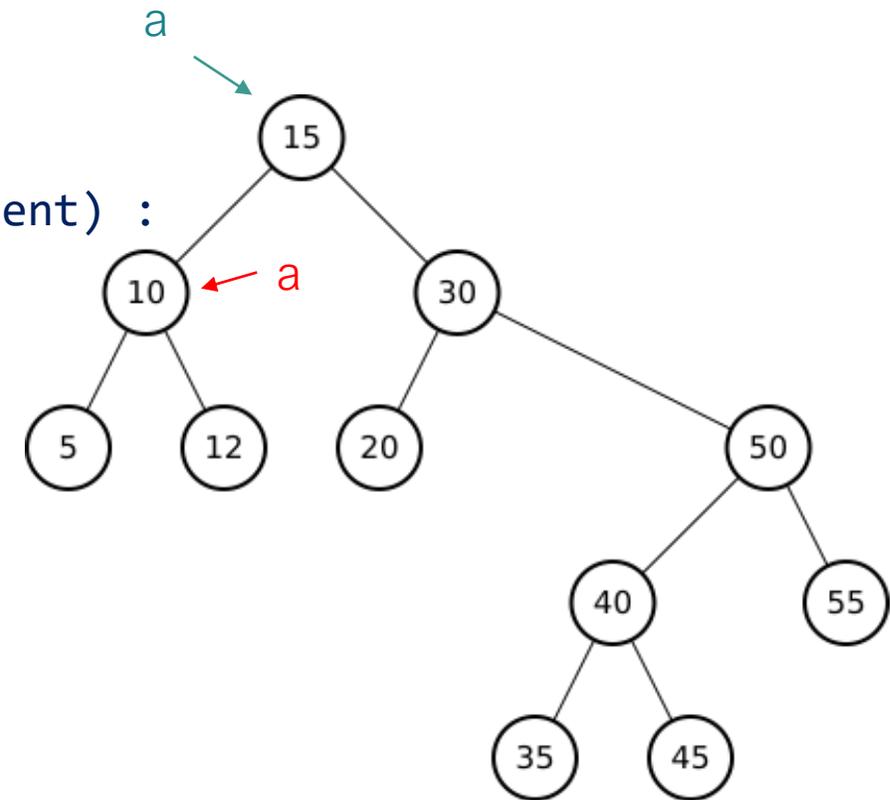
SINON SI (e SUP. STRICT. A element(a)) Alors

→ fd(a) ← insertionABR(fd(a), e)

FIN SI

RETOURNER a

FIN



ABR : opération d'insertion

- Algorithme : Exemple : insertionABR(a, 11)

```
FONCTION insertionABR(a: pointeur sur Arbre, e: Element) :  
pointeur sur Arbre
```

```
→ DEBUT
```

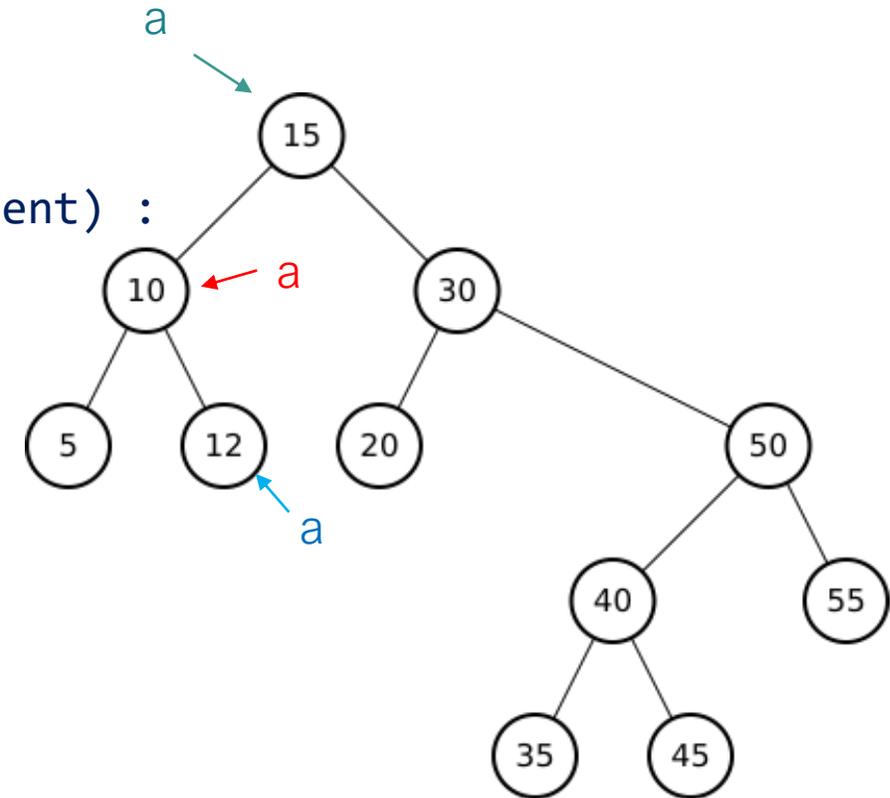
```
SI (a EST EGAL A NULL) ALORS  
    RETOURNER creerArbre(e)
```

```
SINON SI (e EST INF. STRICT. A element(a)) ALORS  
    → fg(a) ← insertionABR(fg(a), e)
```

```
SINON SI (e SUP. STRICT. A element(a)) Alors  
    → fd(a) ← insertionABR(fd(a), e)
```

```
FIN SI  
RETOURNER a
```

```
FIN
```



ABR : opération d'insertion

- Algorithme : Exemple : insertionABR(a, 11)

FONCTION insertionABR(a: pointeur sur Arbre, e: Element) :

pointeur sur Arbre

DEBUT

➡ SI (a EST EGAL A NULL) ALORS
RETOURNER creerArbre(e)

SINON SI (e EST INF. STRICT. A element(a)) ALORS

➡ fg(a) ← insertionABR(fg(a), e)

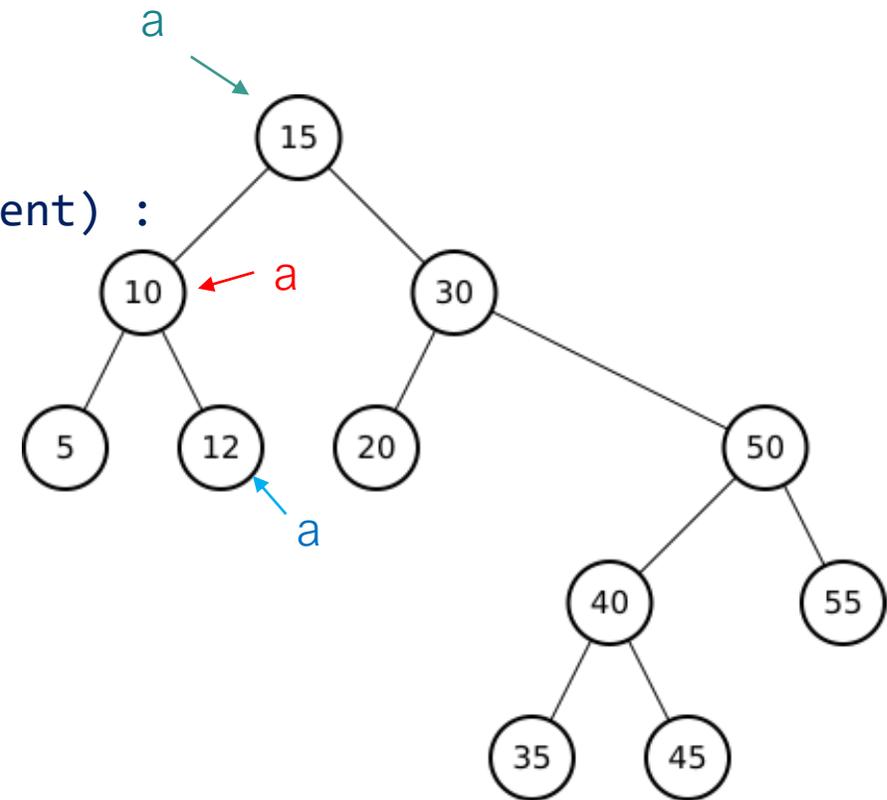
SINON SI (e SUP. STRICT. A element(a)) Alors

➡ fd(a) ← insertionABR(fd(a), e)

FIN SI

RETOURNER a

FIN



ABR : opération d'insertion

- Algorithme : Exemple : insertionABR(a, 11)

FONCTION insertionABR(a: pointeur sur Arbre, e: Element) :

pointeur sur Arbre

DEBUT

SI (a EST EGAL A NULL) ALORS

RETOURNER creerArbre(e)

➡ SINON SI (e EST INF. STRICT. A element(a)) ALORS

➡ fg(a) ← insertionABR(fg(a), e)

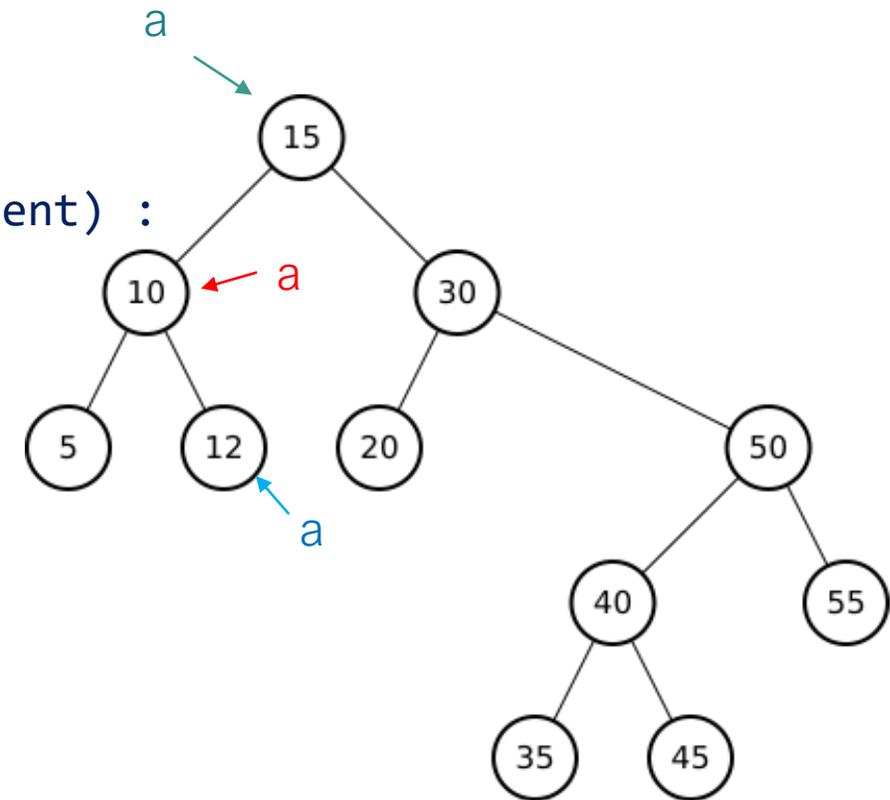
SINON SI (e SUP. STRICT. A element(a)) Alors

➡ fd(a) ← insertionABR(fd(a), e)

FIN SI

RETOURNER a

FIN



ABR : opération d'insertion

- Algorithme : Exemple : insertionABR(a, 11)

FONCTION insertionABR(a: pointeur sur Arbre, e: Element) :

pointeur sur Arbre

DEBUT

SI (a EST EGAL A NULL) ALORS

RETOURNER creerArbre(e)

SINON SI (e EST INF. STRICT. A element(a)) ALORS

→ → fg(a) ← insertionABR(fg(a), e)

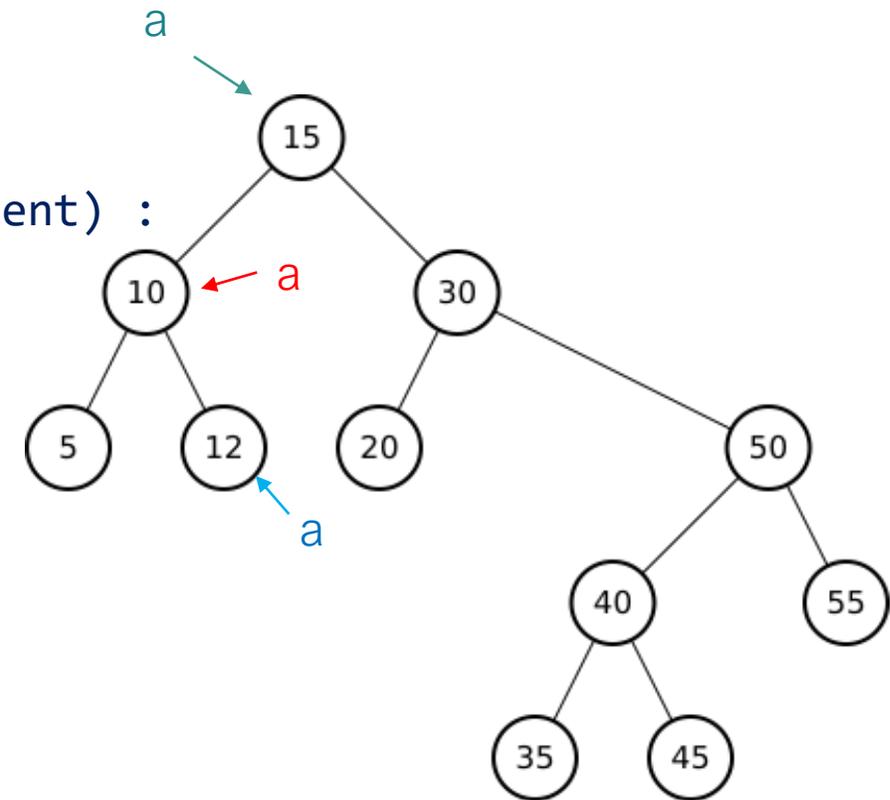
SINON SI (e SUP. STRICT. A element(a)) Alors

→ fd(a) ← insertionABR(fd(a), e)

FIN SI

RETOURNER a

FIN



ABR : opération d'insertion

- Algorithme : Exemple : insertionABR(a, 11)

FONCTION insertionABR(a: pointeur sur Arbre, e: Element) :

pointeur sur Arbre

DEBUT

SI (a EST EGAL A NULL) ALORS

RETOURNER creerArbre(e)

SINON SI (e EST INF. STRICT. A element(a)) ALORS

→ → fg(a) ← insertionABR(fg(a), e)

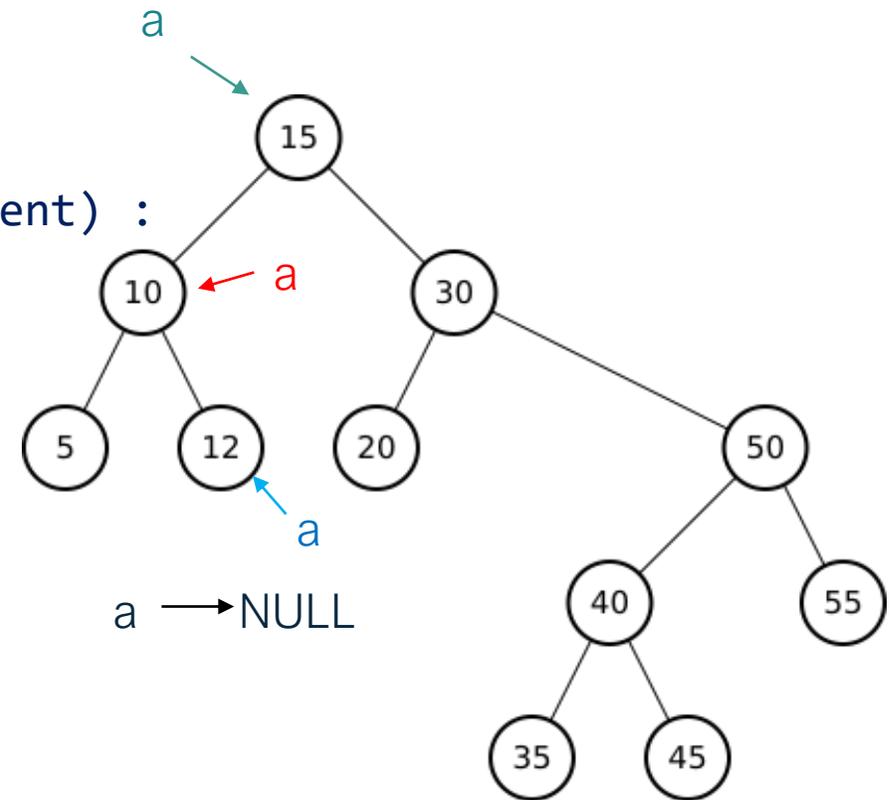
SINON SI (e SUP. STRICT. A element(a)) Alors

→ fd(a) ← insertionABR(fd(a), e)

FIN SI

RETOURNER a

FIN



ABR : opération d'insertion

- Algorithme : Exemple : insertionABR(a, 11)

FONCTION insertionABR(a: pointeur sur Arbre, e: Element) :

pointeur sur Arbre

DEBUT

➔ SI (a EST EGAL A NULL) ALORS
RETOURNER creerArbre(e)

SINON SI (e EST INF. STRICT. A element(a)) ALORS

➔➔ fg(a) ← insertionABR(fg(a), e)

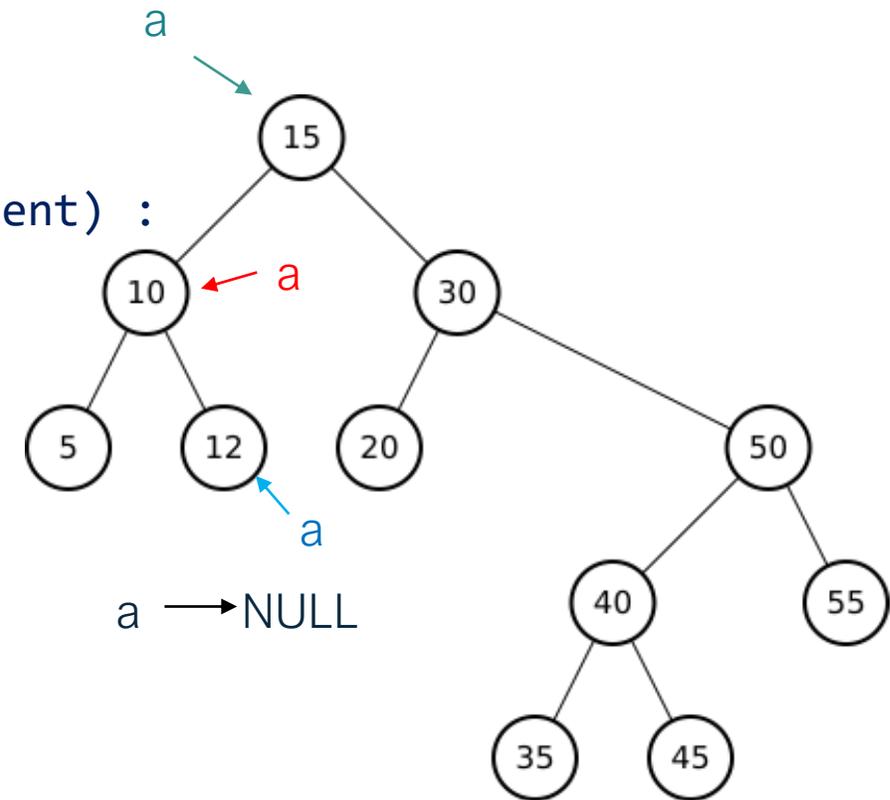
SINON SI (e SUP. STRICT. A element(a)) Alors

➔➔ fd(a) ← insertionABR(fd(a), e)

FIN SI

RETOURNER a

FIN



ABR : opération d'insertion

- Algorithme : Exemple : insertionABR(a, 11)

FONCTION insertionABR(a: pointeur sur Arbre, e: Element) :

pointeur sur Arbre

DEBUT

SI (a EST EGAL A NULL) ALORS

 ➡ RETOURNER creerArbre(e)

SINON SI (e EST INF. STRICT. A element(a)) ALORS

 ➡➡ fg(a) ← insertionABR(fg(a), e)

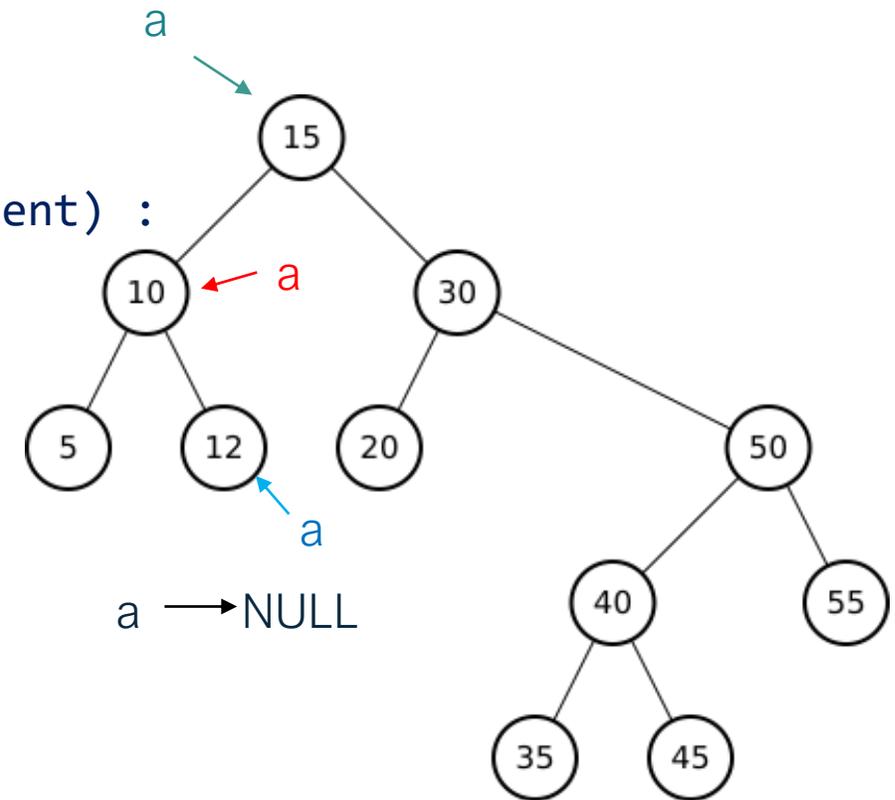
SINON SI (e SUP. STRICT. A element(a)) Alors

 ➡➡ fd(a) ← insertionABR(fd(a), e)

FIN SI

RETOURNER a

FIN



ABR : opération d'insertion

- Algorithme : Exemple : insertionABR(a, 11)

FONCTION insertionABR(a: pointeur sur Arbre, e: Element) :

pointeur sur Arbre

DEBUT

SI (a EST EGAL A NULL) ALORS

 ➡ RETOURNER creerArbre(e)

SINON SI (e EST INF. STRICT. A element(a)) ALORS

 ➡➡ fg(a) ← insertionABR(fg(a), e)

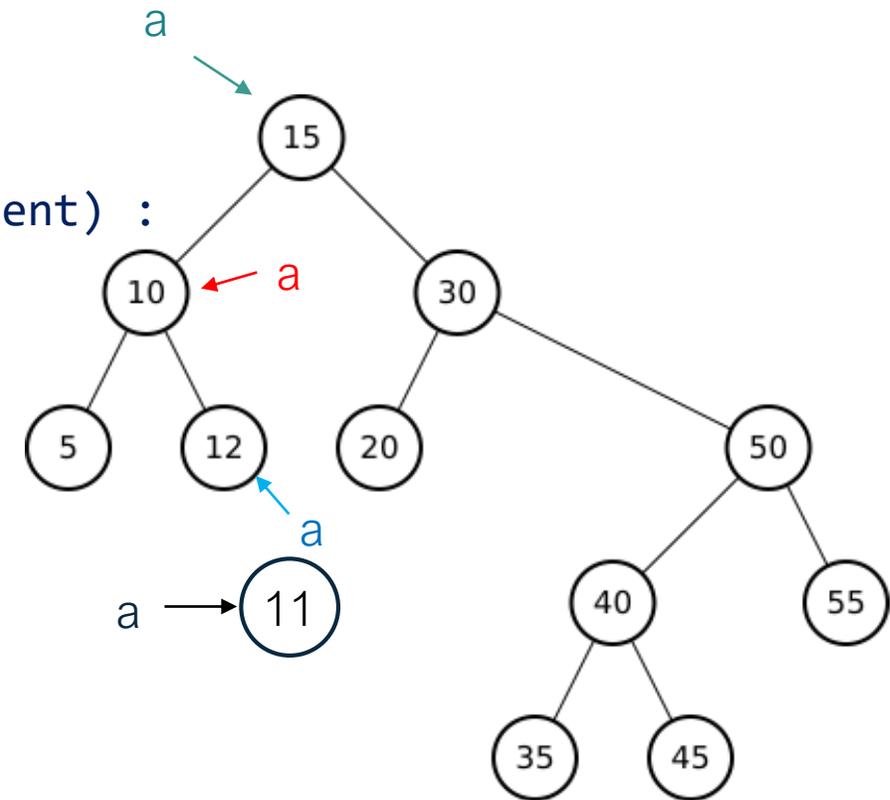
SINON SI (e SUP. STRICT. A element(a)) Alors

 ➡➡ fd(a) ← insertionABR(fd(a), e)

FIN SI

RETOURNER a

FIN



ABR : opération d'insertion

- Algorithme : Exemple : insertionABR(a, 11)

FONCTION insertionABR(a: pointeur sur Arbre, e: Element) :

pointeur sur Arbre

DEBUT

SI (a EST EGAL A NULL) ALORS

RETOURNER creerArbre(e)

SINON SI (e EST INF. STRICT. A element(a)) ALORS

→ → fg(a) ← insertionABR(fg(a), e)

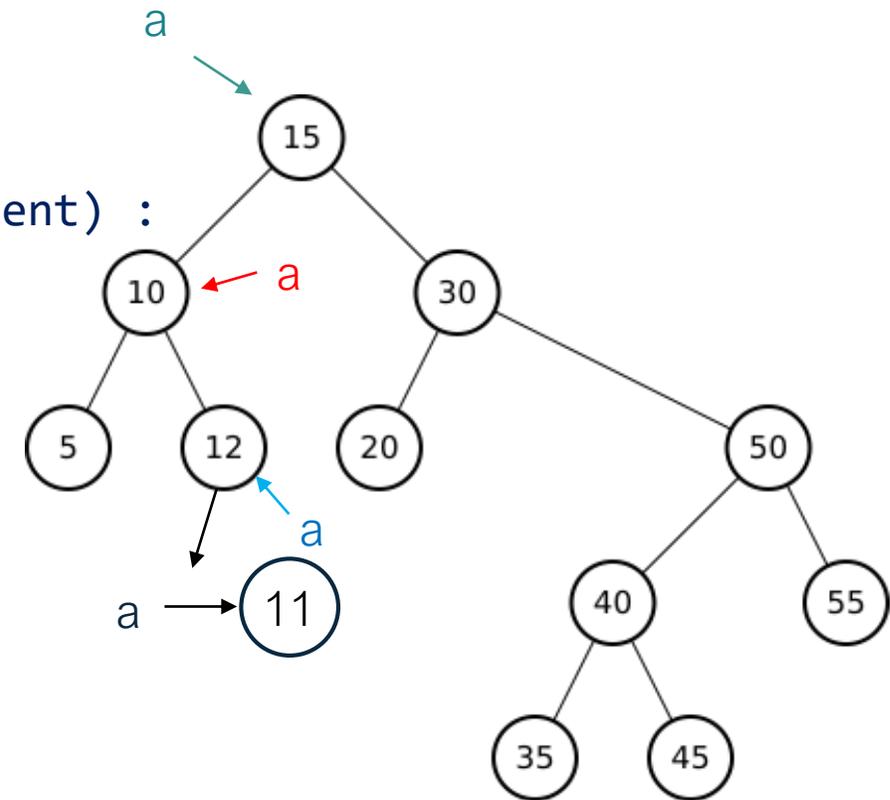
SINON SI (e SUP. STRICT. A element(a)) Alors

→ fd(a) ← insertionABR(fd(a), e)

FIN SI

→ RETOURNER a

FIN



ABR : opération d'insertion

- Algorithme : Exemple : insertionABR(a, 11)

FONCTION insertionABR(a: pointeur sur Arbre, e: Element) :

pointeur sur Arbre

DEBUT

SI (a EST EGAL A NULL) ALORS

RETOURNER creerArbre(e)

SINON SI (e EST INF. STRICT. A element(a)) ALORS

→ → fg(a) ← insertionABR(fg(a), e)

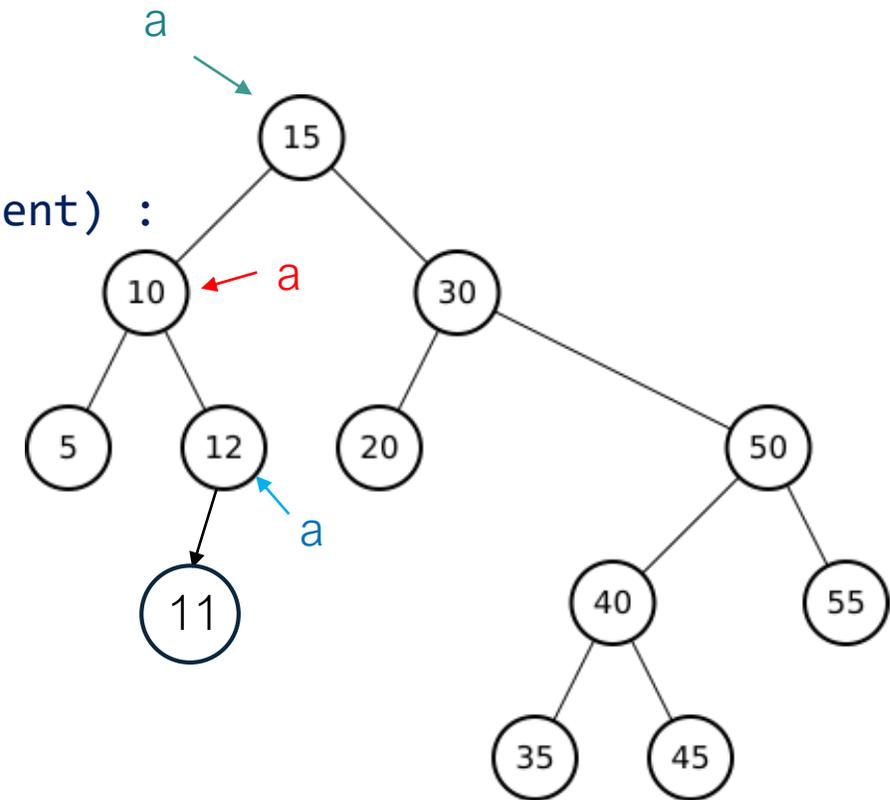
SINON SI (e SUP. STRICT. A element(a)) Alors

→ fd(a) ← insertionABR(fd(a), e)

FIN SI

RETOURNER a

FIN



ABR : opération d'insertion

- Algorithme : Exemple : insertionABR(a, 11)

FONCTION insertionABR(a: pointeur sur Arbre, e: Element) :

pointeur sur Arbre

DEBUT

SI (a EST EGAL A NULL) ALORS

RETOURNER creerArbre(e)

SINON SI (e EST INF. STRICT. A element(a)) ALORS

➡ $fg(a) \leftarrow insertionABR(fg(a), e)$

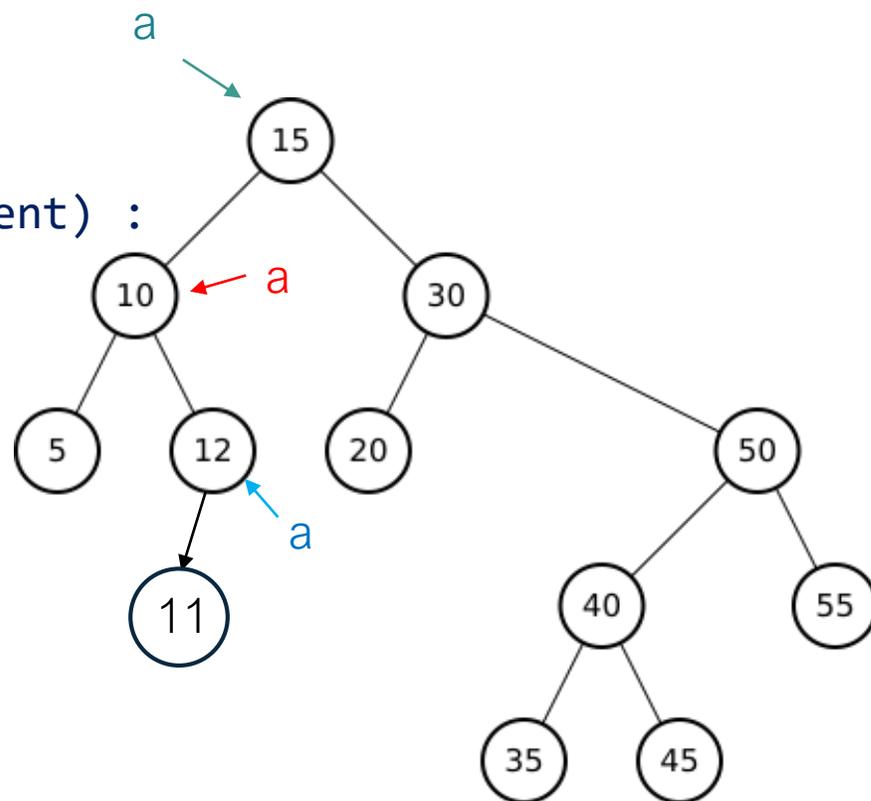
SINON SI (e SUP. STRICT. A element(a)) Alors

➡ $fd(a) \leftarrow insertionABR(fd(a), e)$

FIN SI

➡ RETOURNER a

FIN



ABR : opération d'insertion

- Algorithme : Exemple : insertionABR(a, 11)

FONCTION insertionABR(a: pointeur sur Arbre, e: Element) :

pointeur sur Arbre

DEBUT

SI (a EST EGAL A NULL) ALORS

RETOURNER creerArbre(e)

SINON SI (e EST INF. STRICT. A element(a)) ALORS

→ fg(a) ← insertionABR(fg(a), e)

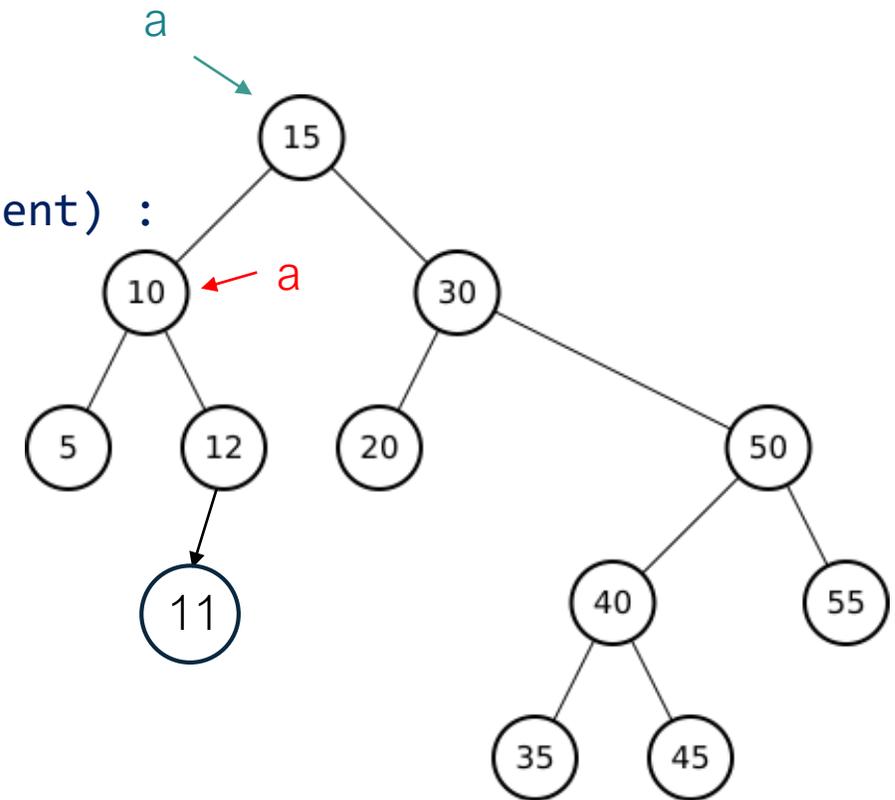
SINON SI (e SUP. STRICT. A element(a)) Alors

→ fd(a) ← insertionABR(fd(a), e)

FIN SI

RETOURNER a

FIN



ABR : opération d'insertion

- Algorithme : Exemple : insertionABR(a, 11)

FONCTION insertionABR(a: pointeur sur Arbre, e: Element) :

pointeur sur Arbre

DEBUT

SI (a EST EGAL A NULL) ALORS

RETOURNER creerArbre(e)

SINON SI (e EST INF. STRICT. A element(a)) ALORS

→ fg(a) ← insertionABR(fg(a), e)

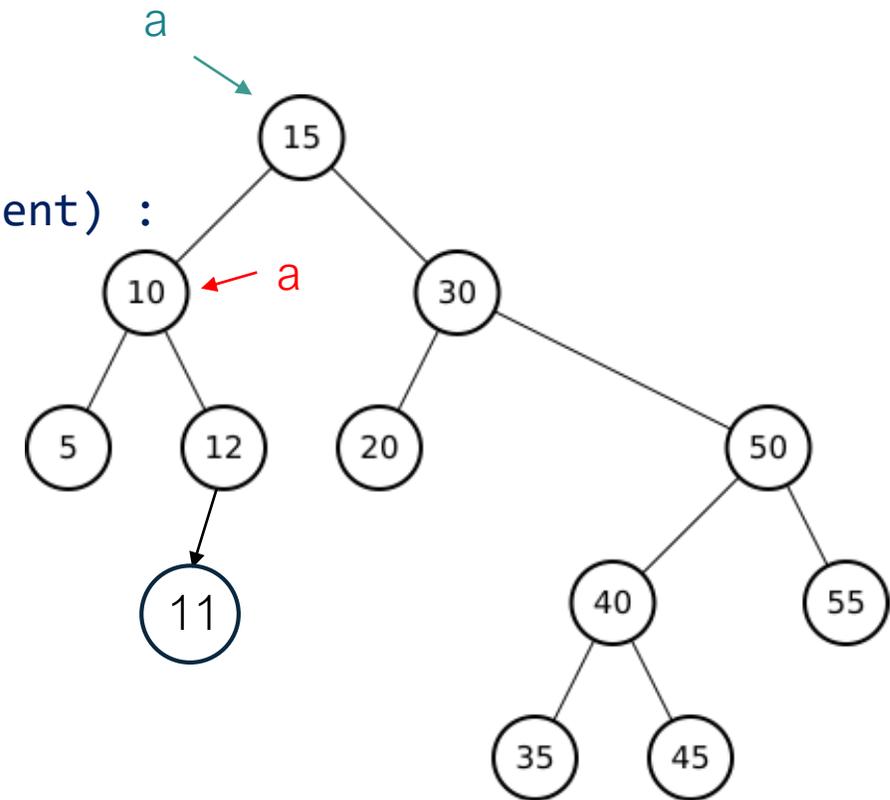
SINON SI (e SUP. STRICT. A element(a)) Alors

fd(a) ← insertionABR(fd(a), e)

FIN SI

→ RETOURNER a

FIN



ABR : opération d'insertion

- Algorithme : Exemple : insertionABR(a, 11)

FONCTION insertionABR(a: pointeur sur Arbre, e: Element) :

pointeur sur Arbre

DEBUT

SI (a EST EGAL A NULL) ALORS

RETOURNER creerArbre(e)

SINON SI (e EST INF. STRICT. A element(a)) ALORS

→ fg(a) ← insertionABR(fg(a), e)

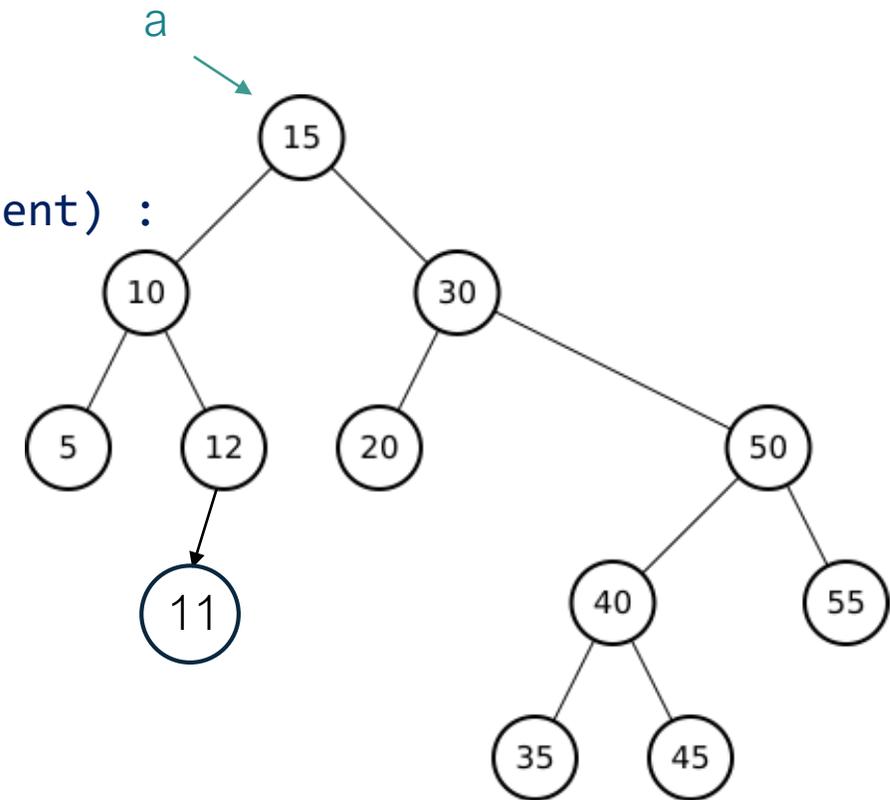
SINON SI (e SUP. STRICT. A element(a)) Alors

fd(a) ← insertionABR(fd(a), e)

FIN SI

RETOURNER a

FIN



ABR : opération d'insertion

- Algorithme : Exemple : insertionABR(a, 11)

FONCTION insertionABR(a: pointeur sur Arbre, e: Element) :

pointeur sur Arbre

DEBUT

SI (a EST EGAL A NULL) ALORS

RETOURNER creerArbre(e)

SINON SI (e EST INF. STRICT. A element(a)) ALORS

fg(a) ← insertionABR(fg(a), e)

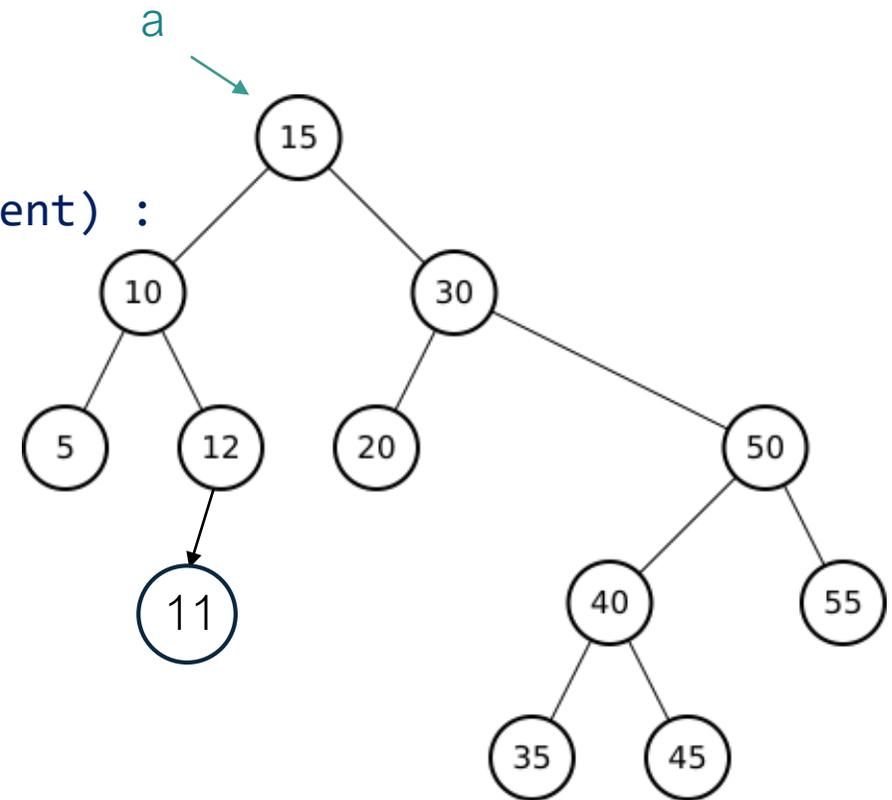
SINON SI (e SUP. STRICT. A element(a)) Alors

fd(a) ← insertionABR(fd(a), e)

FIN SI

→ RETOURNER a

FIN



ABR : opération d'insertion

- Algorithme : Exemple : insertionABR(a, 11)

```
FONCTION insertionABR(a: pointeur sur Arbre, e: Element) :
```

```
pointeur sur Arbre
```

```
DEBUT
```

```
  SI (a EST EGAL A NULL) ALORS
```

```
    RETOURNER creerArbre(e)
```

```
  SINON SI (e EST INF. STRICT. A element(a)) ALORS
```

```
    fg(a) ← insertionABR(fg(a), e)
```

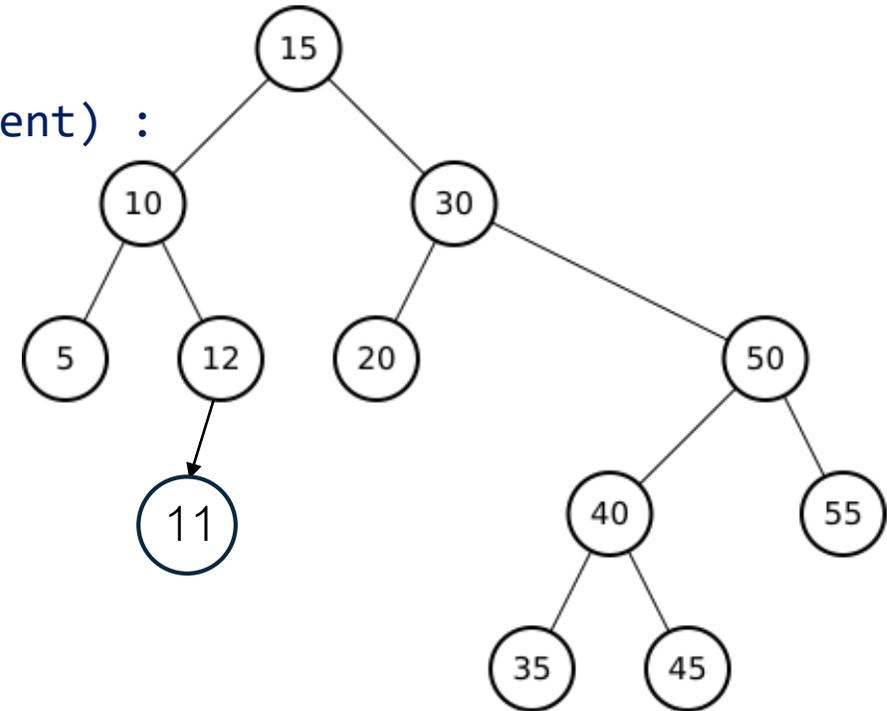
```
  SINON SI (e SUP. STRICT. A element(a)) Alors
```

```
    fd(a) ← insertionABR(fd(a), e)
```

```
  FIN SI
```

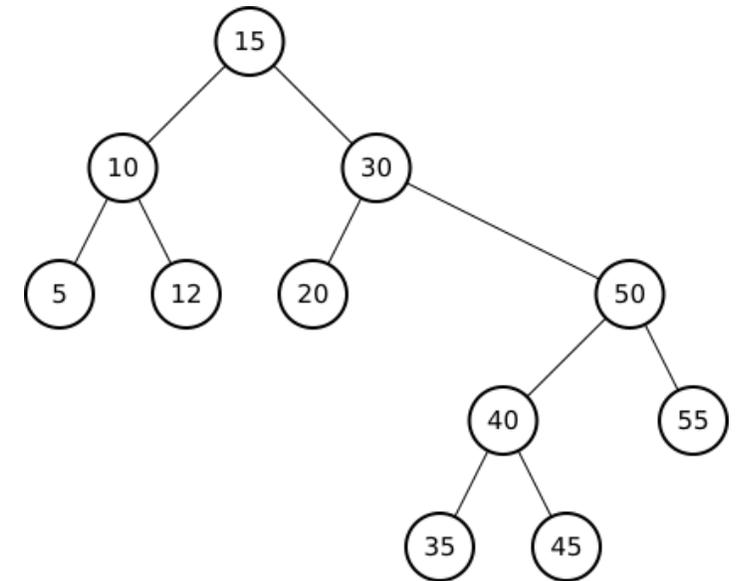
```
  RETOURNER a
```

```
FIN
```



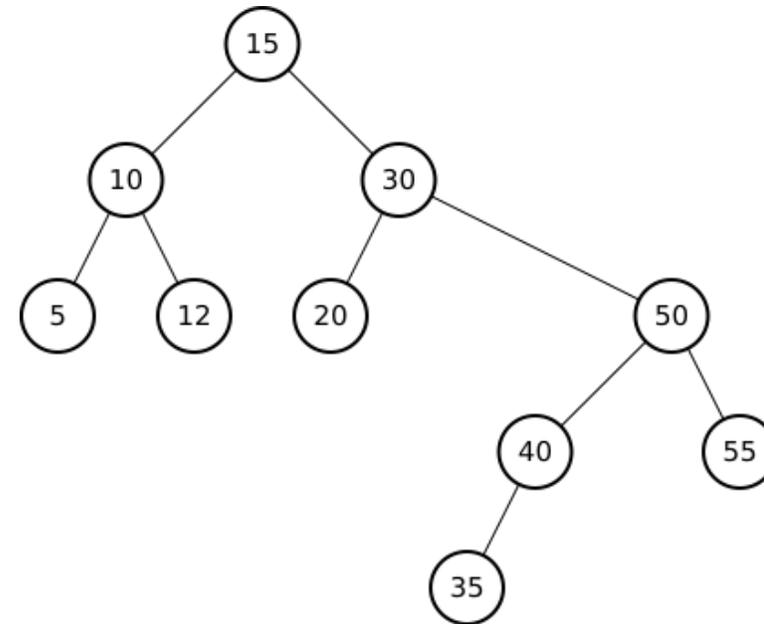
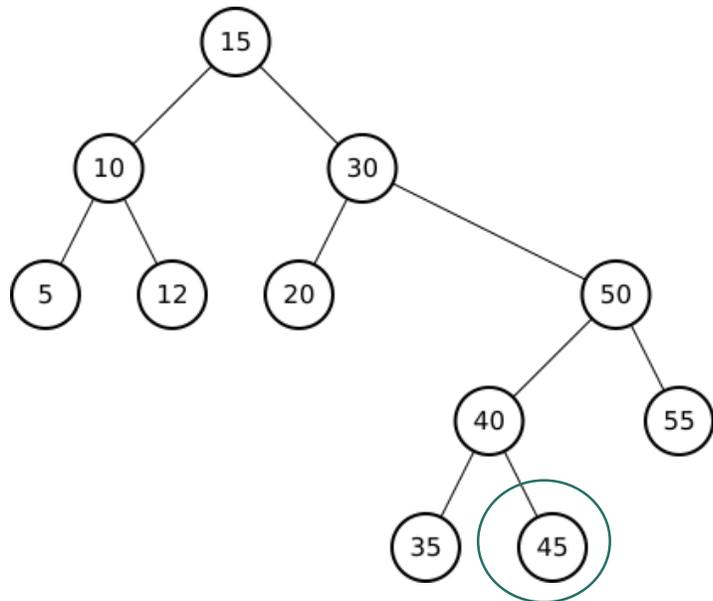
ABR : opération de suppression

- La suppression d'un nœud dans un arbre binaire consiste à supprimer uniquement le nœud ciblé : ses fils doivent rester dans l'arbre.
- Il faut maintenir la relation d'ordre imposée par l'ABR !
- Il y a plusieurs cas à considérer:
 - Suppression d'une feuille
 - Le nœud à supprimer a un fils
 - Le nœud à supprimer a deux fils



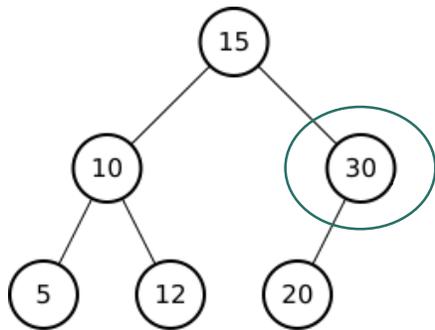
ABR : opération de suppression

- Il y a plusieurs cas à considérer:
 - **Suppression d'une feuille**
 - Le nœud à supprimer a un fils
 - Le nœud à supprimer a deux fils
- Aucune réorganisation de l'arbre est nécessaire :



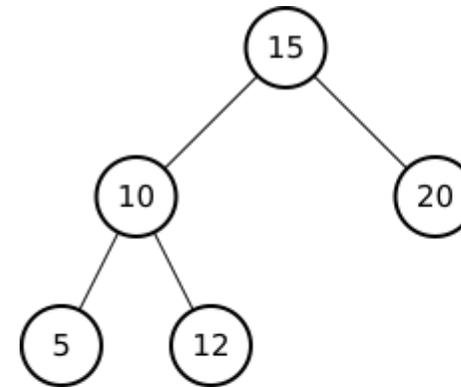
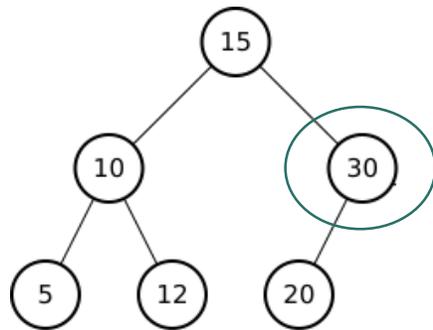
ABR : opération de suppression

- Il y a plusieurs cas à considérer:
 - Suppression d'une feuille
 - **Le nœud à supprimer a un fils**
 - Le nœud à supprimer a deux fils



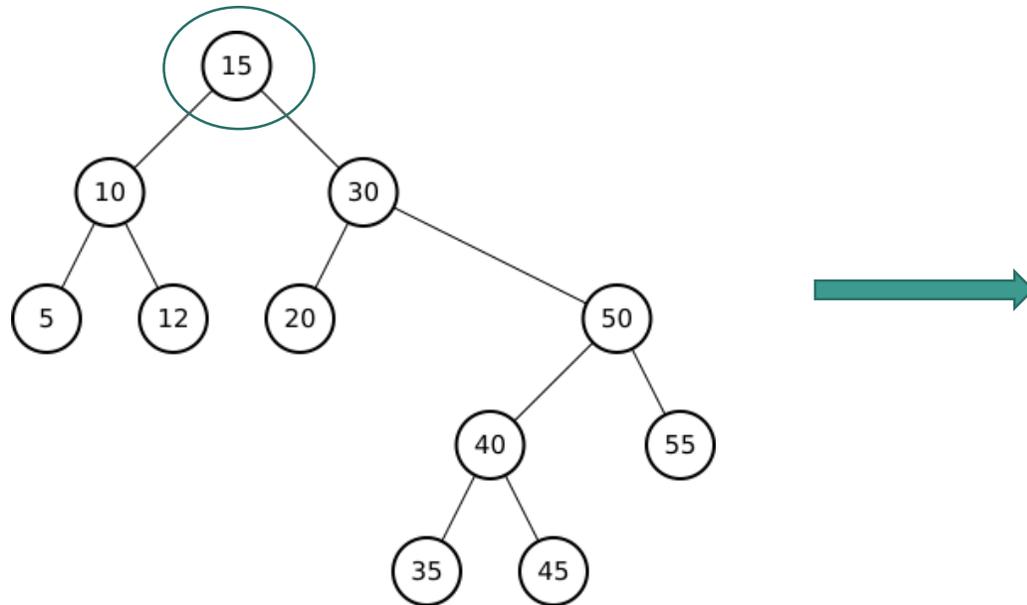
ABR : opération de suppression

- Il y a plusieurs cas à considérer:
 - Suppression d'une feuille
 - **Le nœud à supprimer a un fils**
 - Le nœud à supprimer a deux fils
- On remplace le nœud par son fils unique :



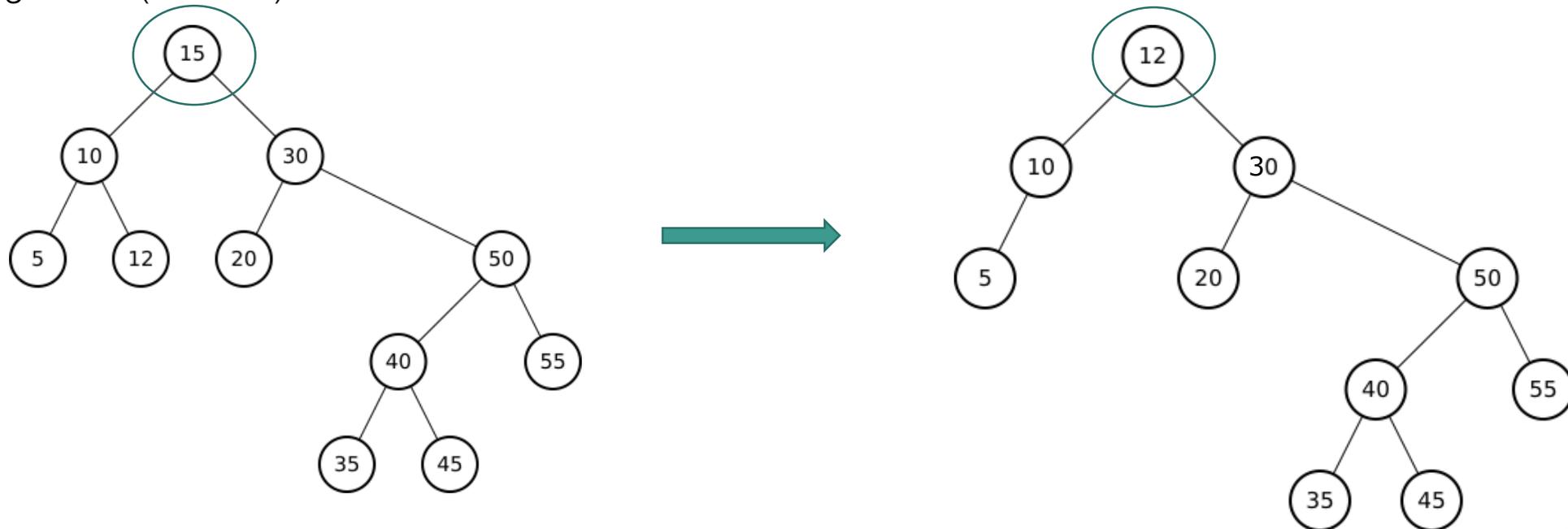
ABR : opération de suppression

- Il y a plusieurs cas à considérer:
 - Suppression d'une feuille
 - Le nœud à supprimer a un fils
 - **Le nœud à supprimer a deux fils**



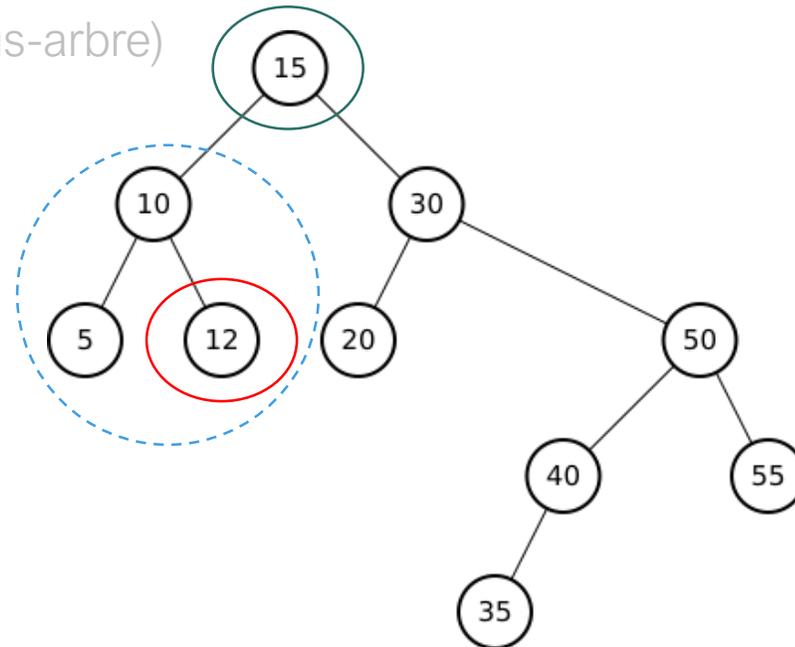
ABR : opération de suppression

- Il y a plusieurs cas à considérer:
 - Suppression d'une feuille
 - Le nœud à supprimer a un fils
 - **Le nœud à supprimer a deux fils**
- On remplace le nœud par son prédécesseur (ou son successeur) en valeur, dans son sous-arbre gauche (ou droit)



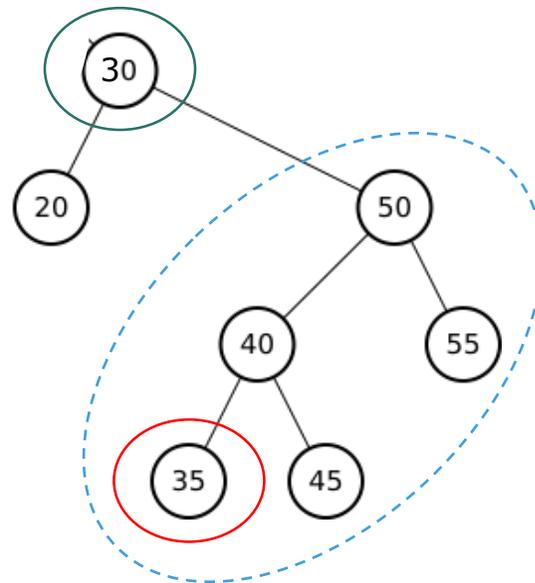
ABR : opération de suppression

- Il y a plusieurs cas à considérer:
 - Suppression d'une feuille
 - Le nœud à supprimer a un fils
 - **Le nœud à supprimer a deux fils**
- On remplace le nœud par son prédécesseur (ou son successeur) dans son sous-arbre gauche (ou droit)
 - **Version #1 : le prédécesseur dans le sous-arbre gauche est la feuille la plus à droite (plus grand élément du sous-arbre)**
 - Version #2 : le successeur dans le sous-arbre droit est la feuille la plus à gauche (plus petit élément du sous-arbre)



ABR : opération de suppression

- Il y a plusieurs cas à considérer:
 - Suppression d'une feuille
 - Le nœud à supprimer a un fils
 - **Le nœud à supprimer a deux fils**
- On remplace le nœud par son prédécesseur (ou son successeur) dans son sous-arbre gauche (ou droit)
 - Version #1 : le prédécesseur dans le sous-arbre gauche est la feuille la plus à droite (plus grand élément du sous-arbre)
 - **Version #2 : le successeur dans le sous-arbre droit est la feuille la plus à gauche (plus petit élément du sous-arbre)**



ABR : opération de suppression

- Il y a plusieurs cas à considérer:
 - Suppression d'une feuille
 - Le nœud à supprimer a un fils
 - **Le nœud à supprimer a deux fils**
- On remplace le nœud par son prédécesseur (ou son successeur) dans son sous-arbre gauche (ou droit)
 - **Version #1 : le prédécesseur dans le sous-arbre gauche est la feuille la plus à droite (plus grand élément du sous-arbre)**
 - Version #2 : le successeur dans le sous-arbre droit est la feuille la plus à gauche (plus petit élément du sous-arbre)
- Principe :
 - On recherche le nœud à supprimer
 - S'il n'a pas de fils gauche, on le remplace par son fils droit (fils unique)
 - Sinon on cherche la plus grande valeur du sous-arbre gauche (valeur la plus à droite) et on échange sa valeur avec la valeur du nœud à supprimer.

ABR : opération de suppression

- Algorithme :

```
FONCTION suppression(a: pointeur sur Arbre, e: element) : pointeur sur Arbre
VARIABLE
    tmp : ptr sur Arbre
DEBUT
    // element non présent dans l'arbre
    SI (a EST EGAL A NULL) Alors
        RETOURNER a
    // parcours récursif de l'arbre
    SINON SI (e SUP. STRICT. A element(a))
        fd(a) ← suppression(fd(a), e)
    SINON SI (e INF. STRICT. A element(a))
        fg(a) ← suppression(fg(a), e)
    ...
    FIN SI
    RETOURNER a
FIN
```

ABR : opération de suppression

- Algorithme :

FONCTION suppression(a: pointeur sur Arbre, e: element) : pointeur sur Arbre

VARIABLE

tmp : ptr sur Arbre

DEBUT

// element non présent dans l'arbre

SI (a EST EGAL A NULL) Alors

RETOURNER a

// parcours récursif de l'arbre

SINON SI (e SUP. STRICT. A element(a))

fd(a) ← suppression(fd(a), e)

SINON SI (e INF. STRICT. A element(a))

fg(a) ← suppression(fg(a), e)

// élément trouvé : remplacement par fils unique

SINON SI NON (existeFilsGauche(a))

tmp ← a

a ← fd(a)

libérer(tmp)

...

FIN SI

RETOURNER a

FIN

ABR : opération de suppression

- Algorithme :

FONCTION suppression(a: pointeur sur Arbre, e: element) : pointeur sur Arbre

VARIABLE

tmp : ptr sur Arbre

DEBUT

// element non présent dans l'arbre

SI (a EST EGAL A NULL) Alors

RETOURNER a

// parcours récursif de l'arbre

SINON SI (e SUP. STRICT. A element(a))

fd(a) ← suppression(fd(a), e)

SINON SI (e INF. STRICT. A element(a))

fg(a) ← suppression(fg(a), e)

// élément trouvé : remplacement par fils unique

SINON SI NON (existeFilsGauche(a))

tmp ← a

a ← fd(a)

libérer(tmp)

// élément trouvé : remplacement par prédécesseur

SINON

fg(a) ← supMax(fg(a), adresse(element(a)))

FIN SI

RETOURNER a

FIN

ABR : opération de suppression

- Algorithme :

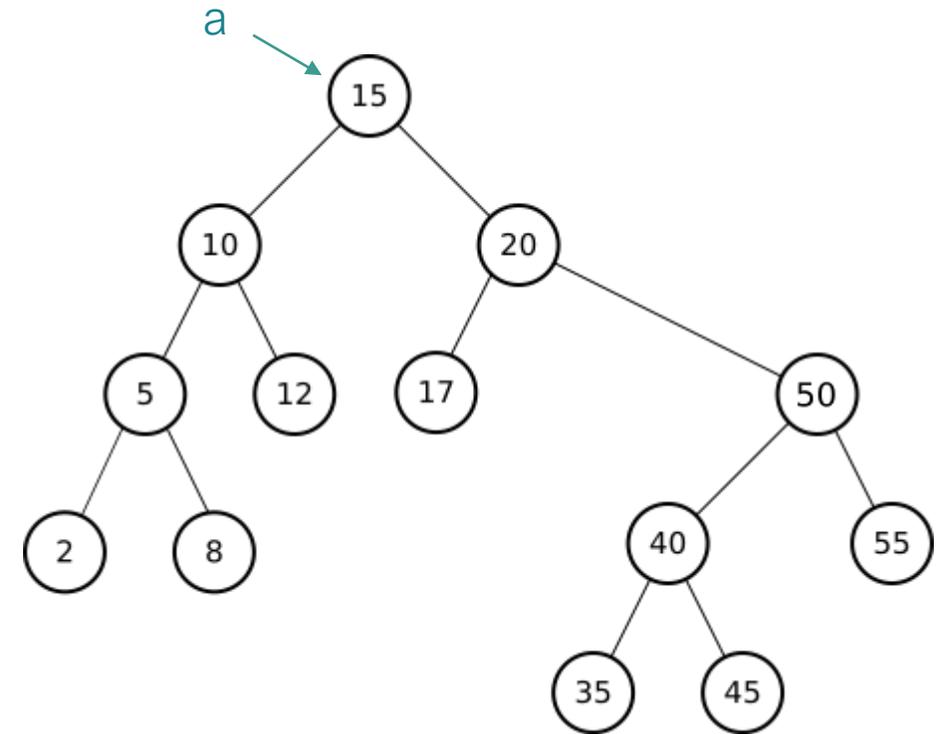
```
FONCTION suppression(a: pointeur sur Arbre, e: element) :  
pointeur sur Arbre  
VARIABLE  
    tmp : ptr sur Arbre  
DEBUT  
    // element non présent dans l'arbre  
    SI (a EST EGAL A NULL) Alors  
        RETOURNER a  
    // parcours récursif de l'arbre  
    SINON SI (e SUP. STRICT. A element(a))  
        fd(a) ← suppression(fd(a), e)  
    SINON SI (e INF. STRICT. A element(a))  
        fg(a) ← suppression(fg(a), e)  
    // élément trouvé : remplacement par fils unique  
    SINON SI NON (existeFilsGauche(a))  
        tmp ← a  
        a ← fd(a)  
        libérer(tmp)  
    // élément trouvé : remplacement par prédécesseur  
    SINON  
        fg(a) ← suppMax(fg(a), adresse(element(a)))  
    FIN SI  
    RETOURNER a
```

FIN

```
FONCTION suppMax(a: arbre, pe: Ptr sur Element) :  
ptr sur Arbre  
VARIABLE  
    tmp : ptr sur Arbre  
DEBUT  
    // on rappelle la fonction avec le fils droit  
    SI (existeFilsDroit(a)) Alors  
        fd(a) ← suppMax(fd(a), pe)  
    // Si plus de fils droit, on a le successeur  
    SINON  
        *pe ← element(a)  
        tmp ← a  
        a ← fg(a)  
        libérer(tmp)  
    FIN SI  
    RETOURNER a  
FIN
```

ABR : opération de suppression

- Algorithme : suppression (a, 8)



ABR : opération de suppression

- Algorithme : suppression (a, 8)

FONCTION suppression(a: pointeur sur Arbre, e: element) :
pointeur sur Arbre

VARIABLE

tmp : ptr sur Arbre

DEBUT

// element non présent dans l'arbre

SI (a EST EGAL A NULL) Alors

RETOURNER a

// parcours récursif de l'arbre

SINON SI (e SUP. STRICT. A element(a))

fd(a) ← suppression(fd(a), e)

SINON SI (e INF. STRICT. A element(a))

fg(a) ← suppression(fg(a), e)

// élément trouvé : remplacement par fils unique

SINON SI NON (existeFilsGauche(a))

tmp ← a

a ← fd(a)

libérer(tmp)

// élément trouvé : remplacement par prédécesseur

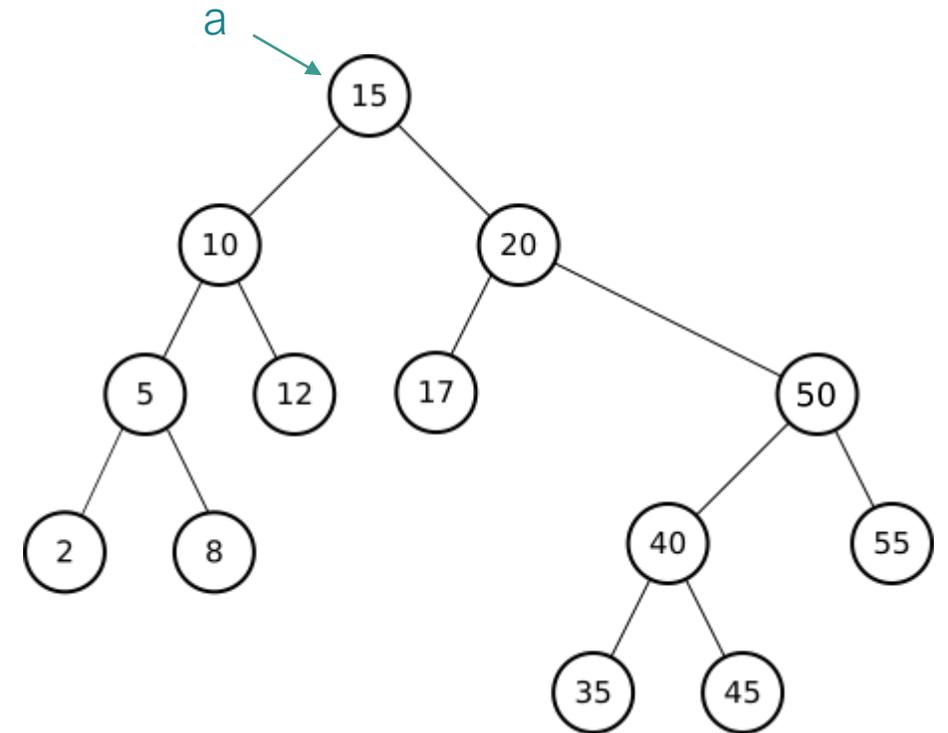
SINON

fg(a) ← suppMax(fg(a), adresse(element(a)))

FIN SI

RETOURNER a

FIN



ABR : opération de suppression

- Algorithme : suppression (a, 8)

FONCTION suppression(a: pointeur sur Arbre, e: element) :
pointeur sur Arbre

VARIABLE

tmp : ptr sur Arbre

DEBUT

// element non présent dans l'arbre

SI (a EST EGAL A NULL) Alors

RETOURNER a

// parcours récursif de l'arbre

SINON SI (e SUP. STRICT. A element(a))

fd(a) ← suppression(fd(a), e)

SINON SI (e INF. STRICT. A element(a))

fg(a) ← suppression(fg(a), e)

// élément trouvé : remplacement par fils unique

SINON SI NON (existeFilsGauche(a))

tmp ← a

a ← fd(a)

libérer(tmp)

// élément trouvé : remplacement par prédécesseur

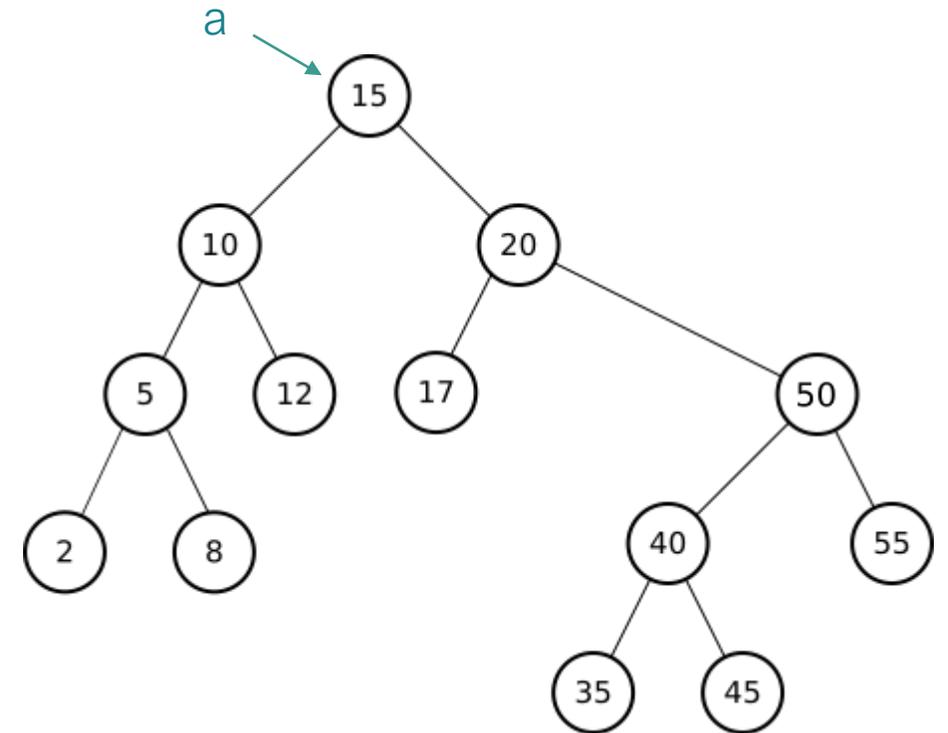
SINON

fg(a) ← suppMax(fg(a), adresse(element(a)))

FIN SI

RETOURNER a

FIN



ABR : opération de suppression

- Algorithme : suppression (a, 8)

FONCTION suppression(a: pointeur sur Arbre, e: element) :
pointeur sur Arbre

VARIABLE

tmp : ptr sur Arbre

DEBUT

// element non présent dans l'arbre

SI (a EST EGAL A NULL) Alors

RETOURNER a

// parcours récursif de l'arbre

→ SINON SI (e SUP. STRICT. A element(a))

fd(a) ← suppression(fd(a), e)

SINON SI (e INF. STRICT. A element(a))

fg(a) ← suppression(fg(a), e)

// élément trouvé : remplacement par fils unique

SINON SI NON (existeFilsGauche(a))

tmp ← a

a ← fd(a)

libérer(tmp)

// élément trouvé : remplacement par prédécesseur

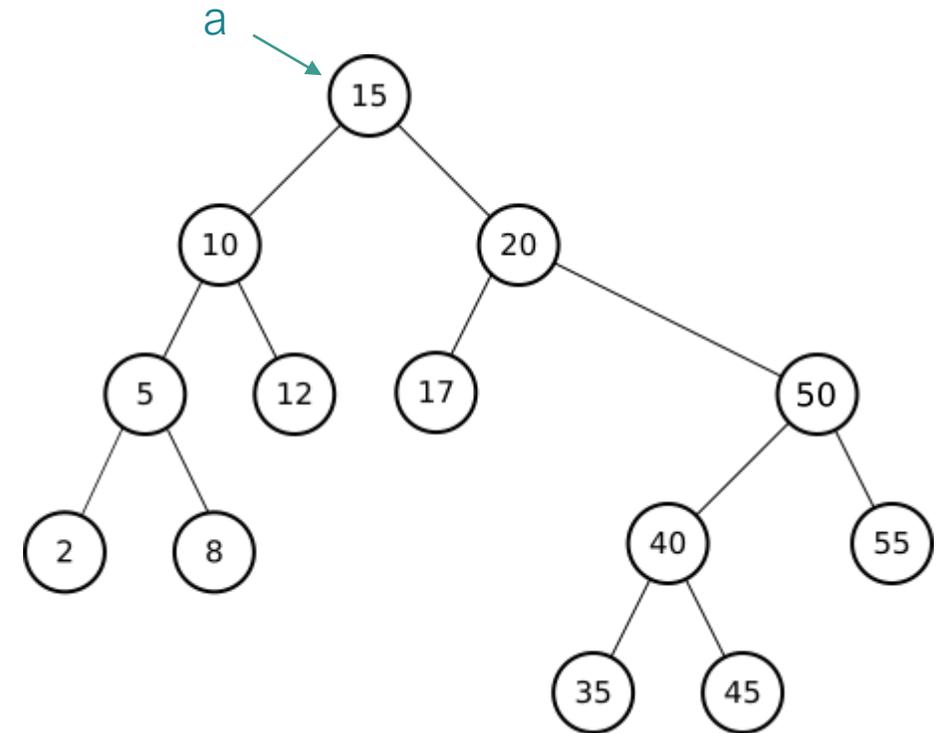
SINON

fg(a) ← suppMax(fg(a), adresse(element(a)))

FIN SI

RETOURNER a

FIN



ABR : opération de suppression

- Algorithme : suppression (a, 8)

FONCTION suppression(a: pointeur sur Arbre, e: element) :
pointeur sur Arbre

VARIABLE

tmp : ptr sur Arbre

DEBUT

// element non présent dans l'arbre

SI (a EST EGAL A NULL) Alors

RETOURNER a

// parcours récursif de l'arbre

SINON SI (e SUP. STRICT. A element(a))

fd(a) ← suppression(fd(a), e)

→ SINON SI (e INF. STRICT. A element(a))

fg(a) ← suppression(fg(a), e)

// élément trouvé : remplacement par fils unique

SINON SI NON (existeFilsGauche(a))

tmp ← a

a ← fd(a)

libérer(tmp)

// élément trouvé : remplacement par prédécesseur

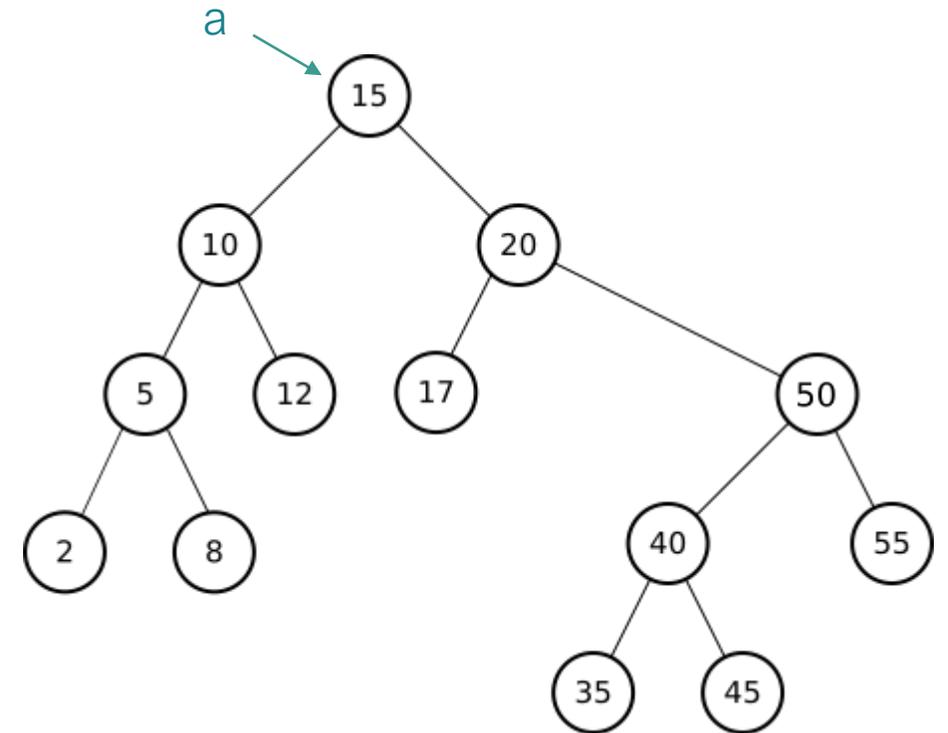
SINON

fg(a) ← suppMax(fg(a), adresse(element(a)))

FIN SI

RETOURNER a

FIN



ABR : opération de suppression

- Algorithme : suppression (a, 8)

FONCTION suppression(a: pointeur sur Arbre, e: element) :
pointeur sur Arbre

VARIABLE

tmp : ptr sur Arbre

DEBUT

// element non présent dans l'arbre

SI (a EST EGAL A NULL) Alors

RETOURNER a

// parcours récursif de l'arbre

SINON SI (e SUP. STRICT. A element(a))

fd(a) ← suppression(fd(a), e)

SINON SI (e INF. STRICT. A element(a))

➡ fg(a) ← suppression(fg(a), e)

// élément trouvé : remplacement par fils unique

SINON SI NON (existeFilsGauche(a))

tmp ← a

a ← fd(a)

libérer(tmp)

// élément trouvé : remplacement par prédécesseur

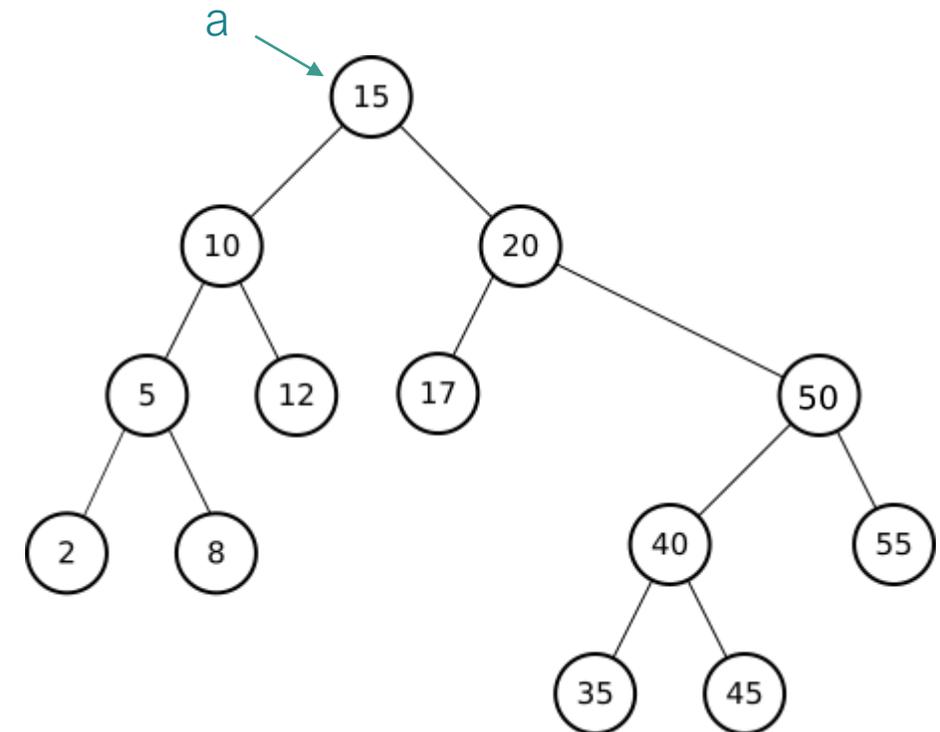
SINON

fg(a) ← suppMax(fg(a), adresse(element(a)))

FIN SI

RETOURNER a

FIN



ABR : opération de suppression

- Algorithme : suppression (a, 8)

FONCTION suppression(a: pointeur sur Arbre, e: element) :
pointeur sur Arbre

VARIABLE

tmp : ptr sur Arbre

→ DEBUT

// element non présent dans l'arbre

SI (a EST EGAL A NULL) Alors

RETOURNER a

// parcours récursif de l'arbre

SINON SI (e SUP. STRICT. A element(a))

fd(a) ← suppression(fd(a), e)

SINON SI (e INF. STRICT. A element(a))

→ fg(a) ← suppression(fg(a), e)

// élément trouvé : remplacement par fils unique

SINON SI NON (existeFilsGauche(a))

tmp ← a

a ← fd(a)

libérer(tmp)

// élément trouvé : remplacement par prédécesseur

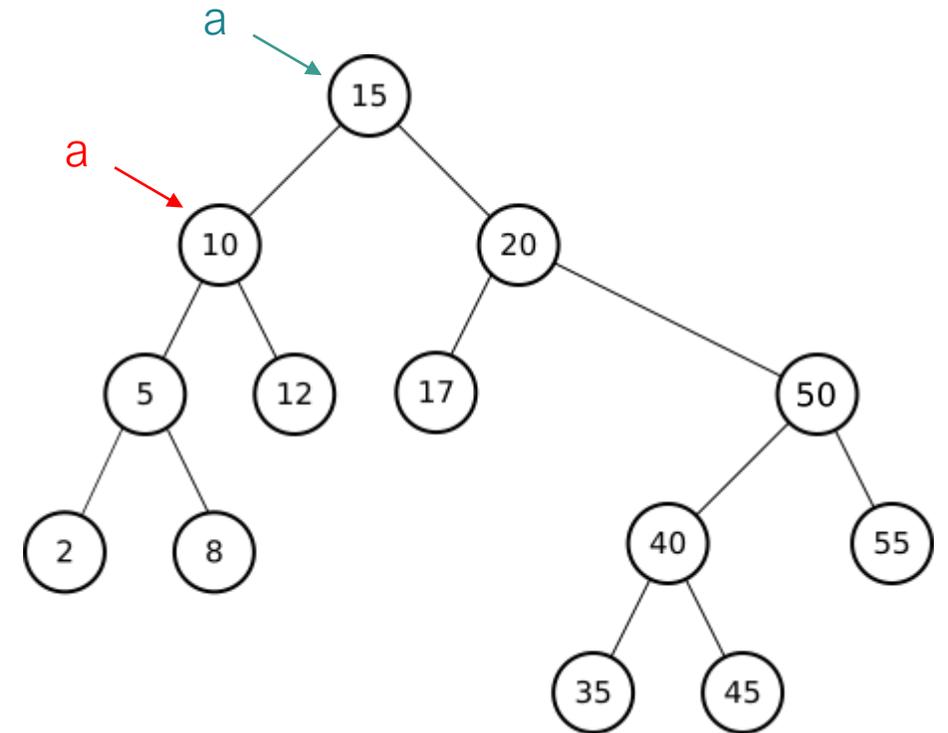
SINON

fg(a) ← suppMax(fg(a), adresse(element(a)))

FIN SI

RETOURNER a

FIN



ABR : opération de suppression

- Algorithme : suppression (a, 8)

FONCTION suppression(a: pointeur sur Arbre, e: element) :
pointeur sur Arbre

VARIABLE

tmp : ptr sur Arbre

DEBUT

// element non présent dans l'arbre

➔ SI (a EST EGAL A NULL) Alors
RETOURNER a

// parcours récursif de l'arbre

SINON SI (e SUP. STRICT. A element(a))

fd(a) ← suppression(fd(a), e)

SINON SI (e INF. STRICT. A element(a))

➔ fg(a) ← suppression(fg(a), e)

// élément trouvé : remplacement par fils unique

SINON SI NON (existeFilsGauche(a))

tmp ← a

a ← fd(a)

libérer(tmp)

// élément trouvé : remplacement par prédécesseur

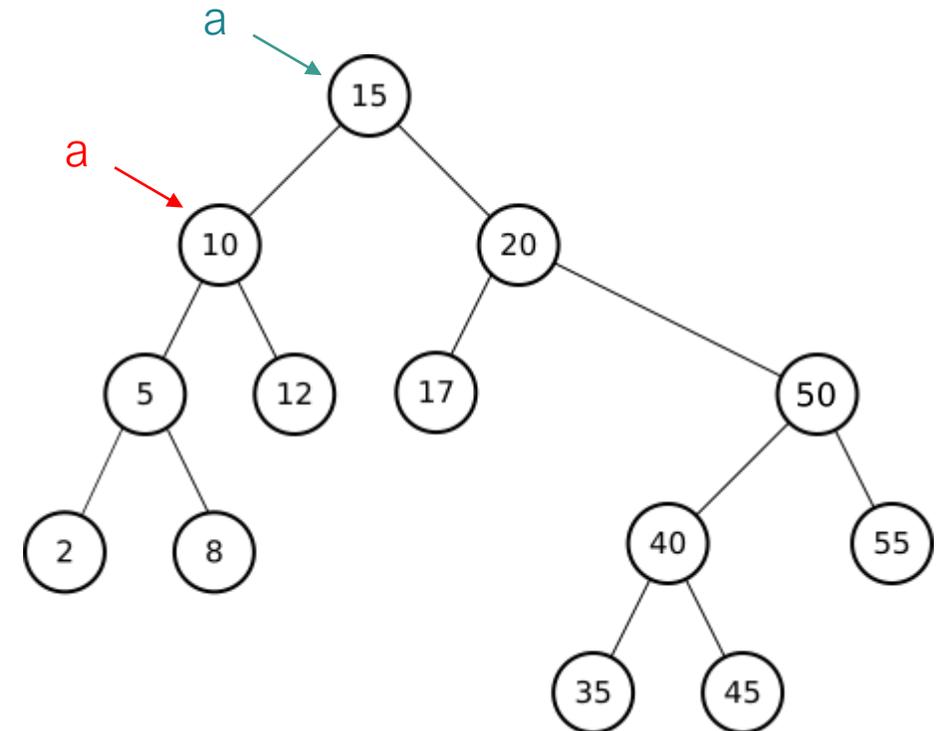
SINON

fg(a) ← suppMax(fg(a), adresse(element(a)))

FIN SI

RETOURNER a

FIN



ABR : opération de suppression

- Algorithme : suppression (a, 8)

FONCTION suppression(a: pointeur sur Arbre, e: element) :
pointeur sur Arbre

VARIABLE

tmp : ptr sur Arbre

DEBUT

// element non présent dans l'arbre

SI (a EST EGAL A NULL) Alors

RETOURNER a

// parcours récursif de l'arbre

→ SINON SI (e SUP. STRICT. A element(a))

fd(a) ← suppression(fd(a), e)

SINON SI (e INF. STRICT. A element(a))

→ fg(a) ← suppression(fg(a), e)

// élément trouvé : remplacement par fils unique

SINON SI NON (existeFilsGauche(a))

tmp ← a

a ← fd(a)

libérer(tmp)

// élément trouvé : remplacement par prédécesseur

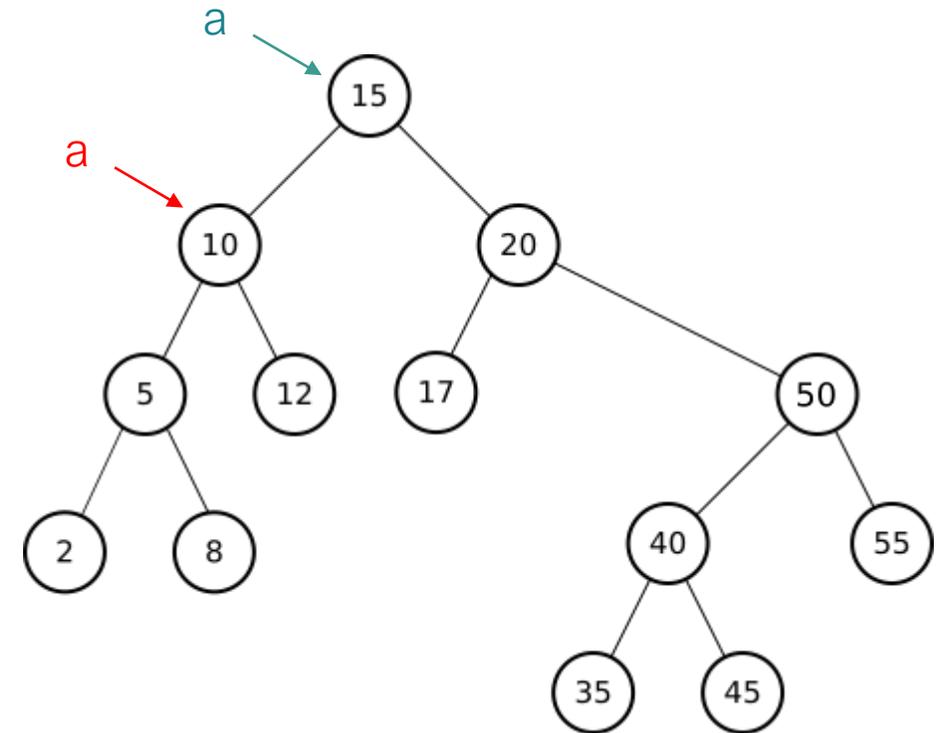
SINON

fg(a) ← suppMax(fg(a), adresse(element(a)))

FIN SI

RETOURNER a

FIN



ABR : opération de suppression

- Algorithme : suppression (a, 8)

FONCTION suppression(a: pointeur sur Arbre, e: element) :
pointeur sur Arbre

VARIABLE

tmp : ptr sur Arbre

DEBUT

// element non présent dans l'arbre

SI (a EST EGAL A NULL) Alors

RETOURNER a

// parcours récursif de l'arbre

SINON SI (e SUP. STRICT. A element(a))

fd(a) ← suppression(fd(a), e)

→ SINON SI (e INF. STRICT. A element(a))

→ fg(a) ← suppression(fg(a), e)

// élément trouvé : remplacement par fils unique

SINON SI NON (existeFilsGauche(a))

tmp ← a

a ← fd(a)

libérer(tmp)

// élément trouvé : remplacement par prédécesseur

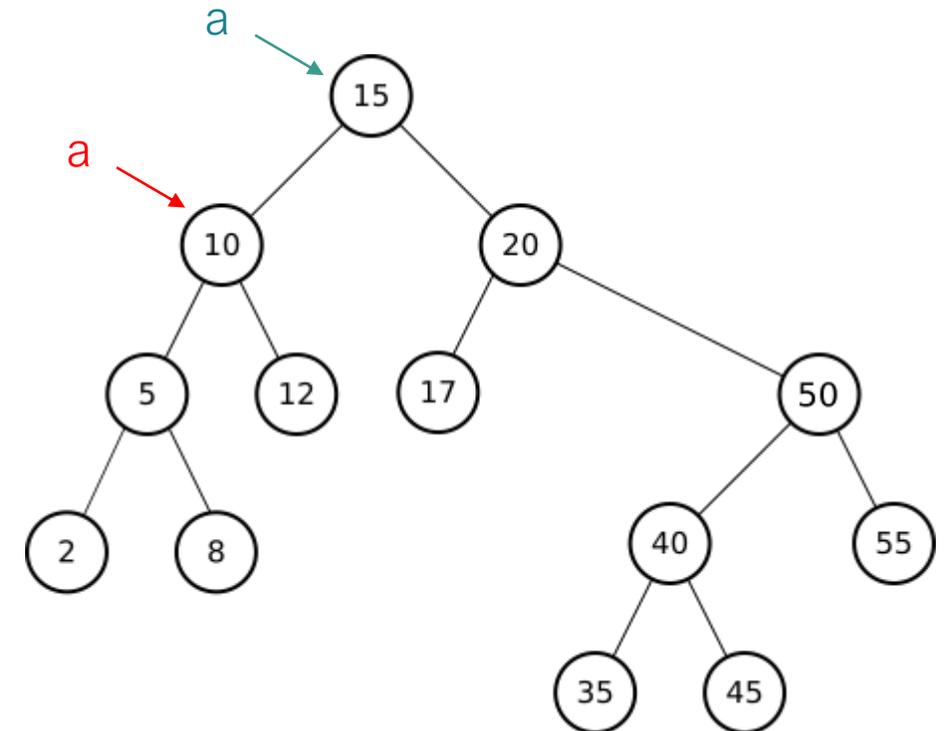
SINON

fg(a) ← suppMax(fg(a), adresse(element(a)))

FIN SI

RETOURNER a

FIN



ABR : opération de suppression

- Algorithme : suppression (a, 8)

FONCTION suppression(a: pointeur sur Arbre, e: element) :
pointeur sur Arbre

VARIABLE

tmp : ptr sur Arbre

DEBUT

// element non présent dans l'arbre

SI (a EST EGAL A NULL) Alors

RETOURNER a

// parcours récursif de l'arbre

SINON SI (e SUP. STRICT. A element(a))

fd(a) ← suppression(fd(a), e)

SINON SI (e INF. STRICT. A element(a))

➡ ➡ fg(a) ← suppression(fg(a), e)

// élément trouvé : remplacement par fils unique

SINON SI NON (existeFilsGauche(a))

tmp ← a

a ← fd(a)

libérer(tmp)

// élément trouvé : remplacement par prédécesseur

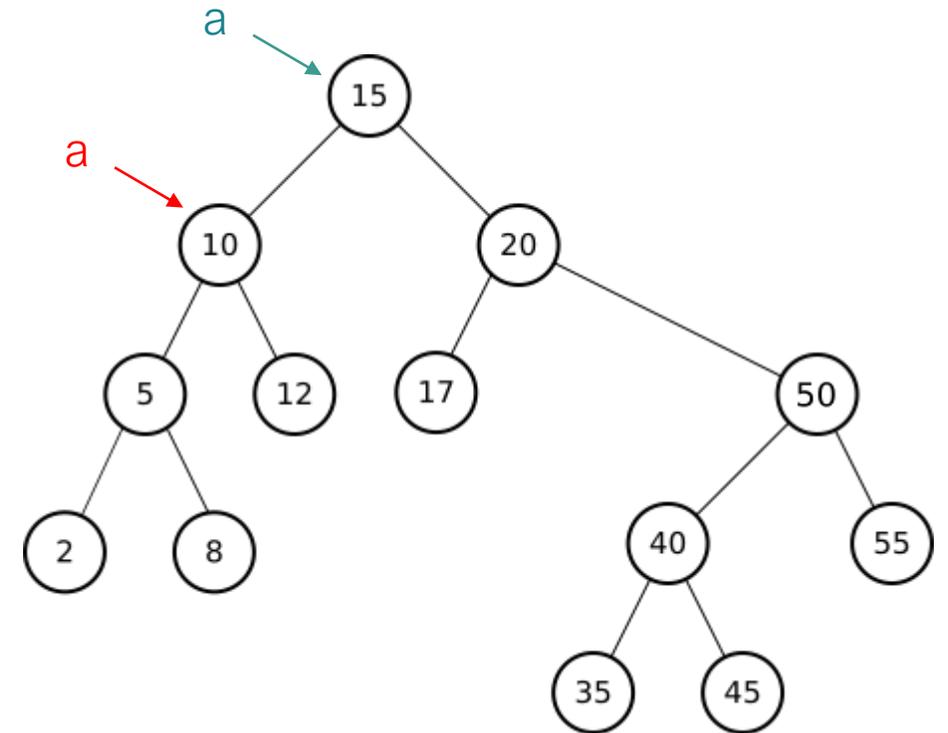
SINON

fg(a) ← suppMax(fg(a), adresse(element(a)))

FIN SI

RETOURNER a

FIN



ABR : opération de suppression

- Algorithme : suppression (a, 8)

FONCTION suppression(a: pointeur sur Arbre, e: element) :
pointeur sur Arbre

VARIABLE

tmp : ptr sur Arbre

DEBUT

// element non présent dans l'arbre

SI (a EST EGAL A NULL) Alors

RETOURNER a

// parcours récursif de l'arbre

SINON SI (e SUP. STRICT. A element(a))

fd(a) ← suppression(fd(a), e)

SINON SI (e INF. STRICT. A element(a))

➡ ➡ fg(a) ← suppression(fg(a), e)

// élément trouvé : remplacement par fils unique

SINON SI NON (existeFilsGauche(a))

tmp ← a

a ← fd(a)

libérer(tmp)

// élément trouvé : remplacement par prédécesseur

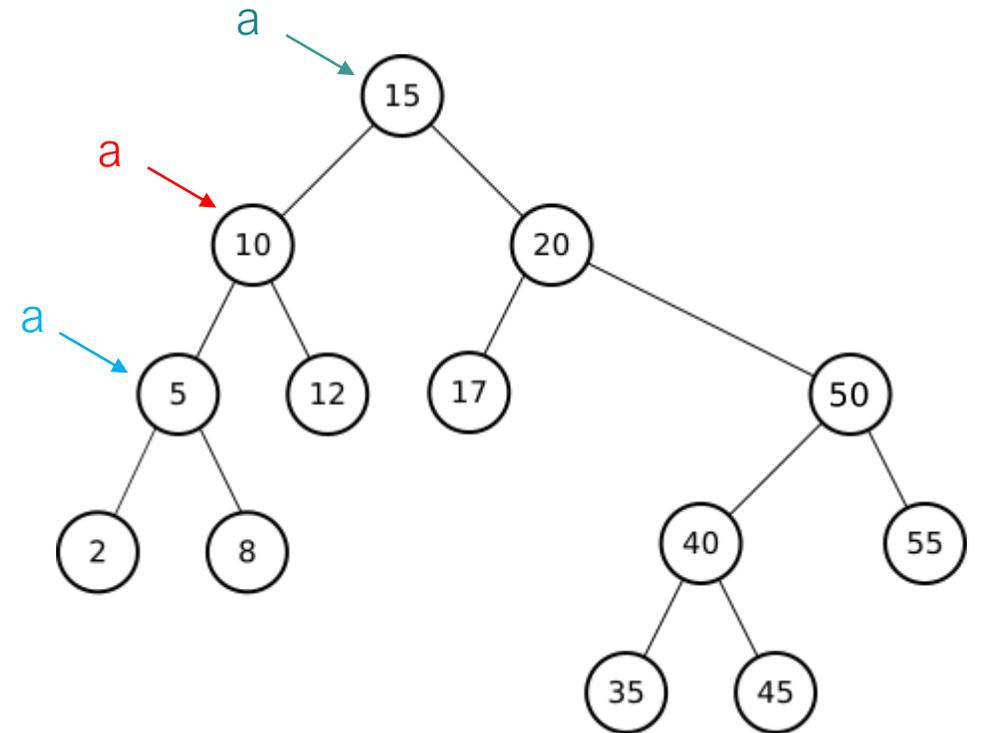
SINON

fg(a) ← suppMax(fg(a), adresse(element(a)))

FIN SI

RETOURNER a

FIN



ABR : opération de suppression

- Algorithme : suppression (a, 8)

FONCTION suppression(a: pointeur sur Arbre, e: element) :
pointeur sur Arbre

VARIABLE

tmp : ptr sur Arbre

DEBUT

// element non présent dans l'arbre

➡ SI (a EST EGAL A NULL) Alors
RETOURNER a

// parcours récursif de l'arbre

SINON SI (e SUP. STRICT. A element(a))

fd(a) ← suppression(fd(a), e)

SINON SI (e INF. STRICT. A element(a))

➡ ➡ fg(a) ← suppression(fg(a), e)

// élément trouvé : remplacement par fils unique

SINON SI NON (existeFilsGauche(a))

tmp ← a

a ← fd(a)

libérer(tmp)

// élément trouvé : remplacement par prédécesseur

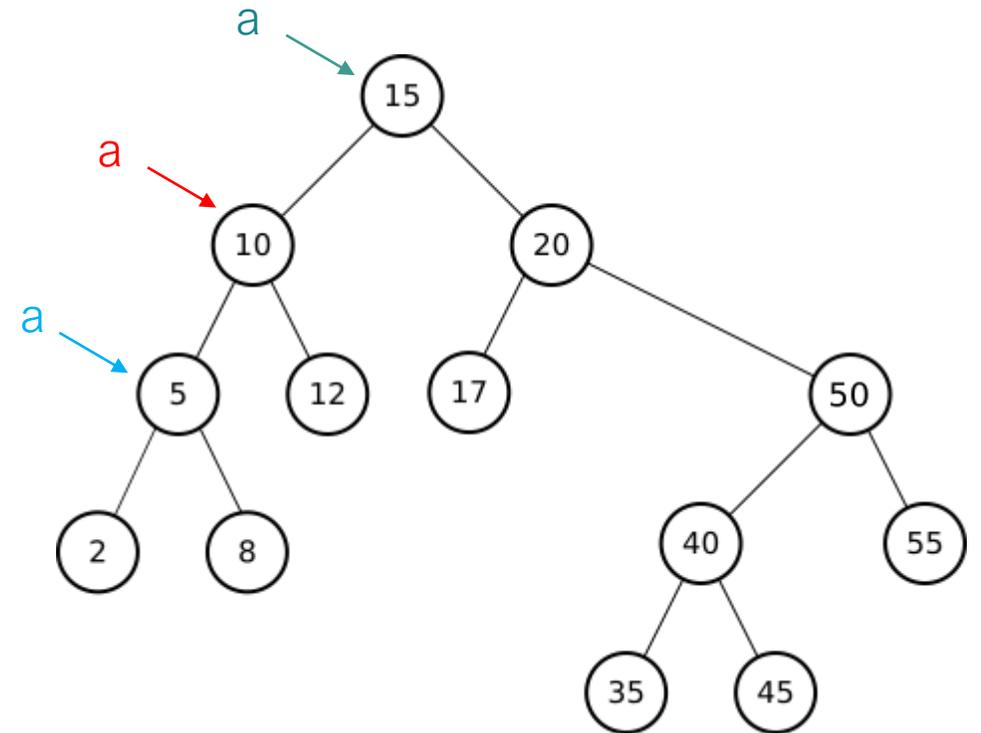
SINON

fg(a) ← suppMax(fg(a), adresse(element(a)))

FIN SI

RETOURNER a

FIN



ABR : opération de suppression

- Algorithme : suppression (a, 8)

FONCTION suppression(a: pointeur sur Arbre, e: element) :
pointeur sur Arbre

VARIABLE

tmp : ptr sur Arbre

DEBUT

// element non présent dans l'arbre

SI (a EST EGAL A NULL) Alors

RETOURNER a

// parcours récursif de l'arbre

➡ SINON SI (e SUP. STRICT. A element(a))

fd(a) ← suppression(fd(a), e)

SINON SI (e INF. STRICT. A element(a))

➡ ➡ fg(a) ← suppression(fg(a), e)

// élément trouvé : remplacement par fils unique

SINON SI NON (existeFilsGauche(a))

tmp ← a

a ← fd(a)

libérer(tmp)

// élément trouvé : remplacement par prédécesseur

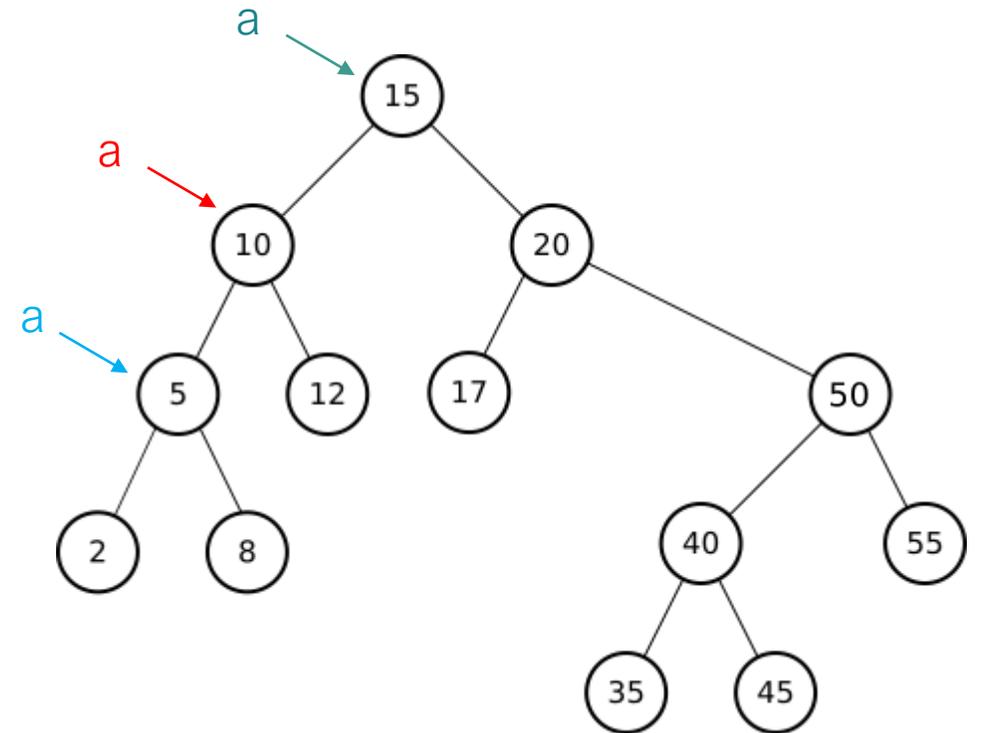
SINON

fg(a) ← suppMax(fg(a), adresse(element(a)))

FIN SI

RETOURNER a

FIN



ABR : opération de suppression

- Algorithme : suppression (a, 8)

FONCTION suppression(a: pointeur sur Arbre, e: element) :
pointeur sur Arbre

VARIABLE

tmp : ptr sur Arbre

DEBUT

// element non présent dans l'arbre

SI (a EST EGAL A NULL) Alors

RETOURNER a

// parcours récursif de l'arbre

SINON SI (e SUP. STRICT. A element(a))

➡ $fd(a) \leftarrow suppression(fd(a), e)$

SINON SI (e INF. STRICT. A element(a))

➡ ➡ $fg(a) \leftarrow suppression(fg(a), e)$

// élément trouvé : remplacement par fils unique

SINON SI NON (existeFilsGauche(a))

tmp \leftarrow a

a \leftarrow fd(a)

libérer(tmp)

// élément trouvé : remplacement par prédécesseur

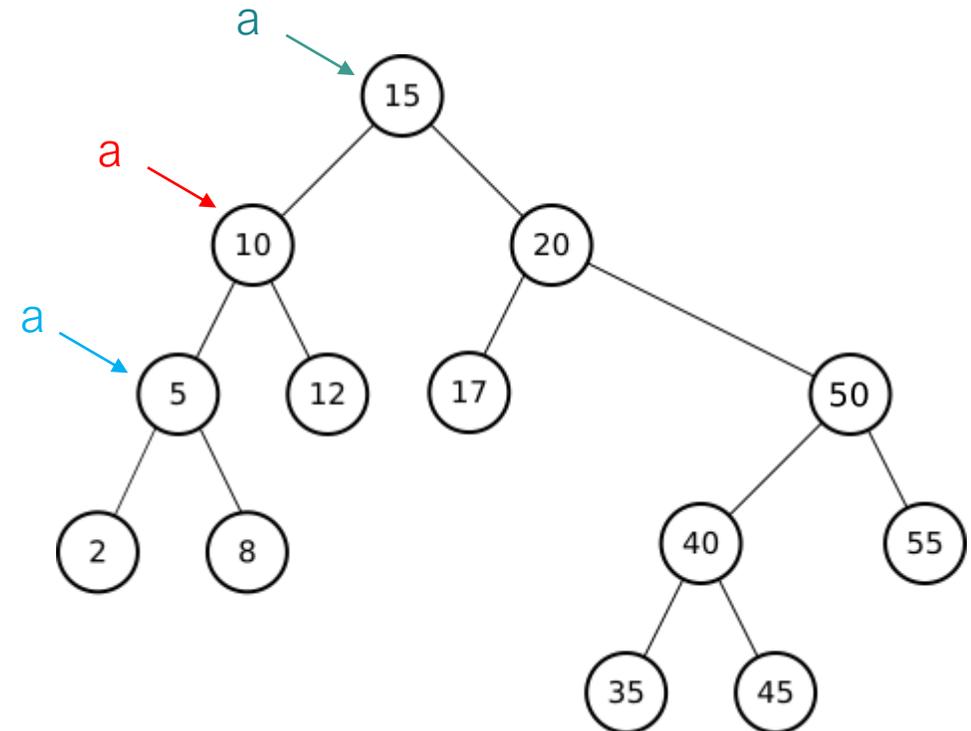
SINON

$fg(a) \leftarrow suppMax(fg(a), adresse(element(a)))$

FIN SI

RETOURNER a

FIN



ABR : opération de suppression

- Algorithme : suppression (a, 8)

FONCTION suppression(a: pointeur sur Arbre, e: element) :
pointeur sur Arbre

VARIABLE

tmp : ptr sur Arbre

DEBUT

// element non présent dans l'arbre

SI (a EST EGAL A NULL) Alors

RETOURNER a

// parcours récursif de l'arbre

SINON SI (e SUP. STRICT. A element(a))

➡ fd(a) ← suppression(fd(a), e)

SINON SI (e INF. STRICT. A element(a))

➡ fg(a) ← suppression(fg(a), e)

// élément trouvé : remplacement par fils unique

SINON SI NON (existeFilsGauche(a))

tmp ← a

a ← fd(a)

libérer(tmp)

// élément trouvé : remplacement par prédécesseur

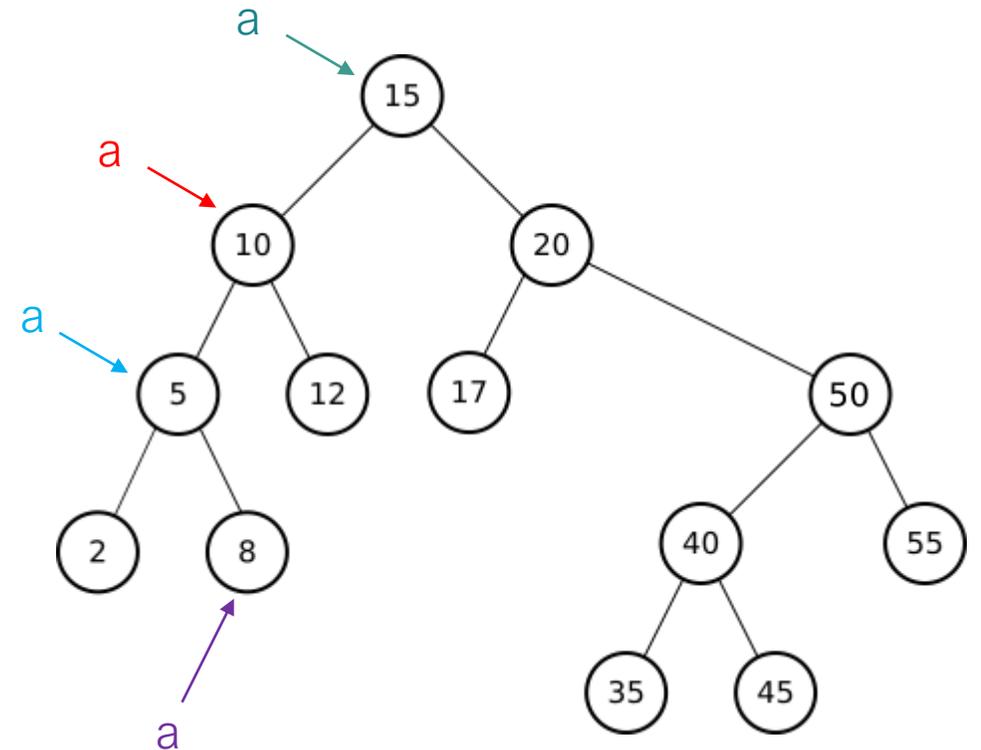
SINON

fg(a) ← suppMax(fg(a), adresse(element(a)))

FIN SI

RETOURNER a

FIN



ABR : opération de suppression

- Algorithme : suppression (a, 8)

FONCTION suppression(a: pointeur sur Arbre, e: element) :
pointeur sur Arbre

VARIABLE

tmp : ptr sur Arbre

DEBUT

// element non présent dans l'arbre

➡ SI (a EST EGAL A NULL) Alors
RETOURNER a

// parcours récursif de l'arbre

SINON SI (e SUP. STRICT. A element(a))

➡ fd(a) ← suppression(fd(a), e)

SINON SI (e INF. STRICT. A element(a))

➡ fg(a) ← suppression(fg(a), e)

// élément trouvé : remplacement par fils unique

SINON SI NON (existeFilsGauche(a))

tmp ← a

a ← fd(a)

libérer(tmp)

// élément trouvé : remplacement par prédécesseur

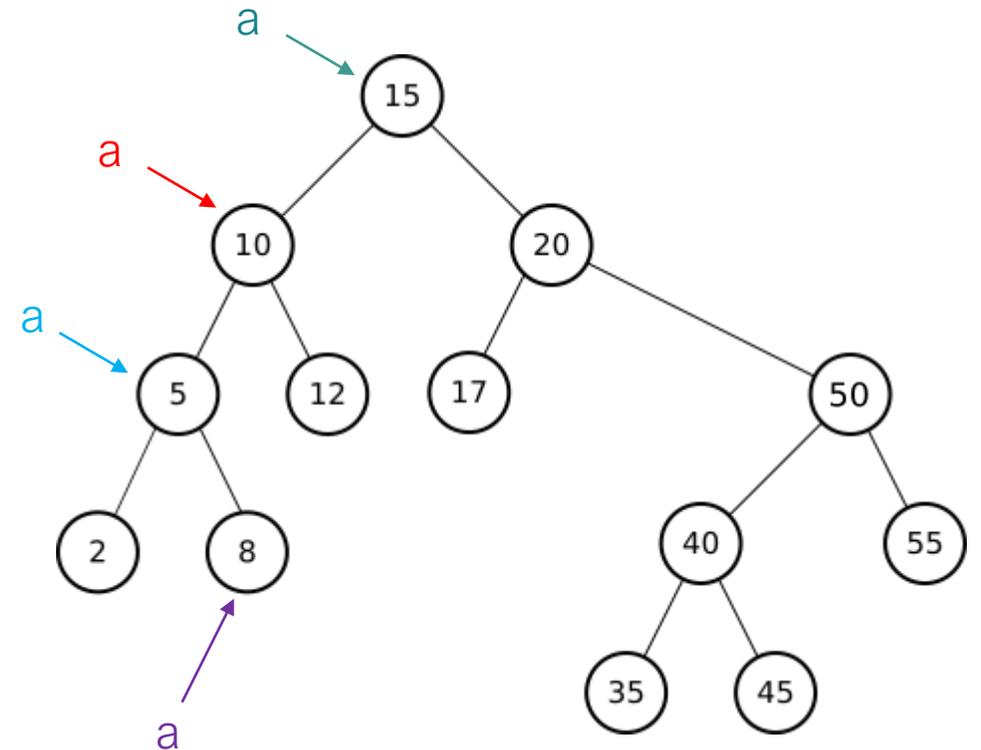
SINON

fg(a) ← suppMax(fg(a), adresse(element(a)))

FIN SI

RETOURNER a

FIN



ABR : opération de suppression

- Algorithme : suppression (a, 8)

FONCTION suppression(a: pointeur sur Arbre, e: element) :
pointeur sur Arbre

VARIABLE

tmp : ptr sur Arbre

DEBUT

// element non présent dans l'arbre

SI (a EST EGAL A NULL) Alors

RETOURNER a

// parcours récursif de l'arbre

➡ SINON SI (e SUP. STRICT. A element(a))

➡ fd(a) ← suppression(fd(a), e)

SINON SI (e INF. STRICT. A element(a))

➡ fg(a) ← suppression(fg(a), e)

// élément trouvé : remplacement par fils unique

SINON SI NON (existeFilsGauche(a))

tmp ← a

a ← fd(a)

libérer(tmp)

// élément trouvé : remplacement par prédécesseur

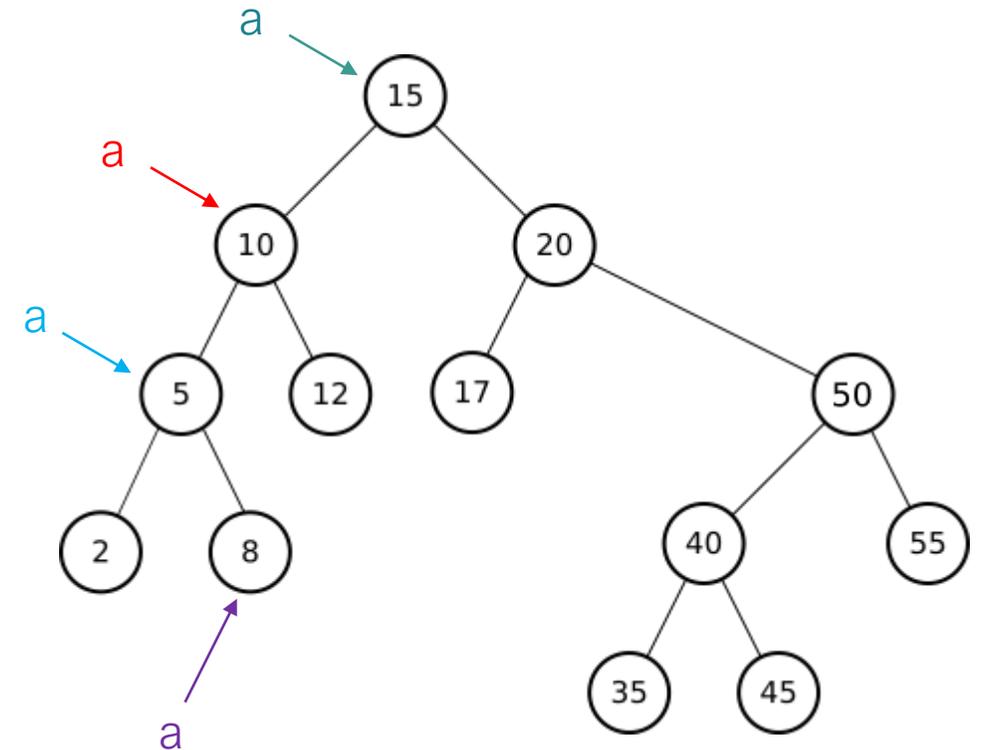
SINON

fg(a) ← suppMax(fg(a), adresse(element(a)))

FIN SI

RETOURNER a

FIN



ABR : opération de suppression

- Algorithme : suppression (a, 8)

FONCTION suppression(a: pointeur sur Arbre, e: element) :
pointeur sur Arbre

VARIABLE

tmp : ptr sur Arbre

DEBUT

// element non présent dans l'arbre

SI (a EST EGAL A NULL) Alors

RETOURNER a

// parcours récursif de l'arbre

SINON SI (e SUP. STRICT. A element(a))

➡ fd(a) ← suppression(fd(a), e)

➡ SINON SI (e INF. STRICT. A element(a))

➡ fg(a) ← suppression(fg(a), e)

// élément trouvé : remplacement par fils unique

SINON SI NON (existeFilsGauche(a))

tmp ← a

a ← fd(a)

libérer(tmp)

// élément trouvé : remplacement par prédécesseur

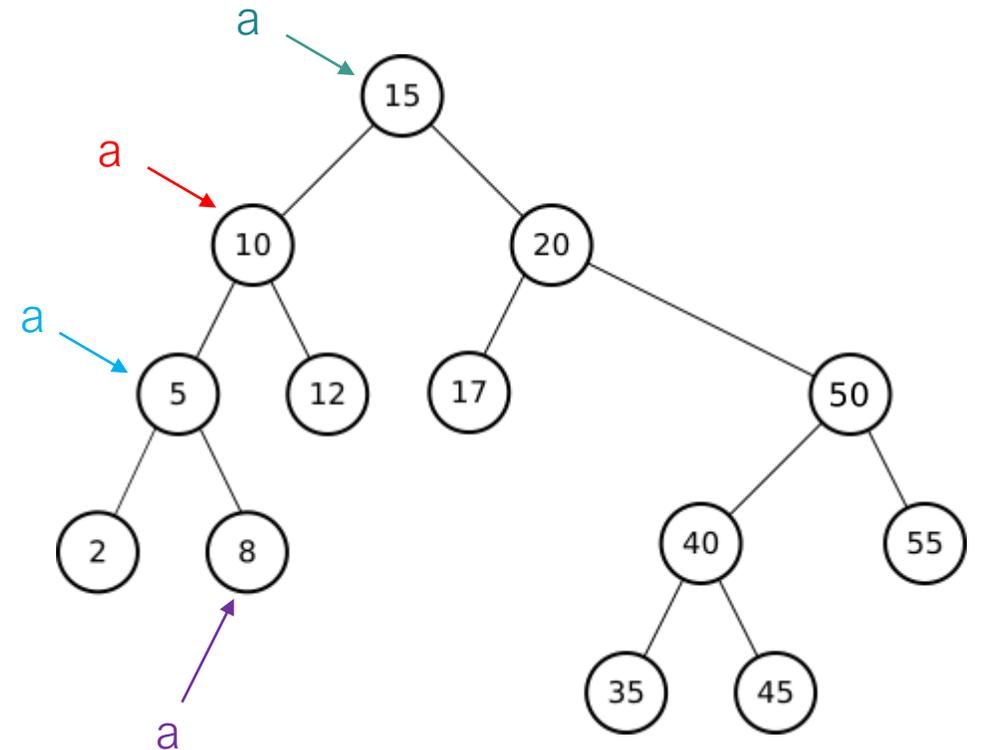
SINON

fg(a) ← supMax(fg(a), adresse(element(a)))

FIN SI

RETOURNER a

FIN



ABR : opération de suppression

- Algorithme : suppression (a, 8)

FONCTION suppression(a: pointeur sur Arbre, e: element) :
pointeur sur Arbre

VARIABLE

tmp : ptr sur Arbre

DEBUT

// element non présent dans l'arbre

SI (a EST EGAL A NULL) Alors

RETOURNER a

// parcours récursif de l'arbre

SINON SI (e SUP. STRICT. A element(a))

➡ fd(a) ← suppression(fd(a), e)

SINON SI (e INF. STRICT. A element(a))

➡ fg(a) ← suppression(fg(a), e)

// élément trouvé : remplacement par fils unique

➡ SINON SI NON (existeFilsGauche(a))

tmp ← a

a ← fd(a)

libérer(tmp)

// élément trouvé : remplacement par prédécesseur

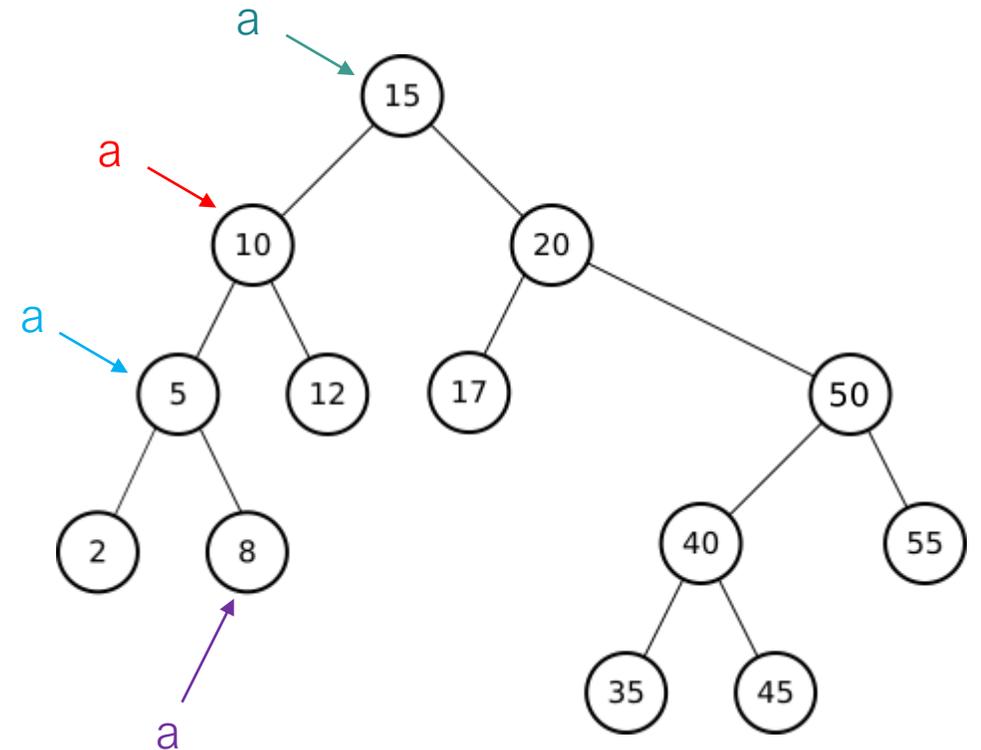
SINON

fg(a) ← suppMax(fg(a), adresse(element(a)))

FIN SI

RETOURNER a

FIN



ABR : opération de suppression

- Algorithme : suppression (a, 8)

FONCTION suppression(a: pointeur sur Arbre, e: element) :
pointeur sur Arbre

VARIABLE

tmp : ptr sur Arbre

DEBUT

// element non présent dans l'arbre

SI (a EST EGAL A NULL) Alors

RETOURNER a

// parcours récursif de l'arbre

SINON SI (e SUP. STRICT. A element(a))

➡ fd(a) ← suppression(fd(a), e)

SINON SI (e INF. STRICT. A element(a))

➡ fg(a) ← suppression(fg(a), e)

// élément trouvé : remplacement par fils unique

SINON SI NON (existeFilsGauche(a))

➡ tmp ← a

a ← fd(a)

libérer(tmp)

// élément trouvé : remplacement par prédécesseur

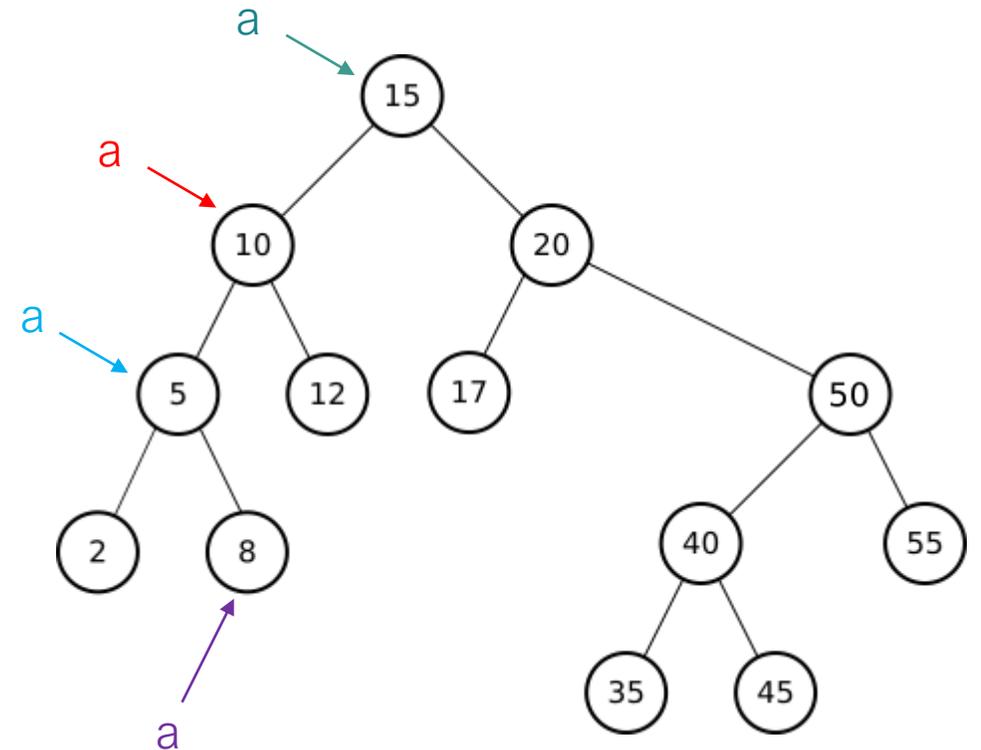
SINON

fg(a) ← suppMax(fg(a), adresse(element(a)))

FIN SI

RETOURNER a

FIN



ABR : opération de suppression

- Algorithme : suppression (a, 8)

FONCTION suppression(a: pointeur sur Arbre, e: element) :
pointeur sur Arbre

VARIABLE

tmp : ptr sur Arbre

DEBUT

// element non présent dans l'arbre

SI (a EST EGAL A NULL) Alors

RETOURNER a

// parcours récursif de l'arbre

SINON SI (e SUP. STRICT. A element(a))

➡ fd(a) ← suppression(fd(a), e)

SINON SI (e INF. STRICT. A element(a))

➡ fg(a) ← suppression(fg(a), e)

// élément trouvé : remplacement par fils unique

SINON SI NON (existeFilsGauche(a))

tmp ← a

➡ a ← fd(a)

libérer(tmp)

// élément trouvé : remplacement par prédécesseur

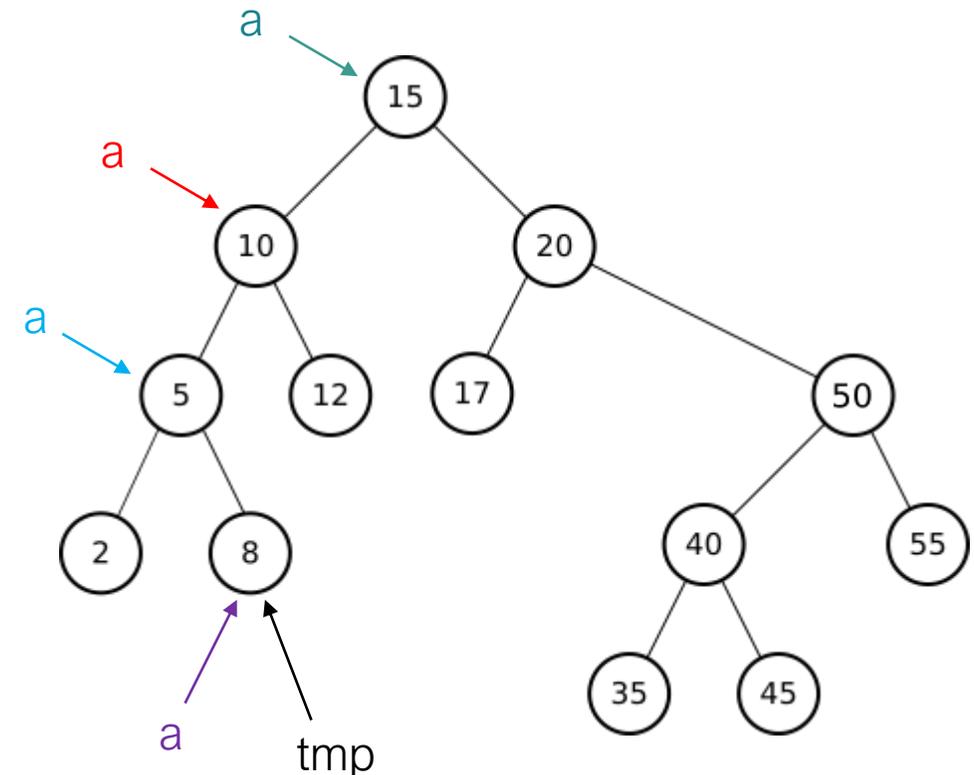
SINON

fg(a) ← suppMax(fg(a), adresse(element(a)))

FIN SI

RETOURNER a

FIN



ABR : opération de suppression

- Algorithme : suppression (a, 8)

FONCTION suppression(a: pointeur sur Arbre, e: element) :
pointeur sur Arbre

VARIABLE

tmp : ptr sur Arbre

DEBUT

// element non présent dans l'arbre

SI (a EST EGAL A NULL) Alors

RETOURNER a

// parcours récursif de l'arbre

SINON SI (e SUP. STRICT. A element(a))

➡ fd(a) ← suppression(fd(a), e)

SINON SI (e INF. STRICT. A element(a))

➡ fg(a) ← suppression(fg(a), e)

// élément trouvé : remplacement par fils unique

SINON SI NON (existeFilsGauche(a))

tmp ← a

a ← fd(a)

➡ libérer(tmp)

// élément trouvé : remplacement par prédécesseur

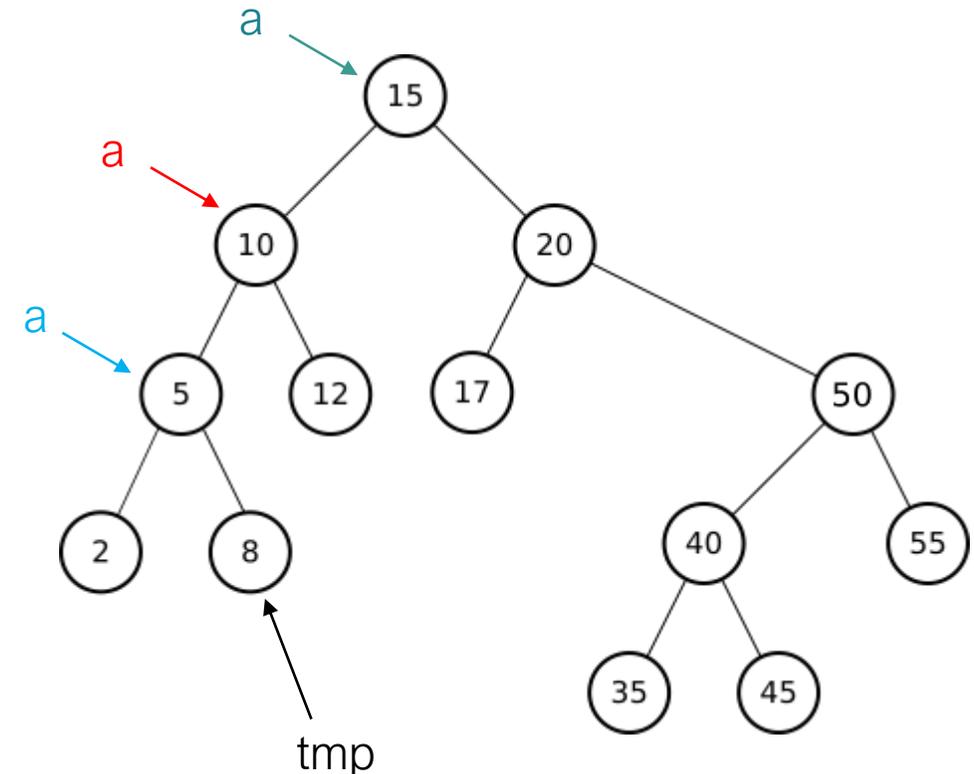
SINON

fg(a) ← suppMax(fg(a), adresse(element(a)))

FIN SI

RETOURNER a

FIN



ABR : opération de suppression

- Algorithme : suppression (a, 8)

FONCTION suppression(a: pointeur sur Arbre, e: element) :
pointeur sur Arbre

VARIABLE

tmp : ptr sur Arbre

DEBUT

// element non présent dans l'arbre

SI (a EST EGAL A NULL) Alors

RETOURNER a

// parcours récursif de l'arbre

SINON SI (e SUP. STRICT. A element(a))

➡ fd(a) ← suppression(fd(a), e)

SINON SI (e INF. STRICT. A element(a))

➡ fg(a) ← suppression(fg(a), e)

// élément trouvé : remplacement par fils unique

SINON SI NON (existeFilsGauche(a))

tmp ← a

a ← fd(a)

libérer(tmp)

// élément trouvé : remplacement par prédécesseur

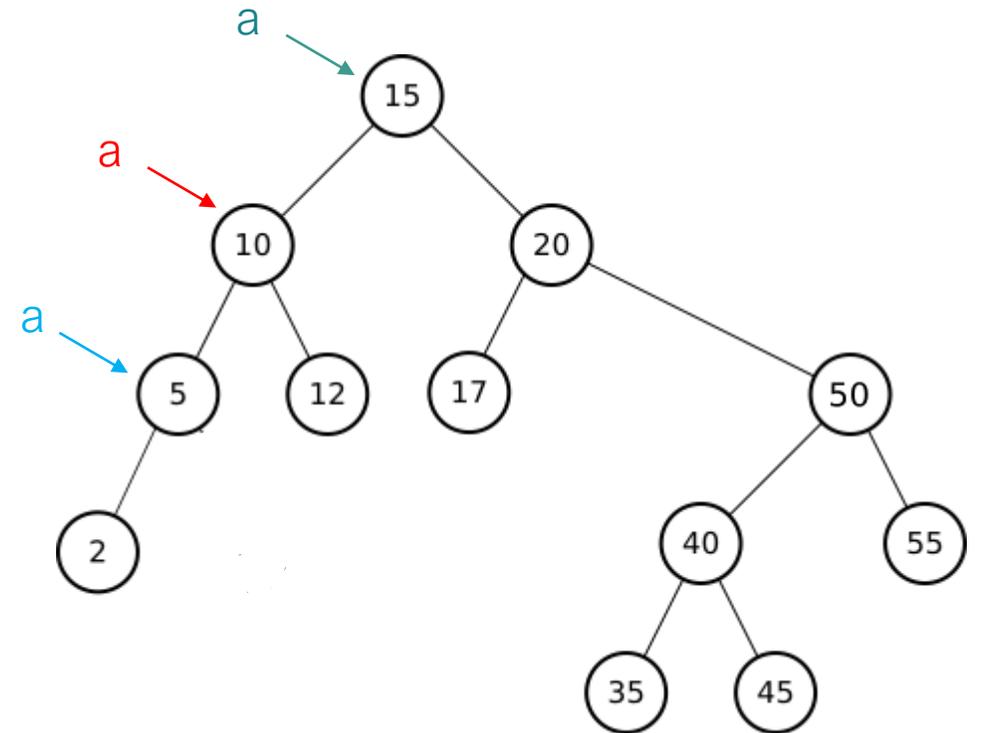
SINON

fg(a) ← suppMax(fg(a), adresse(element(a)))

FIN SI

➡ RETOURNER a

FIN



a ➡ NULL

ABR : opération de suppression

- Algorithme : suppression (a, 8)

FONCTION suppression(a: pointeur sur Arbre, e: element) :
pointeur sur Arbre

VARIABLE

tmp : ptr sur Arbre

DEBUT

// element non présent dans l'arbre

SI (a EST EGAL A NULL) Alors

RETOURNER a

// parcours récursif de l'arbre

SINON SI (e SUP. STRICT. A element(a))

➡ **fd(a)** ← suppression(fd(a), e)

SINON SI (e INF. STRICT. A element(a))

➡ ➡ **fg(a)** ← suppression(fg(a), e)

// élément trouvé : remplacement par fils unique

SINON SI NON (existeFilsGauche(a))

tmp ← a

a ← **fd(a)**

libérer(tmp)

// élément trouvé : remplacement par prédécesseur

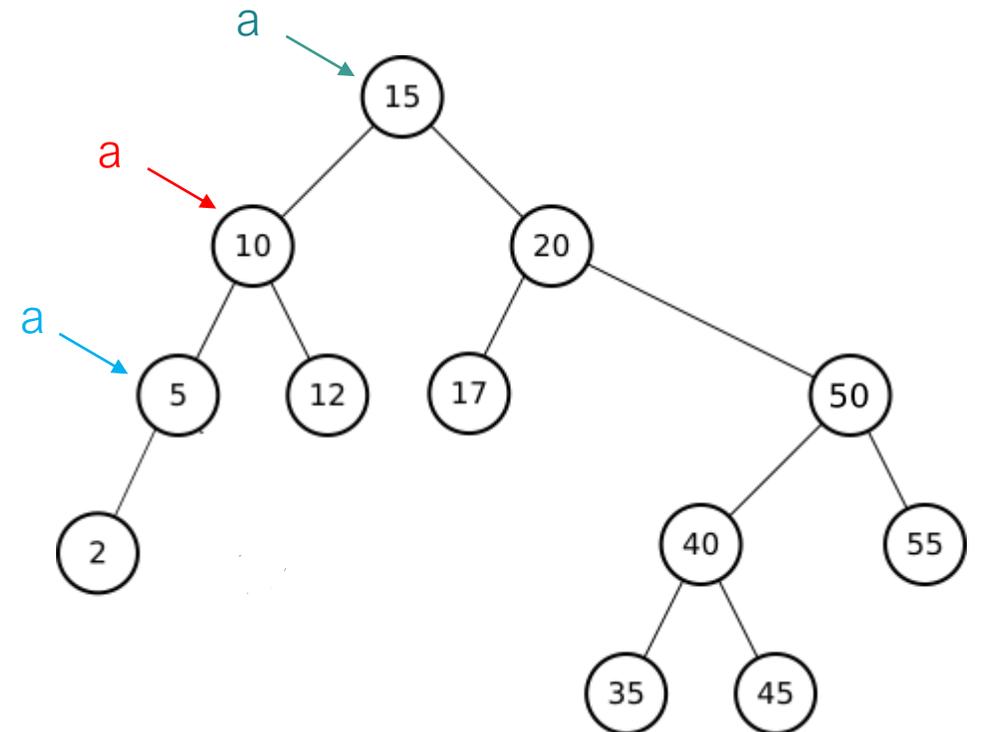
SINON

fg(a) ← supMax(fg(a), adresse(element(a)))

FIN SI

RETOURNER a

FIN



ABR : opération de suppression

- Algorithme : suppression (a, 8)

FONCTION suppression(a: pointeur sur Arbre, e: element) :
pointeur sur Arbre

VARIABLE

tmp : ptr sur Arbre

DEBUT

// element non présent dans l'arbre

SI (a EST EGAL A NULL) Alors

RETOURNER a

// parcours récursif de l'arbre

SINON SI (e SUP. STRICT. A element(a))

fd(a) ← suppression(fd(a), e)

SINON SI (e INF. STRICT. A element(a))

➡ ➡ fg(a) ← suppression(fg(a), e)

// élément trouvé : remplacement par fils unique

SINON SI NON (existeFilsGauche(a))

tmp ← a

a ← fd(a)

libérer(tmp)

// élément trouvé : remplacement par prédécesseur

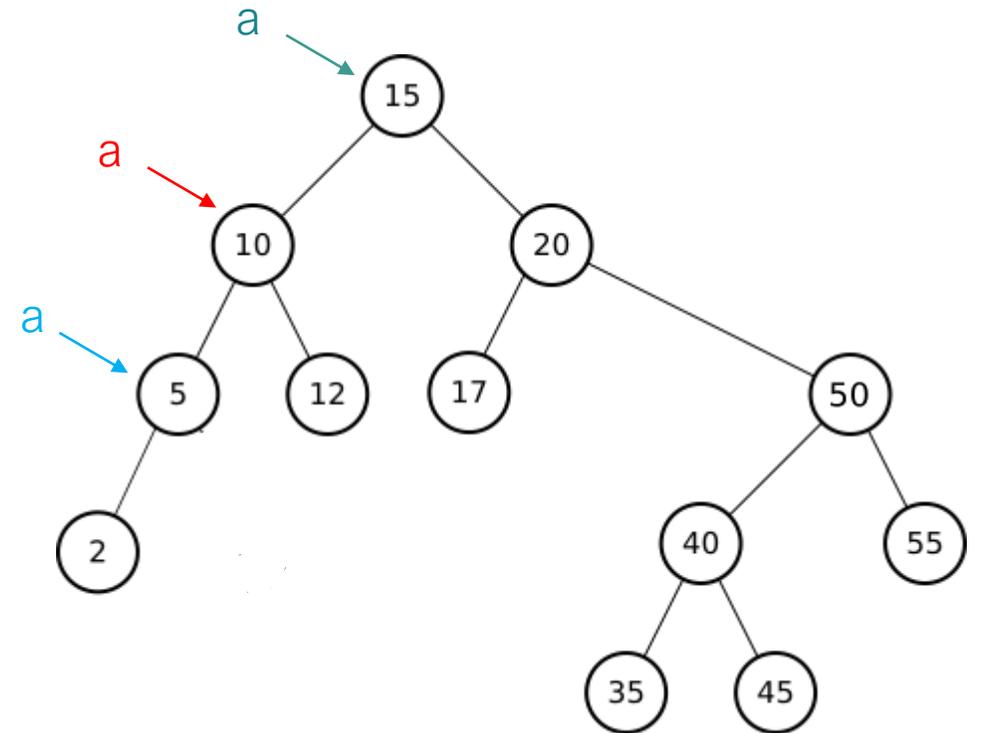
SINON

fg(a) ← suppMax(fg(a), adresse(element(a)))

FIN SI

➡ RETOURNER a

FIN



ABR : opération de suppression

- Algorithme : suppression (a, 8)

FONCTION suppression(a: pointeur sur Arbre, e: element) :
pointeur sur Arbre

VARIABLE

tmp : ptr sur Arbre

DEBUT

// element non présent dans l'arbre

SI (a EST EGAL A NULL) Alors

RETOURNER a

// parcours récursif de l'arbre

SINON SI (e SUP. STRICT. A element(a))

fd(a) ← suppression(fd(a), e)

SINON SI (e INF. STRICT. A element(a))

➡ ➡ fg(a) ← suppression(fg(a), e)

// élément trouvé : remplacement par fils unique

SINON SI NON (existeFilsGauche(a))

tmp ← a

a ← fd(a)

libérer(tmp)

// élément trouvé : remplacement par prédécesseur

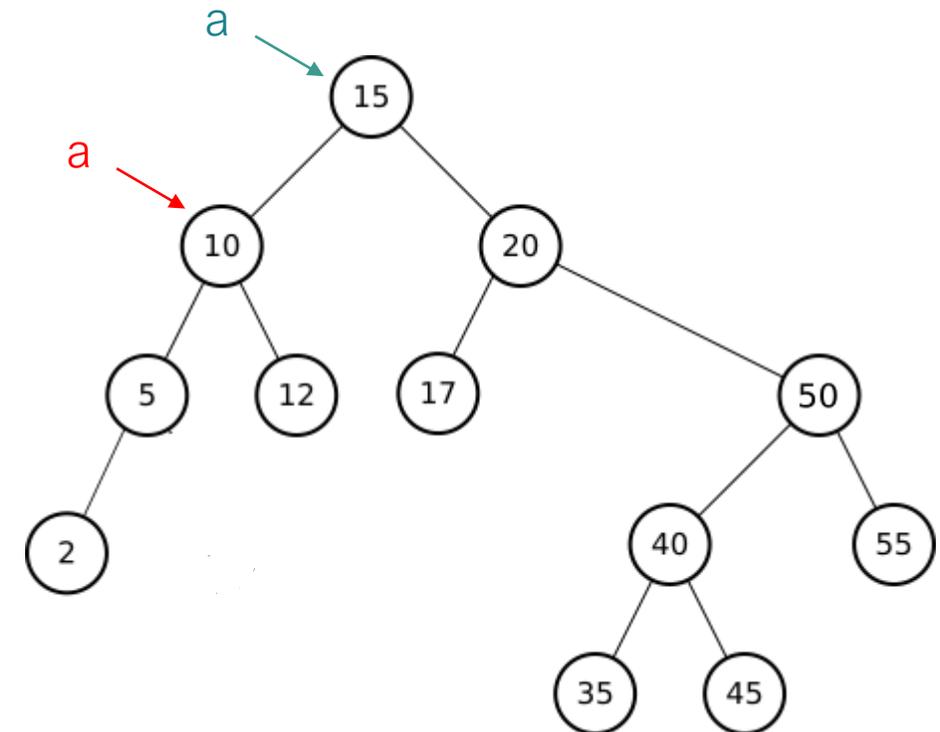
SINON

fg(a) ← suppMax(fg(a), adresse(element(a)))

FIN SI

RETOURNER a

FIN



ABR : opération de suppression

- Algorithme : suppression (a, 8)

FONCTION suppression(a: pointeur sur Arbre, e: element) :
pointeur sur Arbre

VARIABLE

tmp : ptr sur Arbre

DEBUT

// element non présent dans l'arbre

SI (a EST EGAL A NULL) Alors

RETOURNER a

// parcours récursif de l'arbre

SINON SI (e SUP. STRICT. A element(a))

fd(a) ← suppression(fd(a), e)

SINON SI (e INF. STRICT. A element(a))

➡ fg(a) ← suppression(fg(a), e)

// élément trouvé : remplacement par fils unique

SINON SI NON (existeFilsGauche(a))

tmp ← a

a ← fd(a)

libérer(tmp)

// élément trouvé : remplacement par prédécesseur

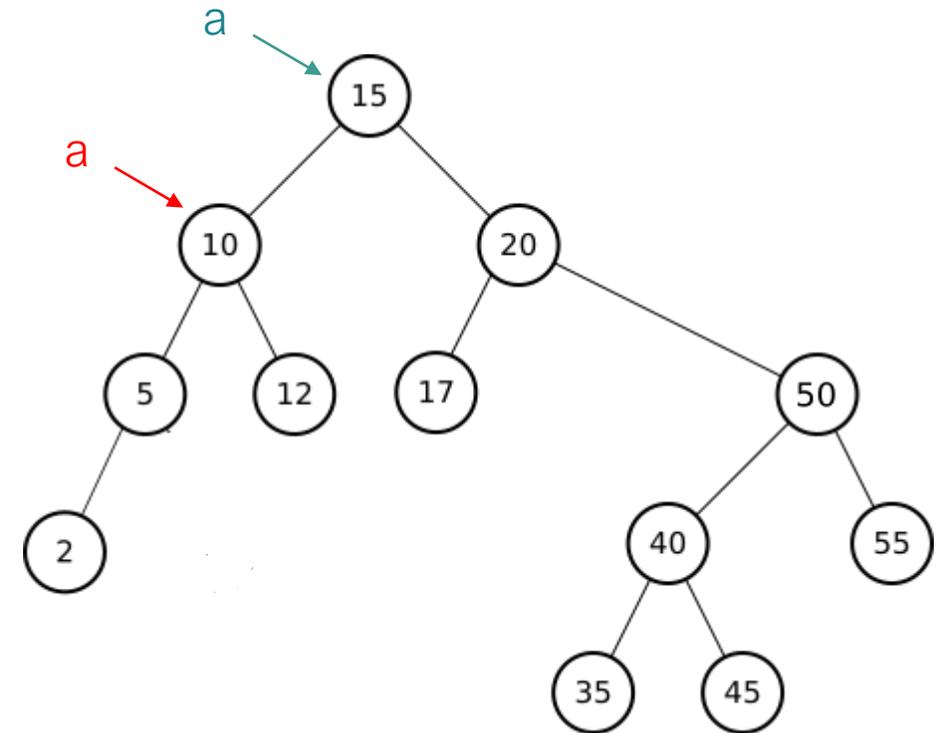
SINON

fg(a) ← suppMax(fg(a), adresse(element(a)))

FIN SI

➡ RETOURNER a

FIN



ABR : opération de suppression

- Algorithme : suppression (a, 8)

FONCTION suppression(a: pointeur sur Arbre, e: element) :
pointeur sur Arbre

VARIABLE

tmp : ptr sur Arbre

DEBUT

// element non présent dans l'arbre

SI (a EST EGAL A NULL) Alors

RETOURNER a

// parcours récursif de l'arbre

SINON SI (e SUP. STRICT. A element(a))

fd(a) ← suppression(fd(a), e)

SINON SI (e INF. STRICT. A element(a))

➡ fg(a) ← suppression(fg(a), e)

// élément trouvé : remplacement par fils unique

SINON SI NON (existeFilsGauche(a))

tmp ← a

a ← fd(a)

libérer(tmp)

// élément trouvé : remplacement par prédécesseur

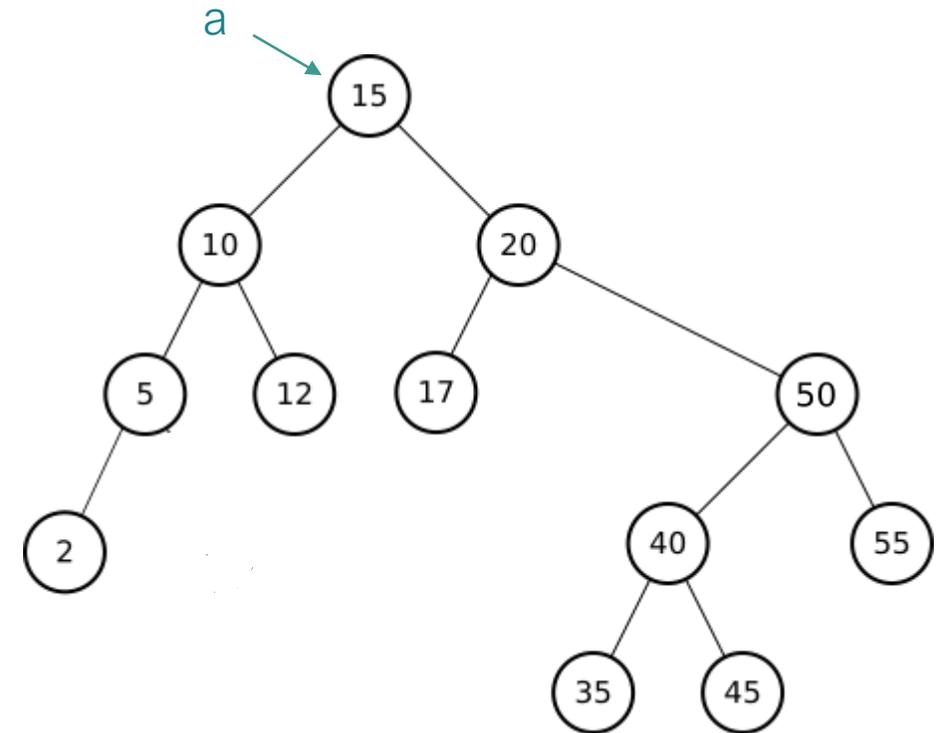
SINON

fg(a) ← suppMax(fg(a), adresse(element(a)))

FIN SI

RETOURNER a

FIN



ABR : opération de suppression

- Algorithme : suppression (a, 8)

FONCTION suppression(a: pointeur sur Arbre, e: element) :
pointeur sur Arbre

VARIABLE

tmp : ptr sur Arbre

DEBUT

// element non présent dans l'arbre

SI (a EST EGAL A NULL) Alors

RETOURNER a

// parcours récursif de l'arbre

SINON SI (e SUP. STRICT. A element(a))

fd(a) ← suppression(fd(a), e)

SINON SI (e INF. STRICT. A element(a))

fg(a) ← suppression(fg(a), e)

// élément trouvé : remplacement par fils unique

SINON SI NON (existeFilsGauche(a))

tmp ← a

a ← fd(a)

libérer(tmp)

// élément trouvé : remplacement par prédécesseur

SINON

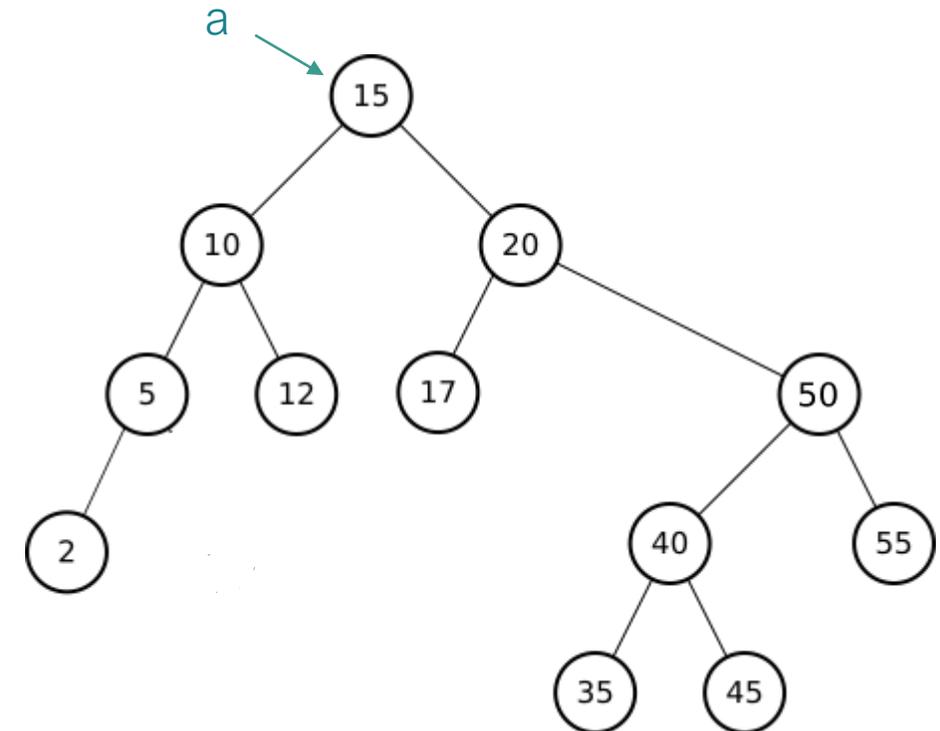
fg(a) ← suppMax(fg(a), adresse(element(a)))

FIN SI

RETOURNER a

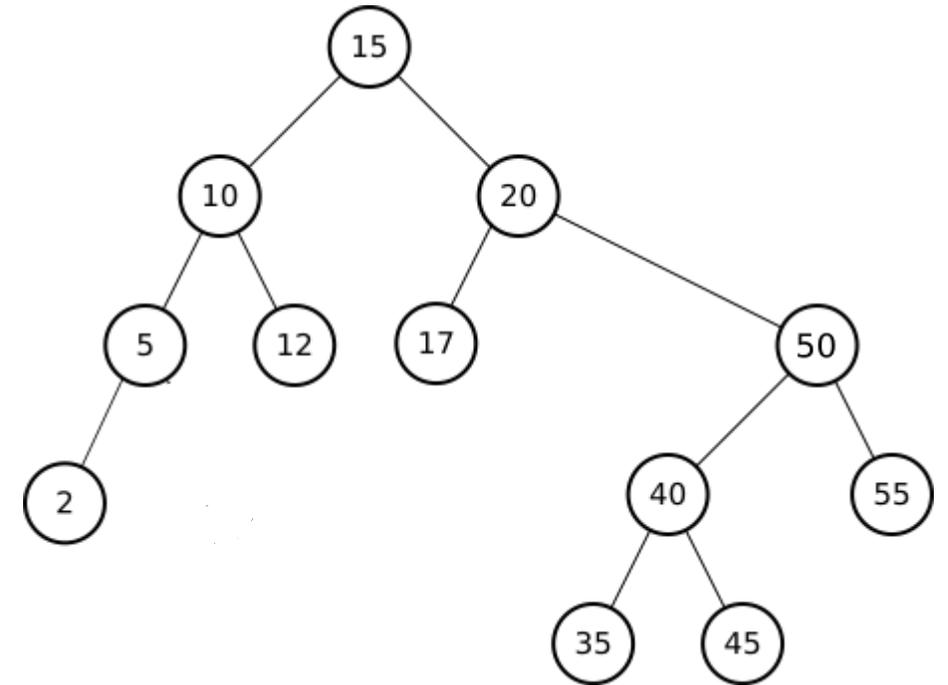


FIN



ABR : opération de suppression

- Algorithme : suppression (a, 6)



ABR : opération de suppression

- Algorithme : suppression (a, 6)

FONCTION suppression(a: pointeur sur Arbre, e: element) :
pointeur sur Arbre

VARIABLE

tmp : ptr sur Arbre

DEBUT

// element non présent dans l'arbre

SI (a EST EGAL A NULL) Alors

RETOURNER a

// parcours récursif de l'arbre

SINON SI (e SUP. STRICT. A element(a))

fd(a) ← suppression(fd(a), e)

SINON SI (e INF. STRICT. A element(a))

fg(a) ← suppression(fg(a), e)

// élément trouvé : remplacement par fils unique

SINON SI NON (existeFilsGauche(a))

tmp ← a

a ← fd(a)

libérer(tmp)

// élément trouvé : remplacement par prédécesseur

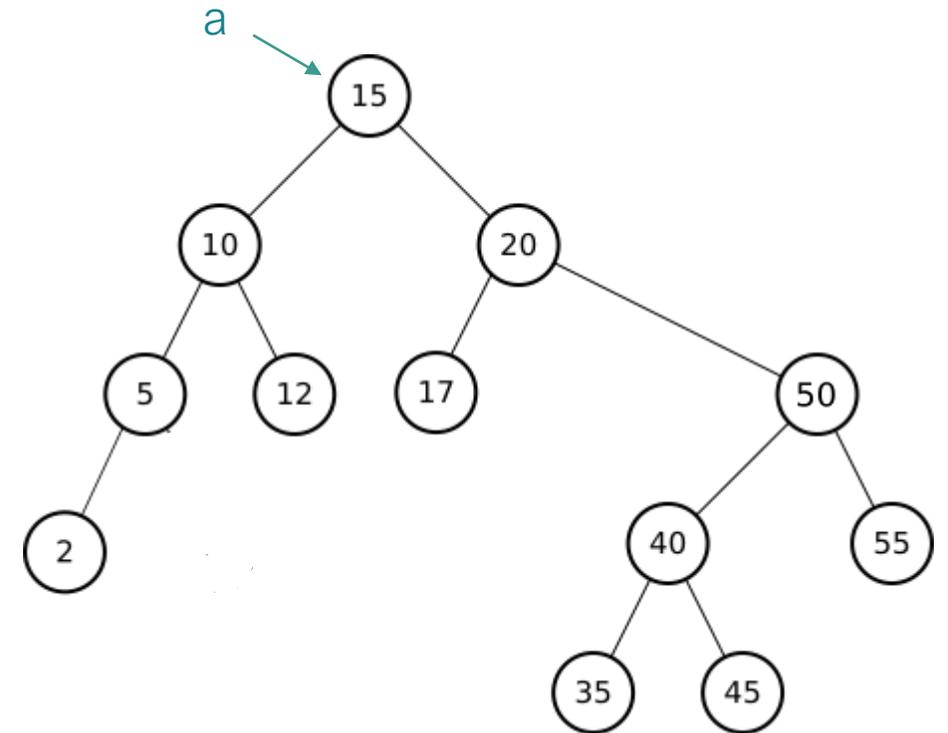
SINON

fg(a) ← suppMax(fg(a), adresse(element(a)))

FIN SI

RETOURNER a

FIN



ABR : opération de suppression

- Algorithme : suppression (a, 6)

FONCTION suppression(a: pointeur sur Arbre, e: element) :
pointeur sur Arbre

VARIABLE

tmp : ptr sur Arbre

DEBUT

// element non présent dans l'arbre

➔ SI (a EST EGAL A NULL) Alors

RETOURNER a

// parcours récursif de l'arbre

SINON SI (e SUP. STRICT. A element(a))

fd(a) ← suppression(fd(a), e)

SINON SI (e INF. STRICT. A element(a))

fg(a) ← suppression(fg(a), e)

// élément trouvé : remplacement par fils unique

SINON SI NON (existeFilsGauche(a))

tmp ← a

a ← fd(a)

libérer(tmp)

// élément trouvé : remplacement par prédécesseur

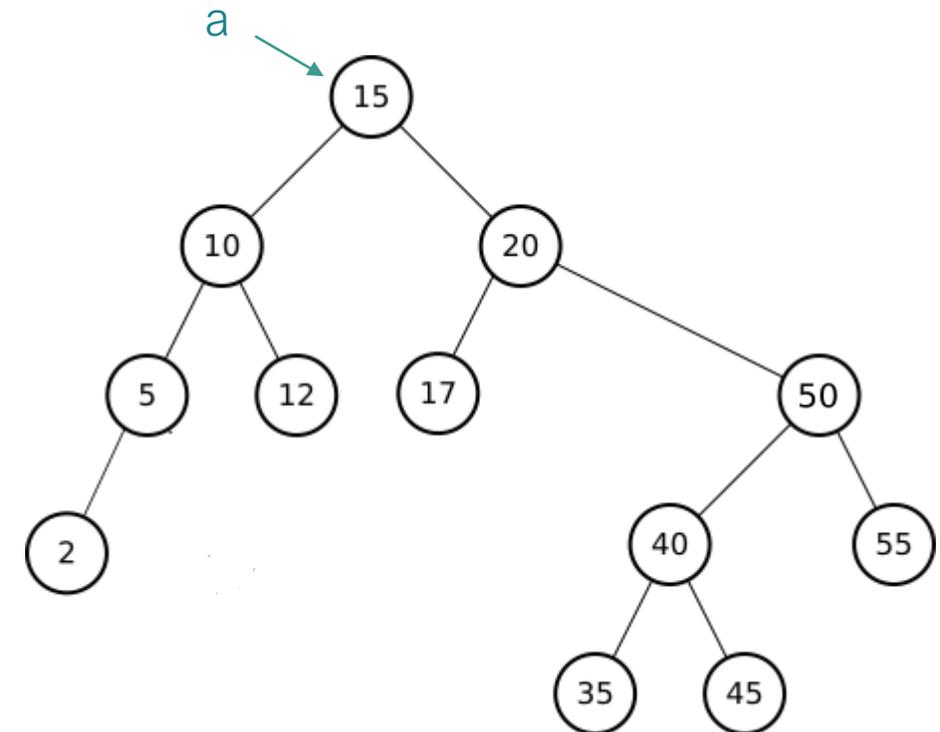
SINON

fg(a) ← suppMax(fg(a), adresse(element(a)))

FIN SI

RETOURNER a

FIN



ABR : opération de suppression

- Algorithme : suppression (a, 6)

FONCTION suppression(a: pointeur sur Arbre, e: element) :
pointeur sur Arbre

VARIABLE

tmp : ptr sur Arbre

DEBUT

// element non présent dans l'arbre

SI (a EST EGAL A NULL) Alors

RETOURNER a

// parcours récursif de l'arbre

➔ SINON SI (e SUP. STRICT. A element(a))

fd(a) ← suppression(fd(a), e)

SINON SI (e INF. STRICT. A element(a))

fg(a) ← suppression(fg(a), e)

// élément trouvé : remplacement par fils unique

SINON SI NON (existeFilsGauche(a))

tmp ← a

a ← fd(a)

libérer(tmp)

// élément trouvé : remplacement par prédécesseur

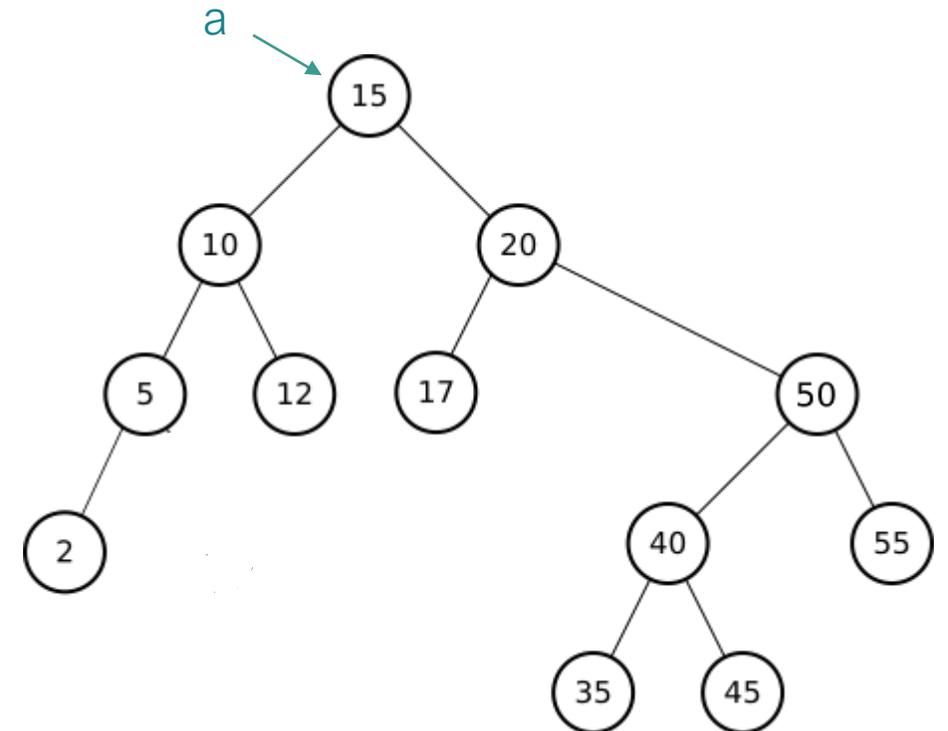
SINON

fg(a) ← suppMax(fg(a), adresse(element(a)))

FIN SI

RETOURNER a

FIN



ABR : opération de suppression

- Algorithme : suppression (a, 6)

FONCTION suppression(a: pointeur sur Arbre, e: element) :
pointeur sur Arbre

VARIABLE

tmp : ptr sur Arbre

DEBUT

// element non présent dans l'arbre

SI (a EST EGAL A NULL) Alors

RETOURNER a

// parcours récursif de l'arbre

SINON SI (e SUP. STRICT. A element(a))

fd(a) ← suppression(fd(a), e)

➔ SINON SI (e INF. STRICT. A element(a))

fg(a) ← suppression(fg(a), e)

// élément trouvé : remplacement par fils unique

SINON SI NON (existeFilsGauche(a))

tmp ← a

a ← fd(a)

libérer(tmp)

// élément trouvé : remplacement par prédécesseur

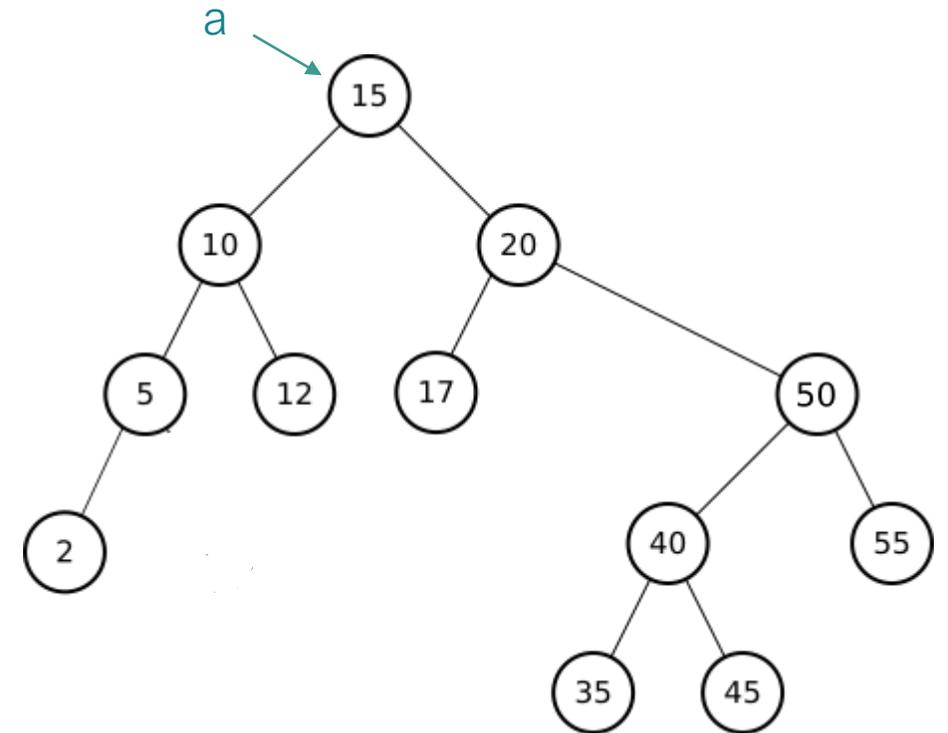
SINON

fg(a) ← suppMax(fg(a), adresse(element(a)))

FIN SI

RETOURNER a

FIN



ABR : opération de suppression

- Algorithme : suppression (a, 6)

FONCTION suppression(a: pointeur sur Arbre, e: element) :
pointeur sur Arbre

VARIABLE

tmp : ptr sur Arbre

DEBUT

// element non présent dans l'arbre

SI (a EST EGAL A NULL) Alors

RETOURNER a

// parcours récursif de l'arbre

SINON SI (e SUP. STRICT. A element(a))

fd(a) ← suppression(fd(a), e)

SINON SI (e INF. STRICT. A element(a))

➡ fg(a) ← suppression(fg(a), e)

// élément trouvé : remplacement par fils unique

SINON SI NON (existeFilsGauche(a))

tmp ← a

a ← fd(a)

libérer(tmp)

// élément trouvé : remplacement par prédécesseur

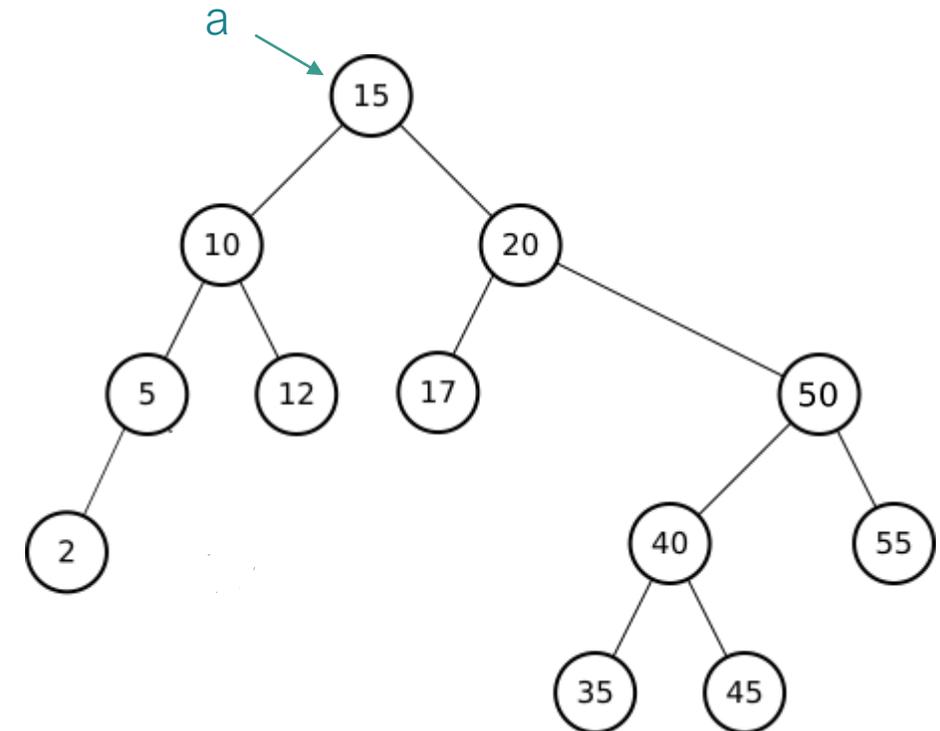
SINON

fg(a) ← suppMax(fg(a), adresse(element(a)))

FIN SI

RETOURNER a

FIN



ABR : opération de suppression

- Algorithme : suppression (a, 6)

FONCTION suppression(a: pointeur sur Arbre, e: element) :
pointeur sur Arbre

VARIABLE

tmp : ptr sur Arbre

→ DEBUT

// element non présent dans l'arbre

SI (a EST EGAL A NULL) Alors

RETOURNER a

// parcours récursif de l'arbre

SINON SI (e SUP. STRICT. A element(a))

fd(a) ← suppression(fd(a), e)

SINON SI (e INF. STRICT. A element(a))

→ fg(a) ← suppression(fg(a), e)

// élément trouvé : remplacement par fils unique

SINON SI NON (existeFilsGauche(a))

tmp ← a

a ← fd(a)

libérer(tmp)

// élément trouvé : remplacement par prédécesseur

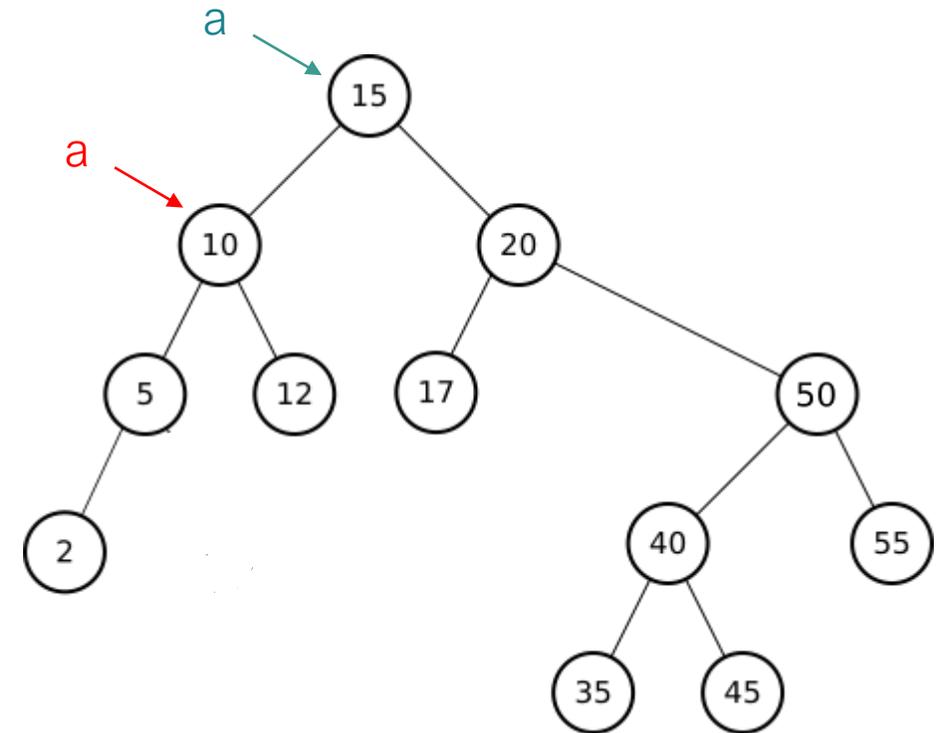
SINON

fg(a) ← suppMax(fg(a), adresse(element(a)))

FIN SI

RETOURNER a

FIN



ABR : opération de suppression

- Algorithme : suppression (a, 6)

FONCTION suppression(a: pointeur sur Arbre, e: element) :
pointeur sur Arbre

VARIABLE

tmp : ptr sur Arbre

DEBUT

// element non présent dans l'arbre

➔ SI (a EST EGAL A NULL) Alors
RETOURNER a

// parcours récursif de l'arbre

SINON SI (e SUP. STRICT. A element(a))

fd(a) ← suppression(fd(a), e)

SINON SI (e INF. STRICT. A element(a))

➔ fg(a) ← suppression(fg(a), e)

// élément trouvé : remplacement par fils unique

SINON SI NON (existeFilsGauche(a))

tmp ← a

a ← fd(a)

libérer(tmp)

// élément trouvé : remplacement par prédécesseur

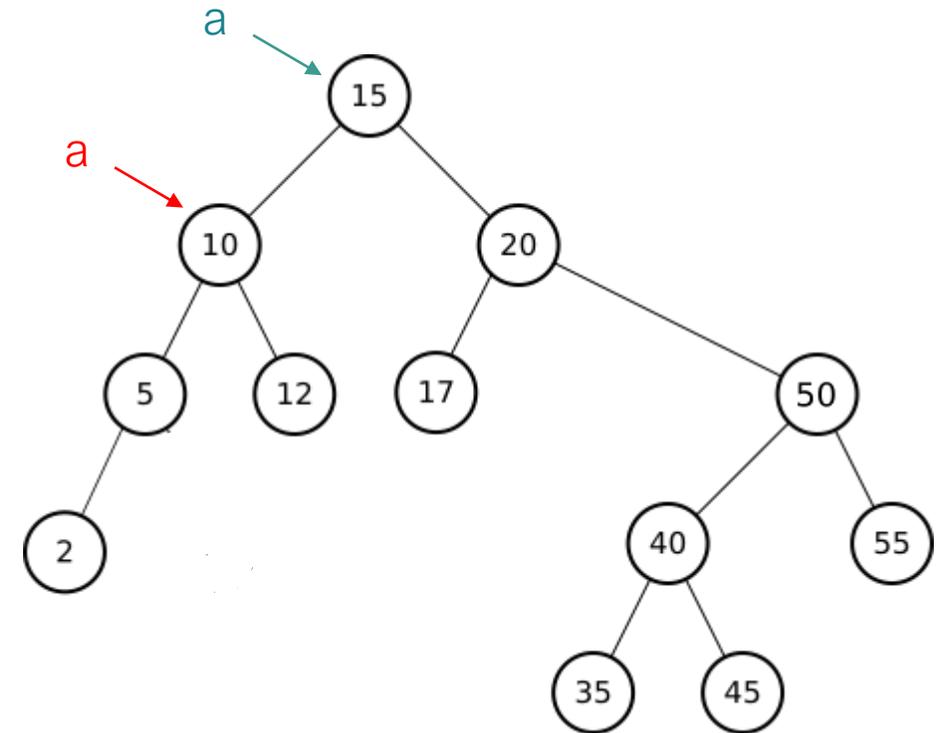
SINON

fg(a) ← suppMax(fg(a), adresse(element(a)))

FIN SI

RETOURNER a

FIN



ABR : opération de suppression

- Algorithme : suppression (a, 6)

FONCTION suppression(a: pointeur sur Arbre, e: element) :
pointeur sur Arbre

VARIABLE

tmp : ptr sur Arbre

DEBUT

// element non présent dans l'arbre

SI (a EST EGAL A NULL) Alors

RETOURNER a

// parcours récursif de l'arbre

➔ SINON SI (e SUP. STRICT. A element(a))

fd(a) ← suppression(fd(a), e)

SINON SI (e INF. STRICT. A element(a))

➔ fg(a) ← suppression(fg(a), e)

// élément trouvé : remplacement par fils unique

SINON SI NON (existeFilsGauche(a))

tmp ← a

a ← fd(a)

libérer(tmp)

// élément trouvé : remplacement par prédécesseur

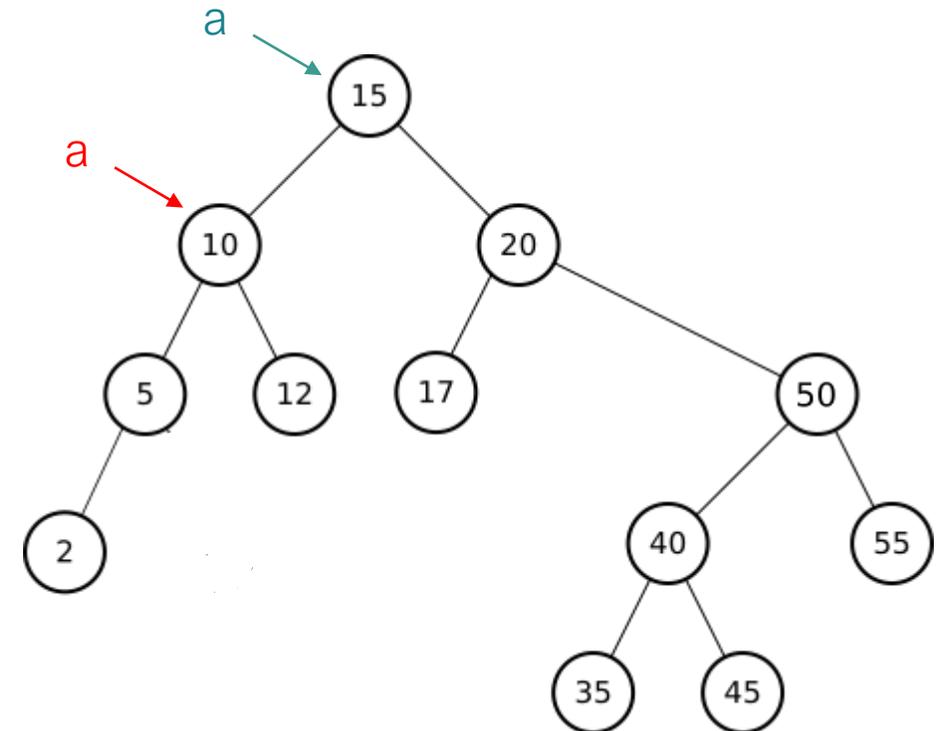
SINON

fg(a) ← suppMax(fg(a), adresse(element(a)))

FIN SI

RETOURNER a

FIN



ABR : opération de suppression

- Algorithme : suppression (a, 6)

FONCTION suppression(a: pointeur sur Arbre, e: element) :
pointeur sur Arbre

VARIABLE

tmp : ptr sur Arbre

DEBUT

// element non présent dans l'arbre

SI (a EST EGAL A NULL) Alors

RETOURNER a

// parcours récursif de l'arbre

SINON SI (e SUP. STRICT. A element(a))

fd(a) ← suppression(fd(a), e)

➔ SINON SI (e INF. STRICT. A element(a))

➔ fg(a) ← suppression(fg(a), e)

// élément trouvé : remplacement par fils unique

SINON SI NON (existeFilsGauche(a))

tmp ← a

a ← fd(a)

libérer(tmp)

// élément trouvé : remplacement par prédécesseur

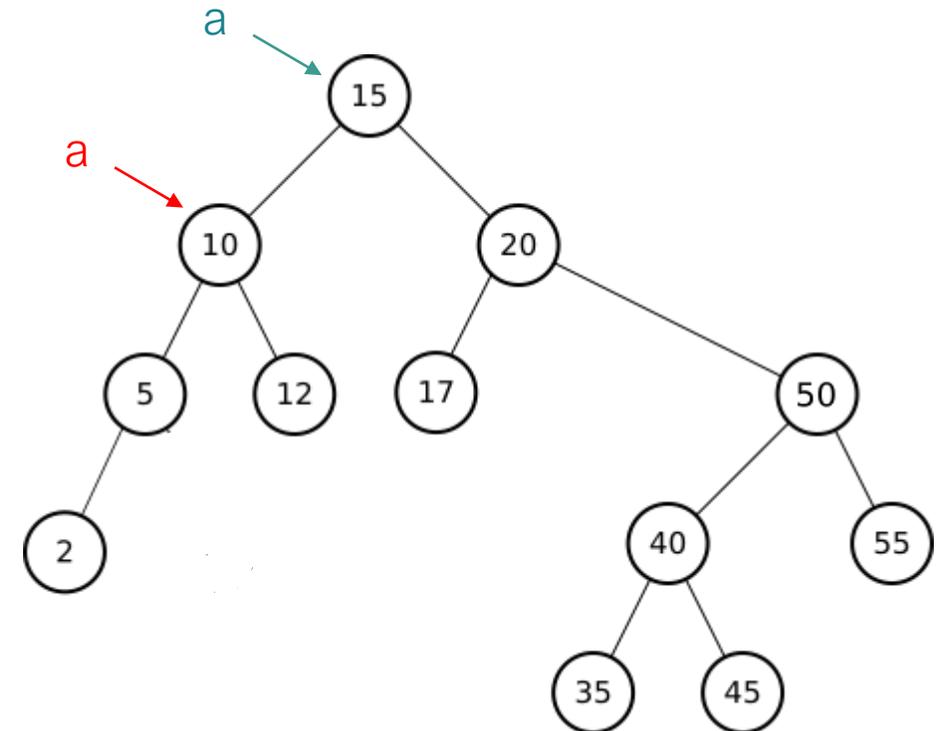
SINON

fg(a) ← suppMax(fg(a), adresse(element(a)))

FIN SI

RETOURNER a

FIN



ABR : opération de suppression

- Algorithme : suppression (a, 6)

FONCTION suppression(a: pointeur sur Arbre, e: element) :
pointeur sur Arbre

VARIABLE

tmp : ptr sur Arbre

DEBUT

// element non présent dans l'arbre

SI (a EST EGAL A NULL) Alors

RETOURNER a

// parcours récursif de l'arbre

SINON SI (e SUP. STRICT. A element(a))

fd(a) ← suppression(fd(a), e)

SINON SI (e INF. STRICT. A element(a))

➡ fg(a) ← suppression(fg(a), e)

// élément trouvé : remplacement par fils unique

SINON SI NON (existeFilsGauche(a))

tmp ← a

a ← fd(a)

libérer(tmp)

// élément trouvé : remplacement par prédécesseur

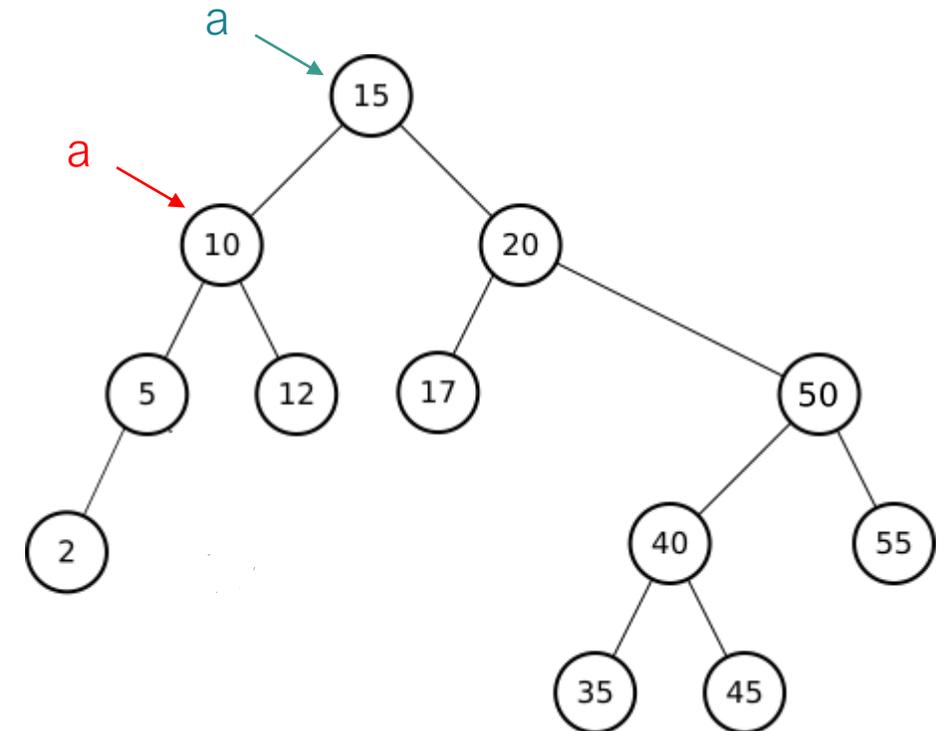
SINON

fg(a) ← suppMax(fg(a), adresse(element(a)))

FIN SI

RETOURNER a

FIN



ABR : opération de suppression

- Algorithme : suppression (a, 6)

FONCTION suppression(a: pointeur sur Arbre, e: element) :
pointeur sur Arbre

VARIABLE

tmp : ptr sur Arbre

→ DEBUT

// element non présent dans l'arbre

SI (a EST EGAL A NULL) Alors

RETOURNER a

// parcours récursif de l'arbre

SINON SI (e SUP. STRICT. A element(a))

fd(a) ← suppression(fd(a), e)

SINON SI (e INF. STRICT. A element(a))

→ fg(a) ← suppression(fg(a), e)

// élément trouvé : remplacement par fils unique

SINON SI NON (existeFilsGauche(a))

tmp ← a

a ← fd(a)

libérer(tmp)

// élément trouvé : remplacement par prédécesseur

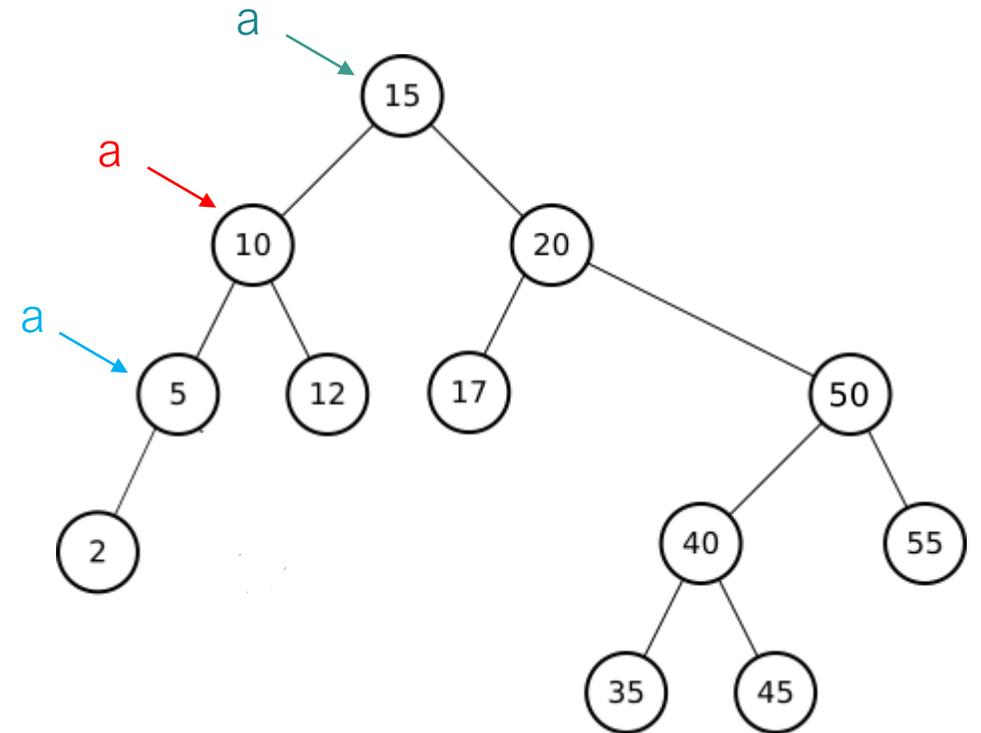
SINON

fg(a) ← suppMax(fg(a), adresse(element(a)))

FIN SI

RETOURNER a

FIN



ABR : opération de suppression

- Algorithme : suppression (a, 6)

FONCTION suppression(a: pointeur sur Arbre, e: element) :
pointeur sur Arbre

VARIABLE

tmp : ptr sur Arbre

DEBUT

// element non présent dans l'arbre

➡ SI (a EST EGAL A NULL) Alors
RETOURNER a

// parcours récursif de l'arbre

SINON SI (e SUP. STRICT. A element(a))

fd(a) ← suppression(fd(a), e)

SINON SI (e INF. STRICT. A element(a))

➡ ➡ fg(a) ← suppression(fg(a), e)

// élément trouvé : remplacement par fils unique

SINON SI NON (existeFilsGauche(a))

tmp ← a

a ← fd(a)

libérer(tmp)

// élément trouvé : remplacement par prédécesseur

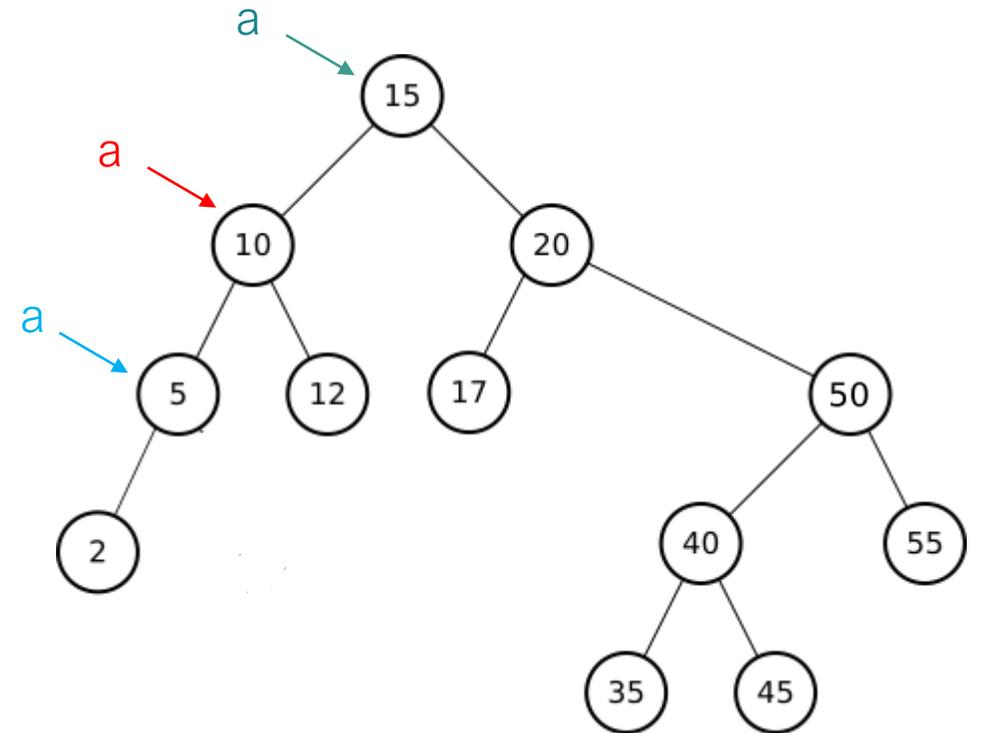
SINON

fg(a) ← suppMax(fg(a), adresse(element(a)))

FIN SI

RETOURNER a

FIN



ABR : opération de suppression

- Algorithme : suppression (a, 6)

FONCTION suppression(a: pointeur sur Arbre, e: element) :
pointeur sur Arbre

VARIABLE

tmp : ptr sur Arbre

DEBUT

// element non présent dans l'arbre

SI (a EST EGAL A NULL) Alors

RETOURNER a

// parcours récursif de l'arbre

➔ SINON SI (e SUP. STRICT. A element(a))

fd(a) ← suppression(fd(a), e)

SINON SI (e INF. STRICT. A element(a))

➔ ➔ fg(a) ← suppression(fg(a), e)

// élément trouvé : remplacement par fils unique

SINON SI NON (existeFilsGauche(a))

tmp ← a

a ← fd(a)

libérer(tmp)

// élément trouvé : remplacement par prédécesseur

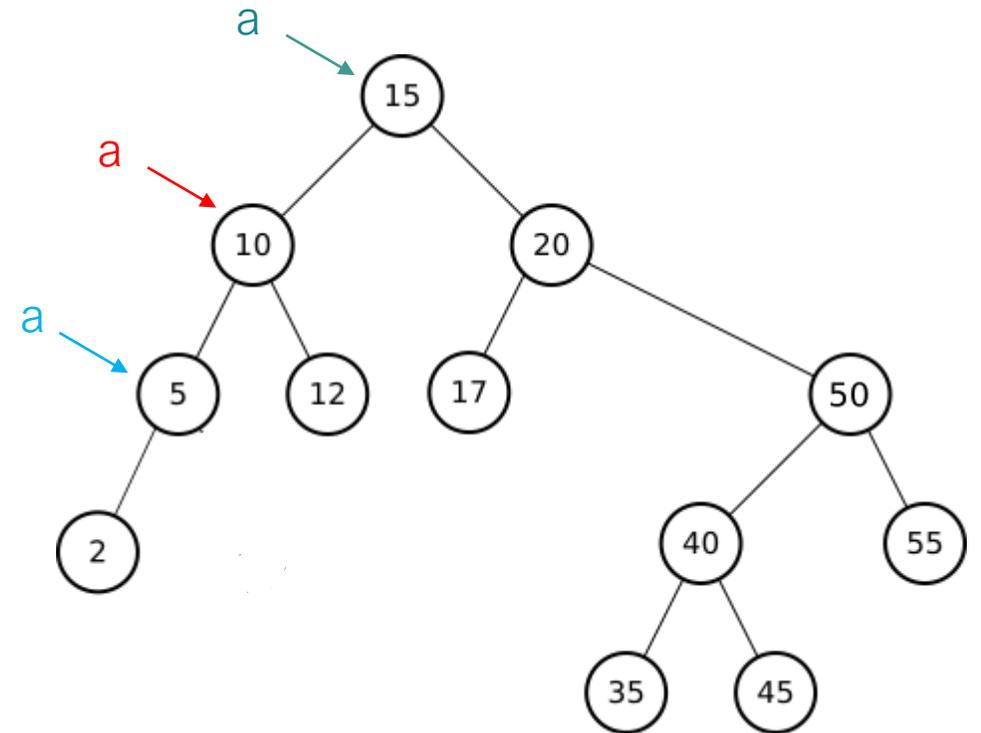
SINON

fg(a) ← suppMax(fg(a), adresse(element(a)))

FIN SI

RETOURNER a

FIN



ABR : opération de suppression

- Algorithme : suppression (a, 6)

FONCTION suppression(a: pointeur sur Arbre, e: element) :
pointeur sur Arbre

VARIABLE

tmp : ptr sur Arbre

DEBUT

// element non présent dans l'arbre

SI (a EST EGAL A NULL) Alors

RETOURNER a

// parcours récursif de l'arbre

SINON SI (e SUP. STRICT. A element(a))

➡ $fd(a) \leftarrow suppression(fd(a), e)$

SINON SI (e INF. STRICT. A element(a))

➡➡ $fg(a) \leftarrow suppression(fg(a), e)$

// élément trouvé : remplacement par fils unique

SINON SI NON (existeFilsGauche(a))

tmp \leftarrow a

a \leftarrow fd(a)

libérer(tmp)

// élément trouvé : remplacement par prédécesseur

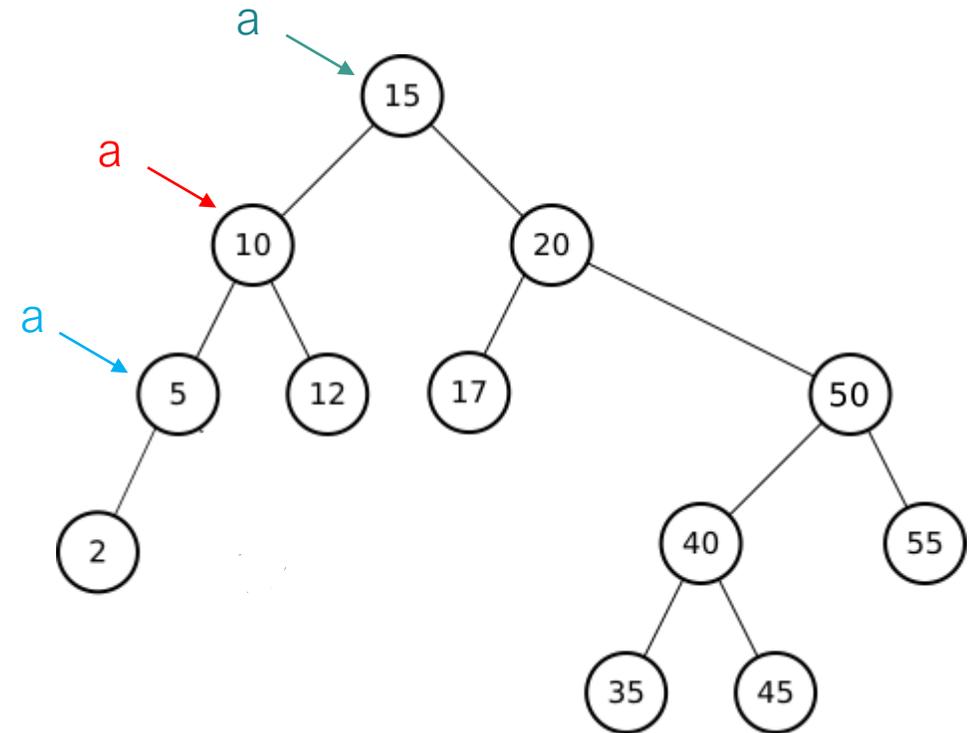
SINON

$fg(a) \leftarrow suppMax(fg(a), adresse(element(a)))$

FIN SI

RETOURNER a

FIN



ABR : opération de suppression

- Algorithme : suppression (a, 6)

FONCTION suppression(a: pointeur sur Arbre, e: element) :
pointeur sur Arbre

VARIABLE

tmp : ptr sur Arbre

DEBUT

// element non présent dans l'arbre

SI (a EST EGAL A NULL) Alors

RETOURNER a

// parcours récursif de l'arbre

SINON SI (e SUP. STRICT. A element(a))

➡ fd(a) ← suppression(fd(a), e)

SINON SI (e INF. STRICT. A element(a))

➡ fg(a) ← suppression(fg(a), e)

// élément trouvé : remplacement par fils unique

SINON SI NON (existeFilsGauche(a))

tmp ← a

a ← fd(a)

libérer(tmp)

// élément trouvé : remplacement par prédécesseur

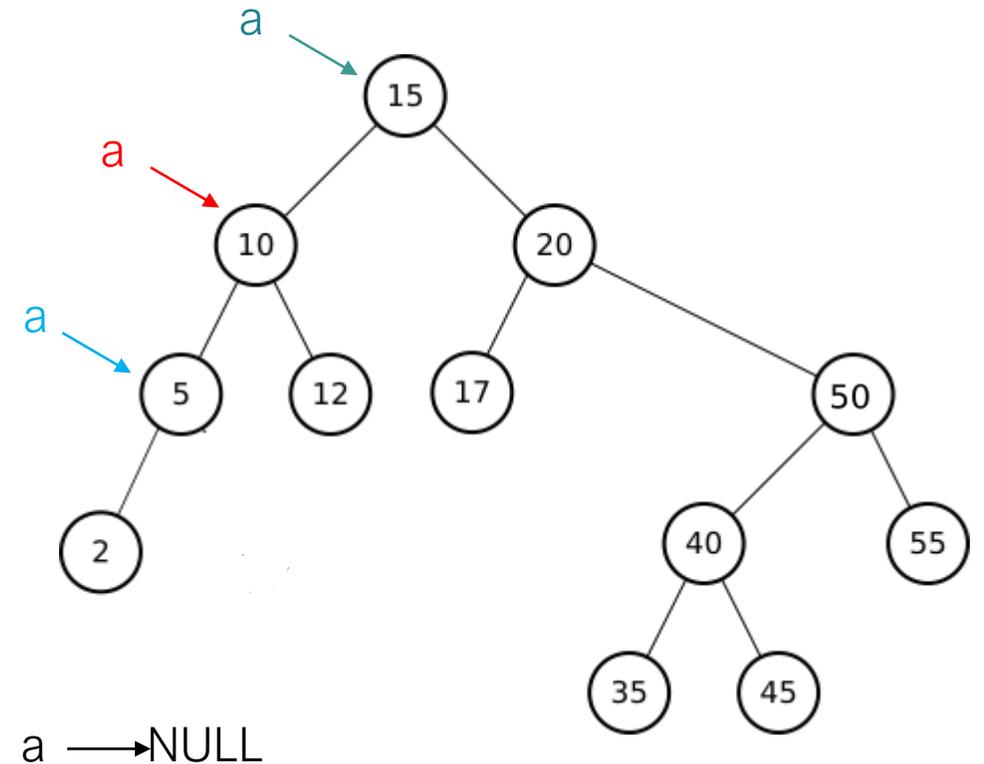
SINON

fg(a) ← suppMax(fg(a), adresse(element(a)))

FIN SI

RETOURNER a

FIN



ABR : opération de suppression

- Algorithme : suppression (a, 6)

FONCTION suppression(a: pointeur sur Arbre, e: element) :
pointeur sur Arbre

VARIABLE

tmp : ptr sur Arbre

DEBUT

// element non présent dans l'arbre

➔ SI (a EST EGAL A NULL) Alors
RETOURNER a

// parcours récursif de l'arbre

SINON SI (e SUP. STRICT. A element(a))

➔ fd(a) ← suppression(fd(a), e)

SINON SI (e INF. STRICT. A element(a))

➔ fg(a) ← suppression(fg(a), e)

// élément trouvé : remplacement par fils unique

SINON SI NON (existeFilsGauche(a))

tmp ← a

a ← fd(a)

libérer(tmp)

// élément trouvé : remplacement par prédécesseur

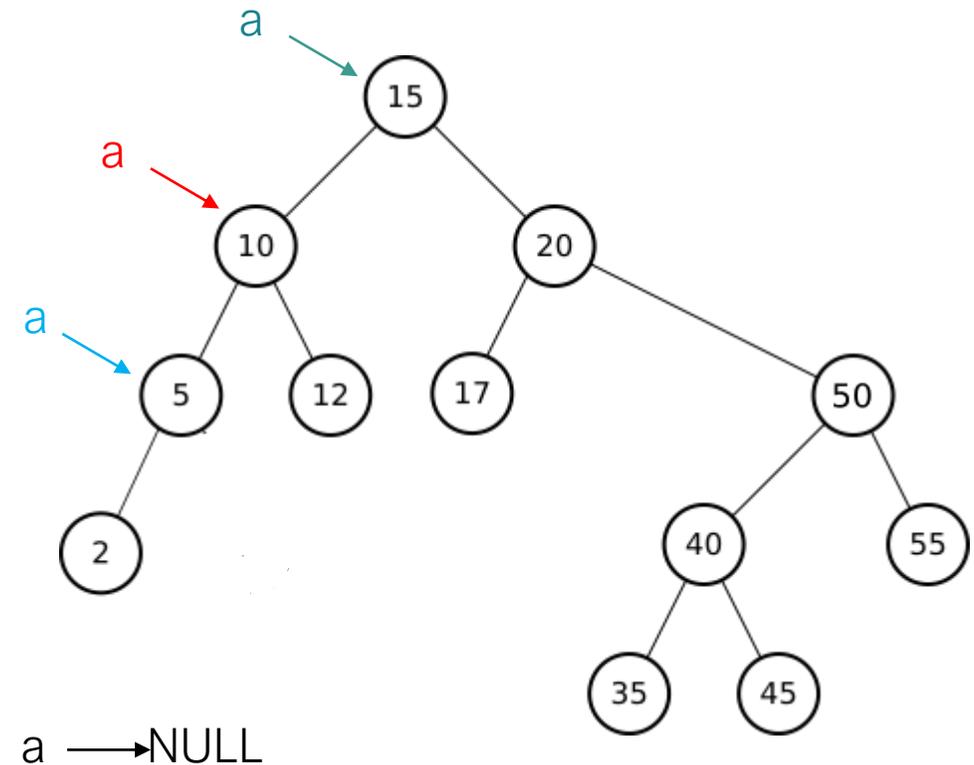
SINON

fg(a) ← suppMax(fg(a), adresse(element(a)))

FIN SI

RETOURNER a

FIN



ABR : opération de suppression

- Algorithme : suppression (a, 6)

FONCTION suppression(a: pointeur sur Arbre, e: element) :
pointeur sur Arbre

VARIABLE

tmp : ptr sur Arbre

DEBUT

// element non présent dans l'arbre

SI (a EST EGAL A NULL) Alors

➡ RETOURNER a

// parcours récursif de l'arbre

SINON SI (e SUP. STRICT. A element(a))

➡ fd(a) ← suppression(fd(a), e)

SINON SI (e INF. STRICT. A element(a))

➡ fg(a) ← suppression(fg(a), e)

// élément trouvé : remplacement par fils unique

SINON SI NON (existeFilsGauche(a))

tmp ← a

a ← fd(a)

libérer(tmp)

// élément trouvé : remplacement par prédécesseur

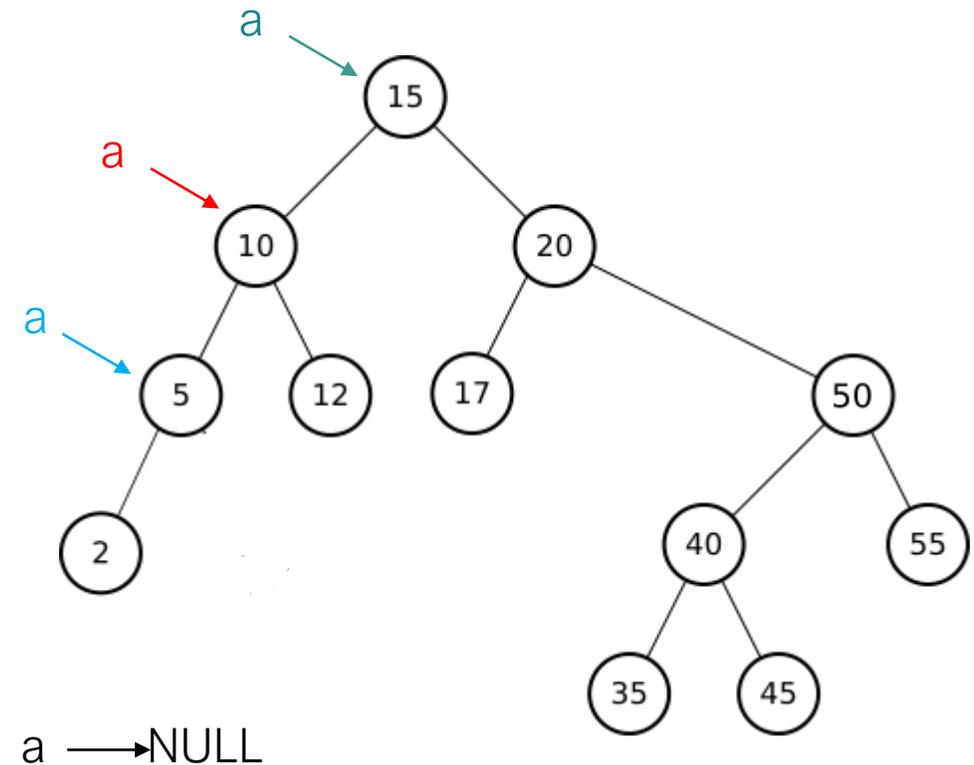
SINON

fg(a) ← suppMax(fg(a), adresse(element(a)))

FIN SI

RETOURNER a

FIN



ABR : opération de suppression

- Algorithme : suppression (a, 6)

FONCTION suppression(a: pointeur sur Arbre, e: element) :
pointeur sur Arbre

VARIABLE

tmp : ptr sur Arbre

DEBUT

// element non présent dans l'arbre

SI (a EST EGAL A NULL) Alors

RETOURNER a

// parcours récursif de l'arbre

SINON SI (e SUP. STRICT. A element(a))

➡ **fd(a)** ← suppression(fd(a), e)

SINON SI (e INF. STRICT. A element(a))

➡➡ **fg(a)** ← suppression(fg(a), e)

// élément trouvé : remplacement par fils unique

SINON SI NON (existeFilsGauche(a))

tmp ← a

a ← **fd(a)**

libérer(tmp)

// élément trouvé : remplacement par prédécesseur

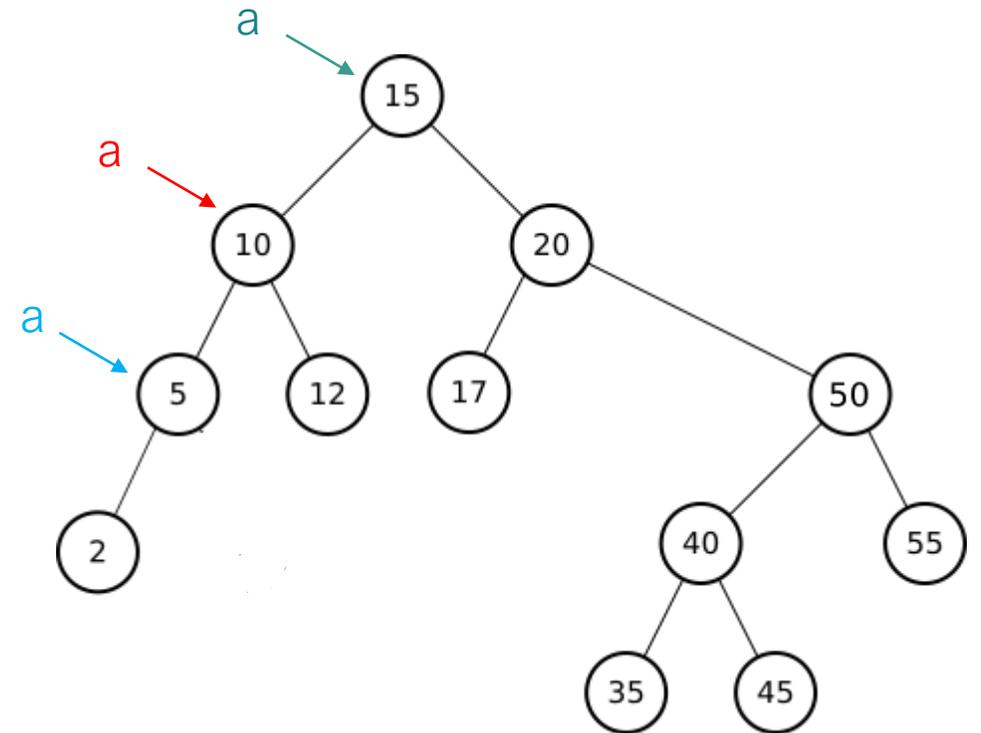
SINON

fg(a) ← supMax(fg(a), adresse(element(a)))

FIN SI

RETOURNER a

FIN



ABR : opération de suppression

- Algorithme : suppression (a, 6)

FONCTION suppression(a: pointeur sur Arbre, e: element) :
pointeur sur Arbre

VARIABLE

tmp : ptr sur Arbre

DEBUT

// element non présent dans l'arbre

SI (a EST EGAL A NULL) Alors

RETOURNER a

// parcours récursif de l'arbre

SINON SI (e SUP. STRICT. A element(a))

fd(a) ← suppression(fd(a), e)

SINON SI (e INF. STRICT. A element(a))

➡ fg(a) ← suppression(fg(a), e)

// élément trouvé : remplacement par fils unique

SINON SI NON (existeFilsGauche(a))

tmp ← a

a ← fd(a)

libérer(tmp)

// élément trouvé : remplacement par prédécesseur

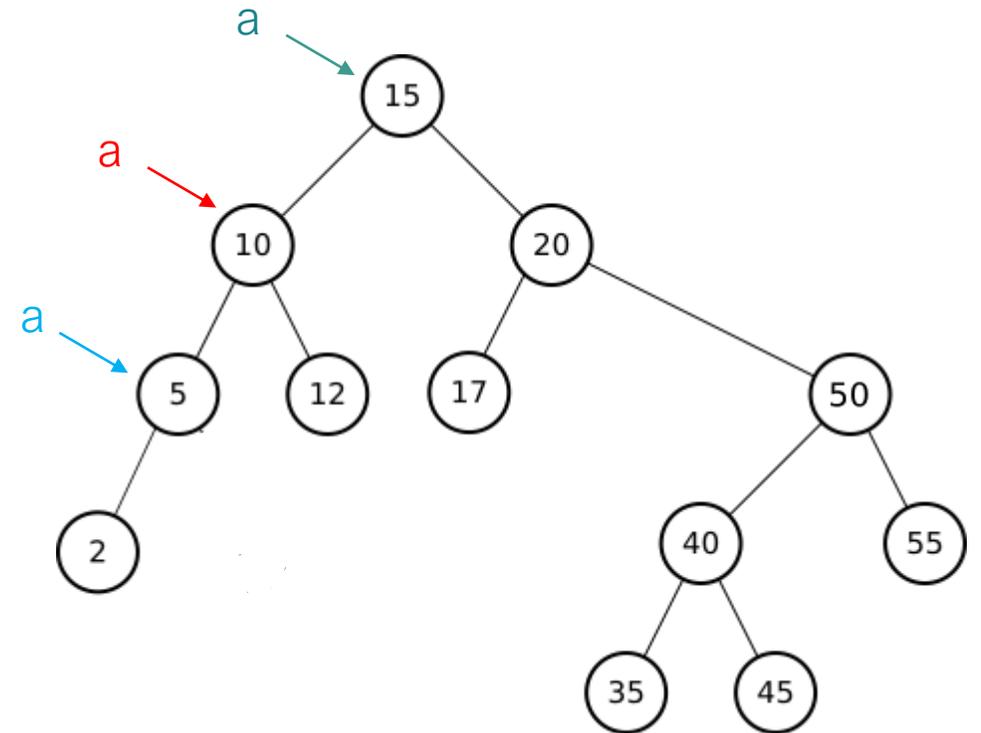
SINON

fg(a) ← suppMax(fg(a), adresse(element(a)))

FIN SI

➡ RETOURNER a

FIN



ABR : opération de suppression

- Algorithme : suppression (a, 6)

FONCTION suppression(a: pointeur sur Arbre, e: element) :
pointeur sur Arbre

VARIABLE

tmp : ptr sur Arbre

DEBUT

// element non présent dans l'arbre

SI (a EST EGAL A NULL) Alors

RETOURNER a

// parcours récursif de l'arbre

SINON SI (e SUP. STRICT. A element(a))

fd(a) ← suppression(fd(a), e)

SINON SI (e INF. STRICT. A element(a))

➡ fg(a) ← suppression(fg(a), e)

// élément trouvé : remplacement par fils unique

SINON SI NON (existeFilsGauche(a))

tmp ← a

a ← fd(a)

libérer(tmp)

// élément trouvé : remplacement par prédécesseur

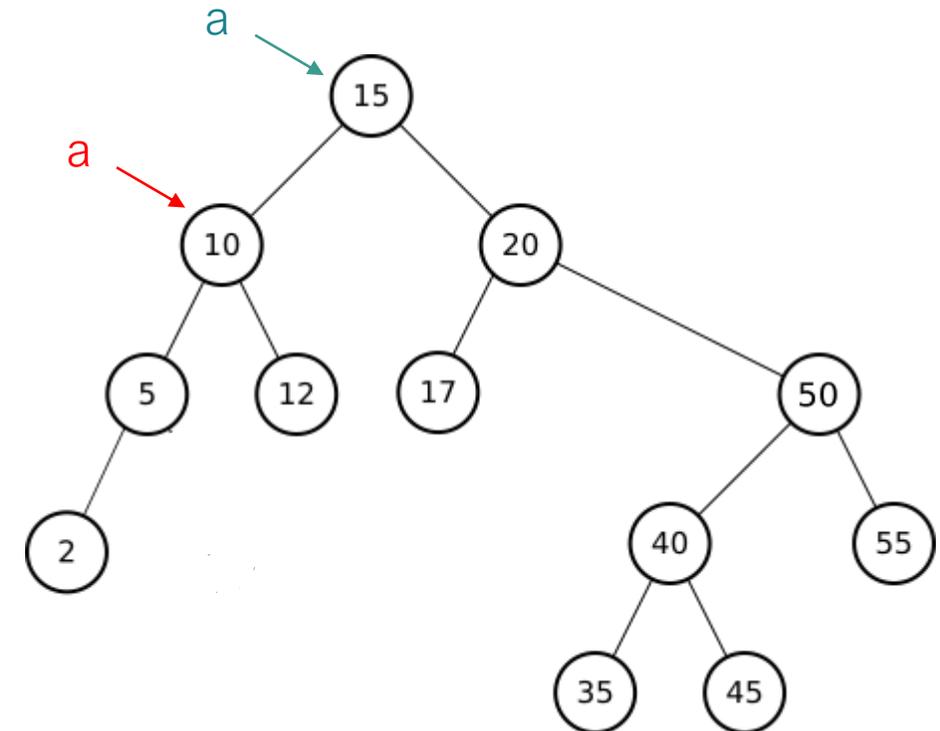
SINON

fg(a) ← suppMax(fg(a), adresse(element(a)))

FIN SI

RETOURNER a

FIN



ABR : opération de suppression

- Algorithme : suppression (a, 6)

FONCTION suppression(a: pointeur sur Arbre, e: element) :
pointeur sur Arbre

VARIABLE

tmp : ptr sur Arbre

DEBUT

// element non présent dans l'arbre

SI (a EST EGAL A NULL) Alors

RETOURNER a

// parcours récursif de l'arbre

SINON SI (e SUP. STRICT. A element(a))

fd(a) ← suppression(fd(a), e)

SINON SI (e INF. STRICT. A element(a))

→ fg(a) ← suppression(fg(a), e)

// élément trouvé : remplacement par fils unique

SINON SI NON (existeFilsGauche(a))

tmp ← a

a ← fd(a)

libérer(tmp)

// élément trouvé : remplacement par prédécesseur

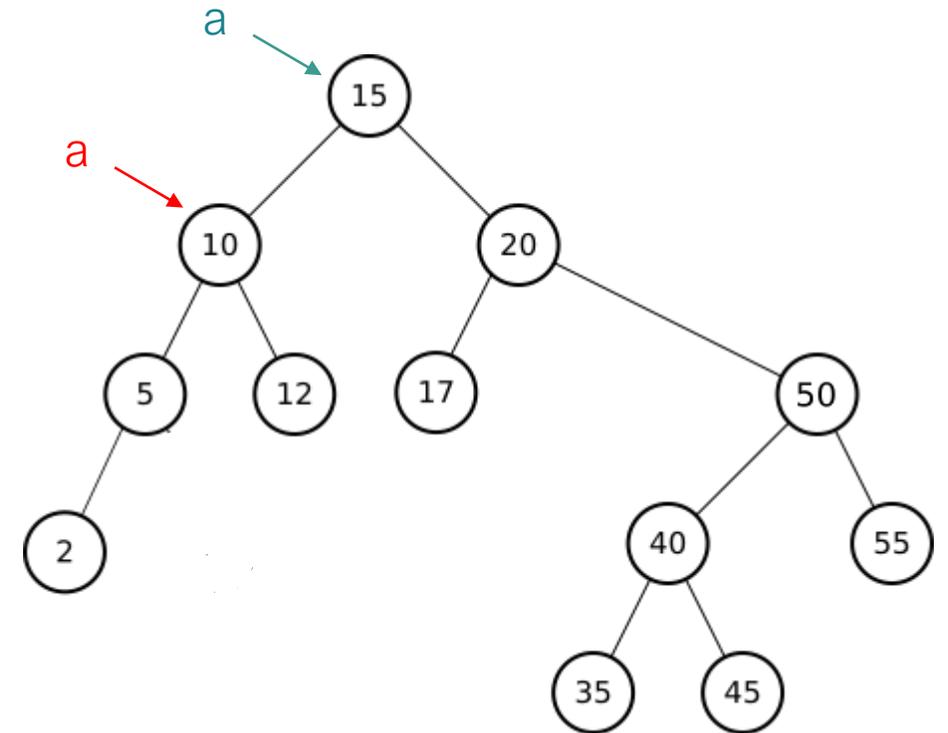
SINON

fg(a) ← suppMax(fg(a), adresse(element(a)))

FIN SI

→ RETOURNER a

FIN



ABR : opération de suppression

- Algorithme : suppression (a, 6)

FONCTION suppression(a: pointeur sur Arbre, e: element) :
pointeur sur Arbre

VARIABLE

tmp : ptr sur Arbre

DEBUT

// element non présent dans l'arbre

SI (a EST EGAL A NULL) Alors

RETOURNER a

// parcours récursif de l'arbre

SINON SI (e SUP. STRICT. A element(a))

fd(a) ← suppression(fd(a), e)

SINON SI (e INF. STRICT. A element(a))

➡ fg(a) ← suppression(fg(a), e)

// élément trouvé : remplacement par fils unique

SINON SI NON (existeFilsGauche(a))

tmp ← a

a ← fd(a)

libérer(tmp)

// élément trouvé : remplacement par prédécesseur

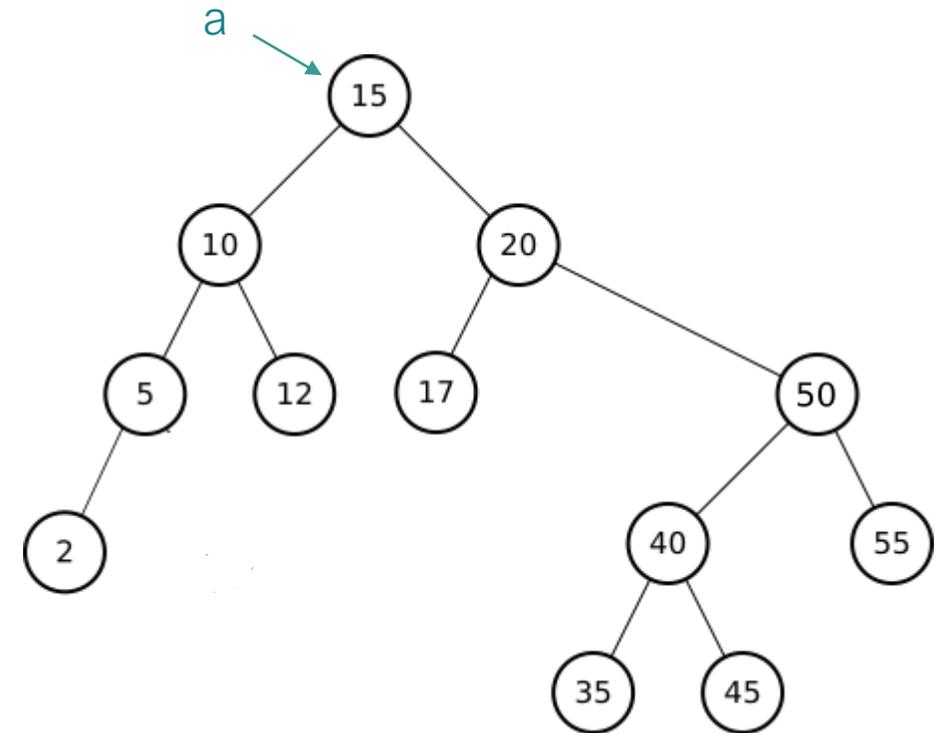
SINON

fg(a) ← suppMax(fg(a), adresse(element(a)))

FIN SI

RETOURNER a

FIN



ABR : opération de suppression

- Algorithme : suppression (a, 6)

FONCTION suppression(a: pointeur sur Arbre, e: element) :
pointeur sur Arbre

VARIABLE

tmp : ptr sur Arbre

DEBUT

// element non présent dans l'arbre

SI (a EST EGAL A NULL) Alors

RETOURNER a

// parcours récursif de l'arbre

SINON SI (e SUP. STRICT. A element(a))

fd(a) ← suppression(fd(a), e)

SINON SI (e INF. STRICT. A element(a))

fg(a) ← suppression(fg(a), e)

// élément trouvé : remplacement par fils unique

SINON SI NON (existeFilsGauche(a))

tmp ← a

a ← fd(a)

libérer(tmp)

// élément trouvé : remplacement par prédécesseur

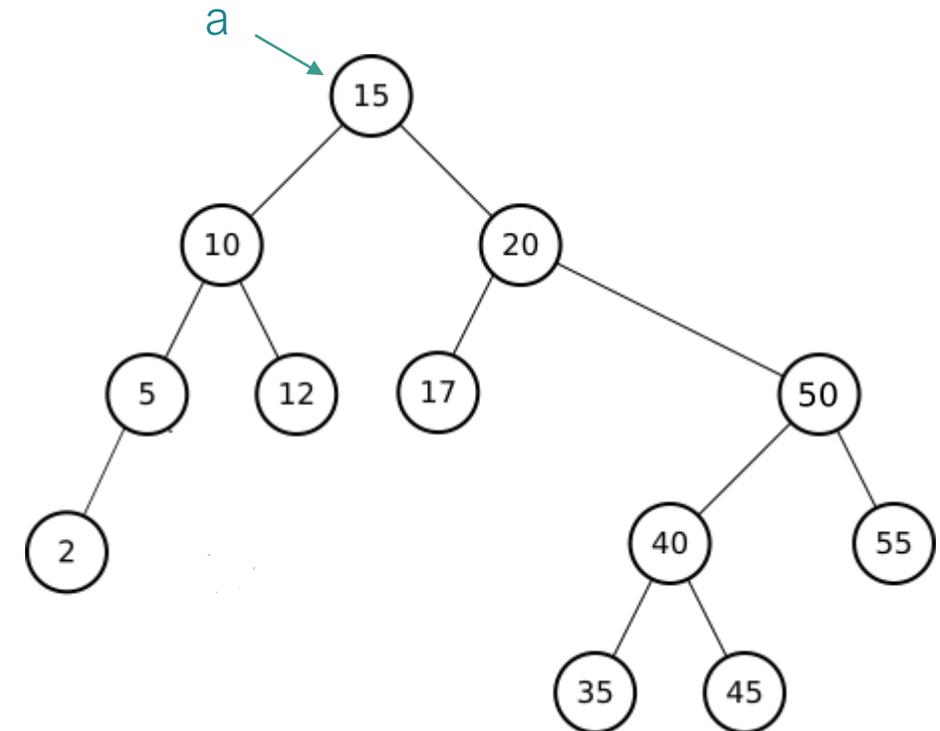
SINON

fg(a) ← suppMax(fg(a), adresse(element(a)))

FIN SI

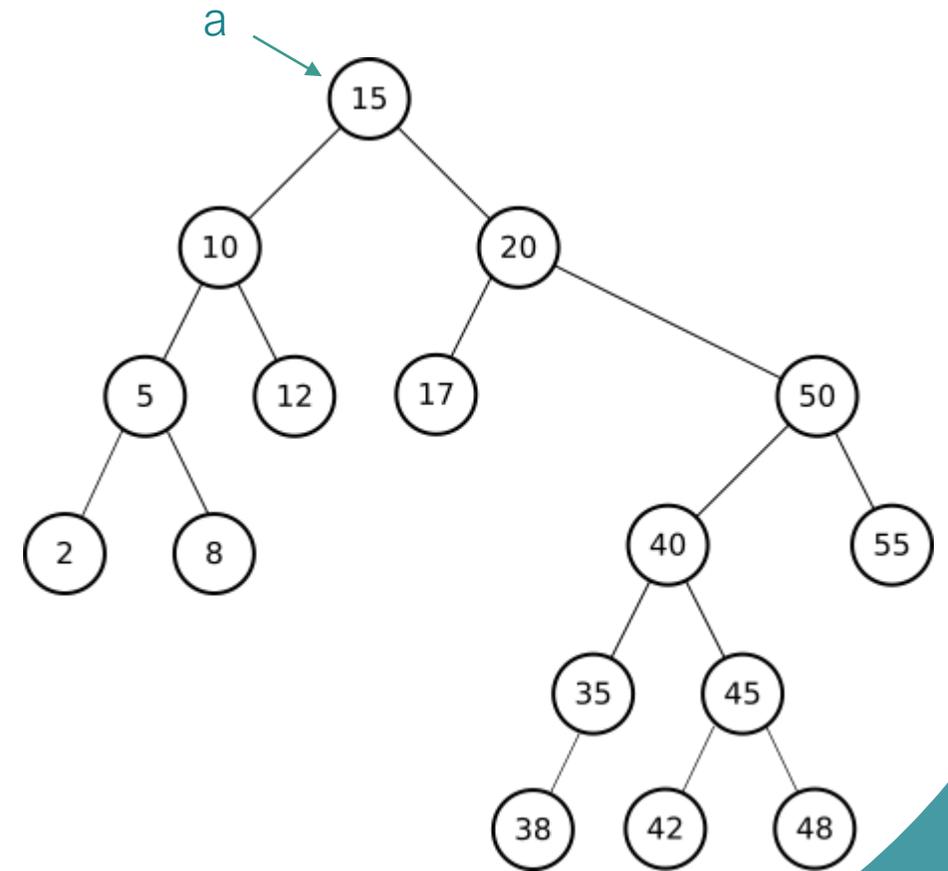
➡ RETOURNER a

FIN



ABR : opération de suppression

- Algorithme : suppression (a, 50)



ABR : opération de suppression

- Algorithme : suppression (a, 50)

FONCTION suppression(a: pointeur sur Arbre, e: element) :
pointeur sur Arbre

VARIABLE

tmp : ptr sur Arbre

DEBUT

// element non présent dans l'arbre

SI (a EST EGAL A NULL) Alors

RETOURNER a

// parcours récursif de l'arbre

SINON SI (e SUP. STRICT. A element(a))

fd(a) ← suppression(fd(a), e)

SINON SI (e INF. STRICT. A element(a))

fg(a) ← suppression(fg(a), e)

// élément trouvé : remplacement par fils unique

SINON SI NON (existeFilsGauche(a))

tmp ← a

a ← fd(a)

libérer(tmp)

// élément trouvé : remplacement par prédécesseur

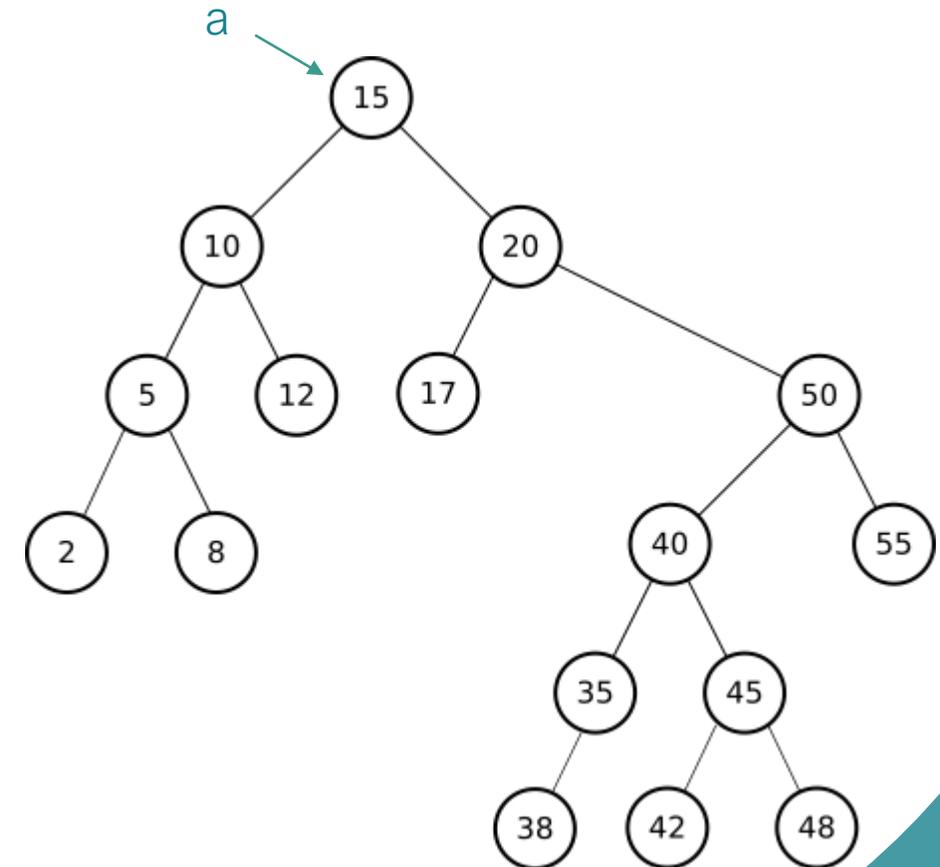
SINON

fg(a) ← suppMax(fg(a), adresse(element(a)))

FIN SI

RETOURNER a

FIN



ABR : opération de suppression

- Algorithme : suppression (a, 50)

FONCTION suppression(a: pointeur sur Arbre, e: element) :
pointeur sur Arbre

VARIABLE

tmp : ptr sur Arbre

DEBUT

// element non présent dans l'arbre

➔ SI (a EST EGAL A NULL) Alors

RETOURNER a

// parcours récursif de l'arbre

SINON SI (e SUP. STRICT. A element(a))

fd(a) ← suppression(fd(a), e)

SINON SI (e INF. STRICT. A element(a))

fg(a) ← suppression(fg(a), e)

// élément trouvé : remplacement par fils unique

SINON SI NON (existeFilsGauche(a))

tmp ← a

a ← fd(a)

libérer(tmp)

// élément trouvé : remplacement par prédécesseur

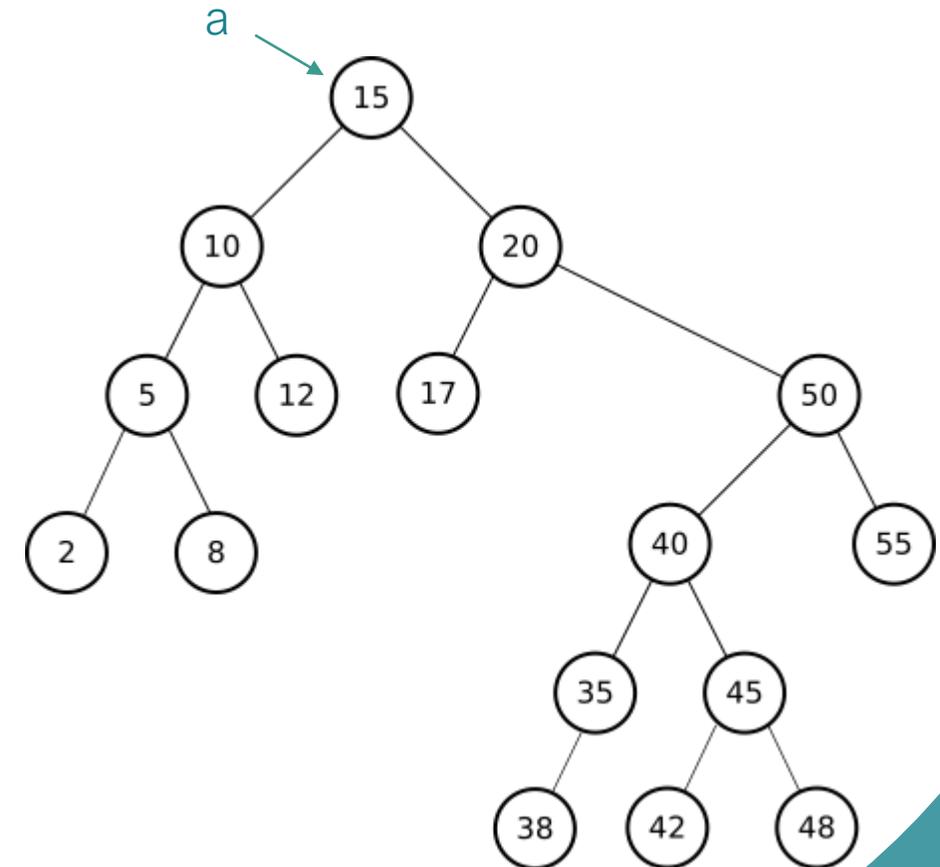
SINON

fg(a) ← suppMax(fg(a), adresse(element(a)))

FIN SI

RETOURNER a

FIN



ABR : opération de suppression

- Algorithme : suppression (a, 50)

FONCTION suppression(a: pointeur sur Arbre, e: element) :
pointeur sur Arbre

VARIABLE

tmp : ptr sur Arbre

DEBUT

// element non présent dans l'arbre

SI (a EST EGAL A NULL) Alors

RETOURNER a

// parcours récursif de l'arbre

➔ SINON SI (e SUP. STRICT. A element(a))

fd(a) ← suppression(fd(a), e)

SINON SI (e INF. STRICT. A element(a))

fg(a) ← suppression(fg(a), e)

// élément trouvé : remplacement par fils unique

SINON SI NON (existeFilsGauche(a))

tmp ← a

a ← fd(a)

libérer(tmp)

// élément trouvé : remplacement par prédécesseur

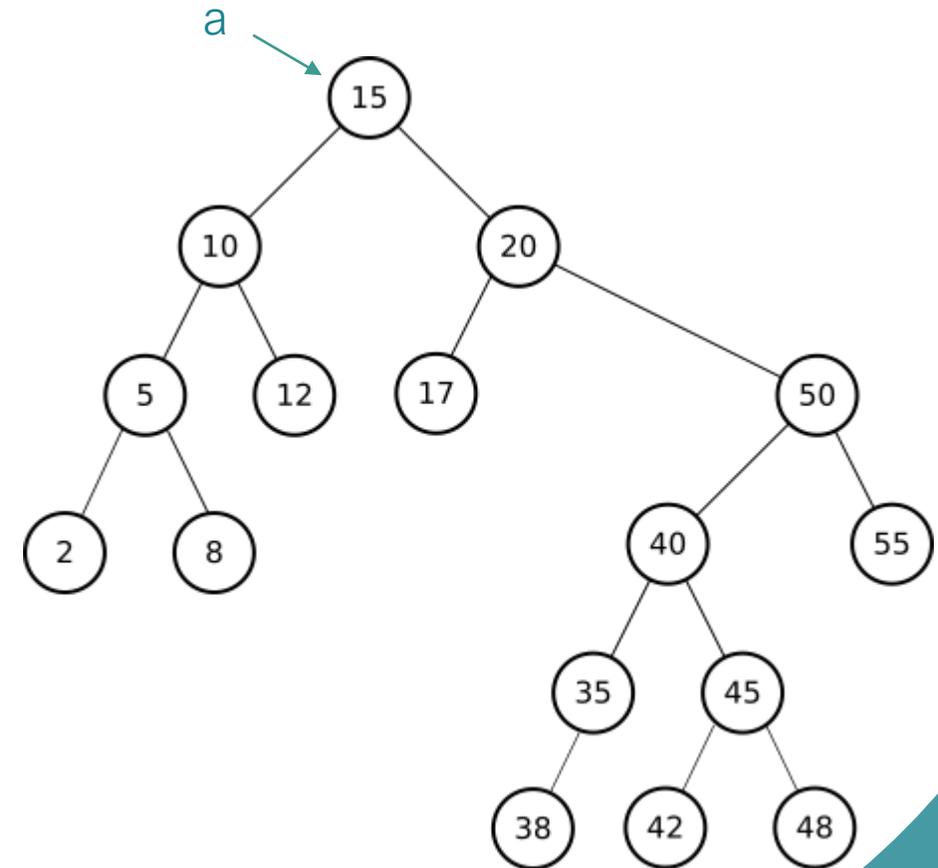
SINON

fg(a) ← suppMax(fg(a), adresse(element(a)))

FIN SI

RETOURNER a

FIN



ABR : opération de suppression

- Algorithme : suppression (a, 50)

FONCTION suppression(a: pointeur sur Arbre, e: element) :
pointeur sur Arbre

VARIABLE

tmp : ptr sur Arbre

DEBUT

// element non présent dans l'arbre

SI (a EST EGAL A NULL) Alors

RETOURNER a

// parcours récursif de l'arbre

SINON SI (e SUP. STRICT. A element(a))

➡ **fd(a)** ← suppression(fd(a), e)

SINON SI (e INF. STRICT. A element(a))

fg(a) ← suppression(fg(a), e)

// élément trouvé : remplacement par fils unique

SINON SI NON (existeFilsGauche(a))

tmp ← a

a ← **fd(a)**

libérer(tmp)

// élément trouvé : remplacement par prédécesseur

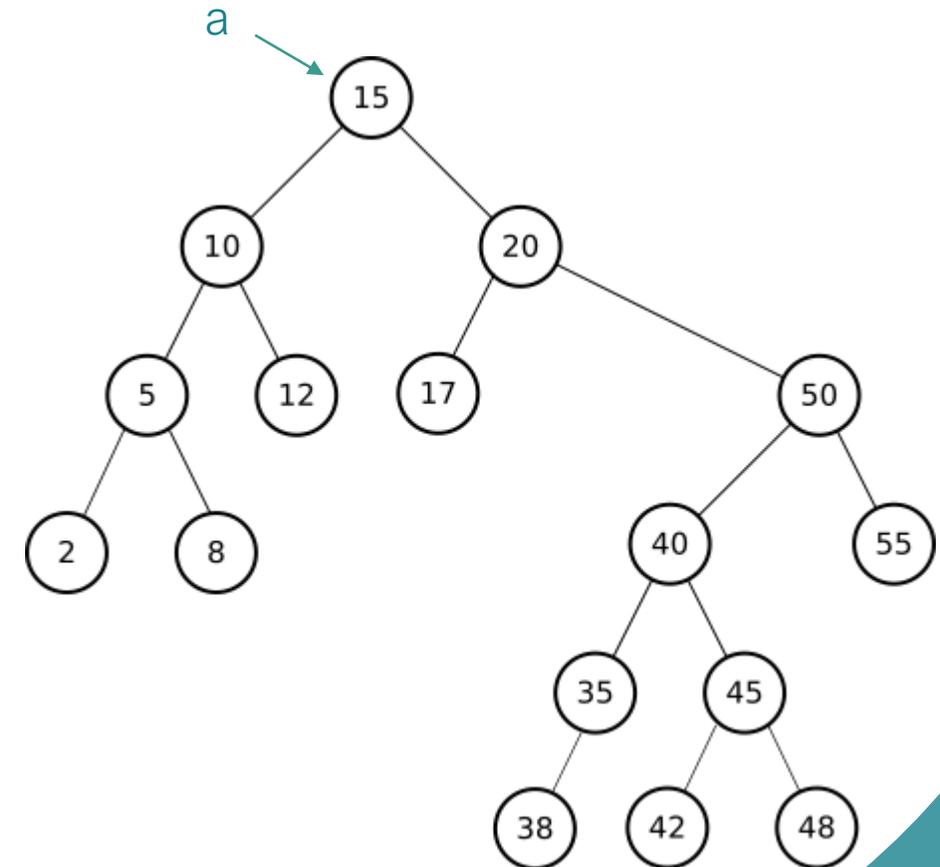
SINON

fg(a) ← supMax(fg(a), adresse(element(a)))

FIN SI

RETOURNER a

FIN



ABR : opération de suppression

- Algorithme : suppression (a, 50)

FONCTION suppression(a: pointeur sur Arbre, e: element) :
pointeur sur Arbre

VARIABLE

tmp : ptr sur Arbre

→ DEBUT

// element non présent dans l'arbre

SI (a EST EGAL A NULL) Alors

RETOURNER a

// parcours récursif de l'arbre

SINON SI (e SUP. STRICT. A element(a))

→ fd(a) ← suppression(fd(a), e)

SINON SI (e INF. STRICT. A element(a))

fg(a) ← suppression(fg(a), e)

// élément trouvé : remplacement par fils unique

SINON SI NON (existeFilsGauche(a))

tmp ← a

a ← fd(a)

libérer(tmp)

// élément trouvé : remplacement par prédécesseur

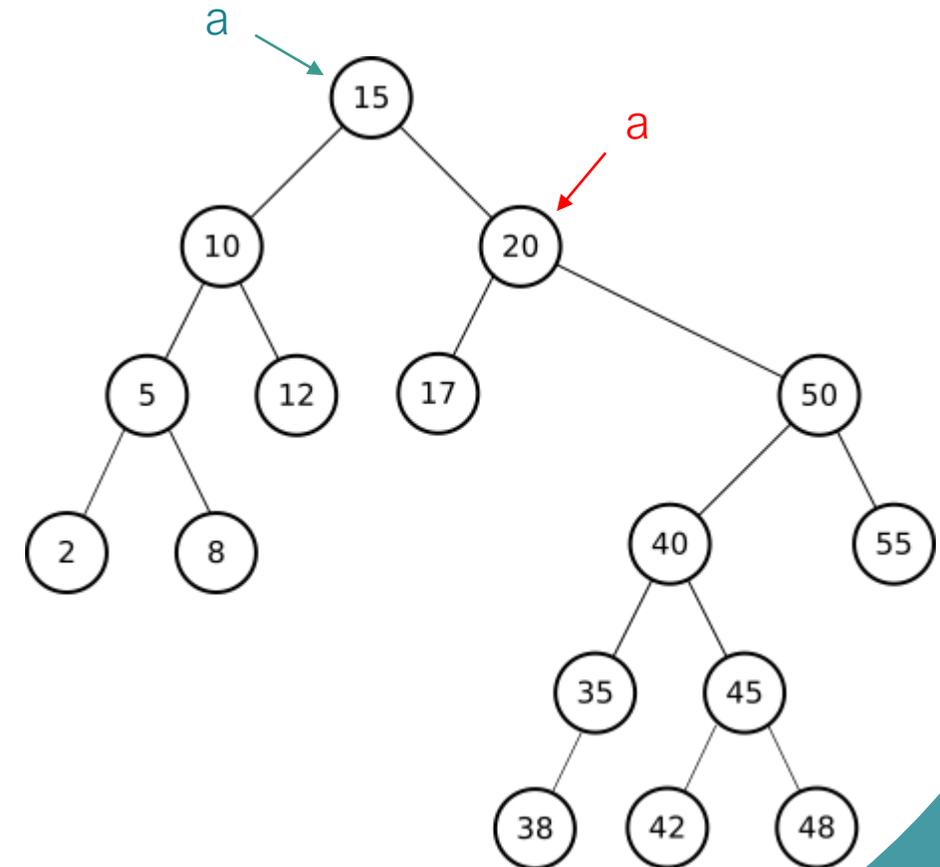
SINON

fg(a) ← suppMax(fg(a), adresse(element(a)))

FIN SI

RETOURNER a

FIN



ABR : opération de suppression

- Algorithme : suppression (a, 50)

FONCTION suppression(a: pointeur sur Arbre, e: element) :
pointeur sur Arbre

VARIABLE

tmp : ptr sur Arbre

DEBUT

// element non présent dans l'arbre

➔ SI (a EST EGAL A NULL) Alors

RETOURNER a

// parcours récursif de l'arbre

SINON SI (e SUP. STRICT. A element(a))

➔ fd(a) ← suppression(fd(a), e)

SINON SI (e INF. STRICT. A element(a))

fg(a) ← suppression(fg(a), e)

// élément trouvé : remplacement par fils unique

SINON SI NON (existeFilsGauche(a))

tmp ← a

a ← fd(a)

libérer(tmp)

// élément trouvé : remplacement par prédécesseur

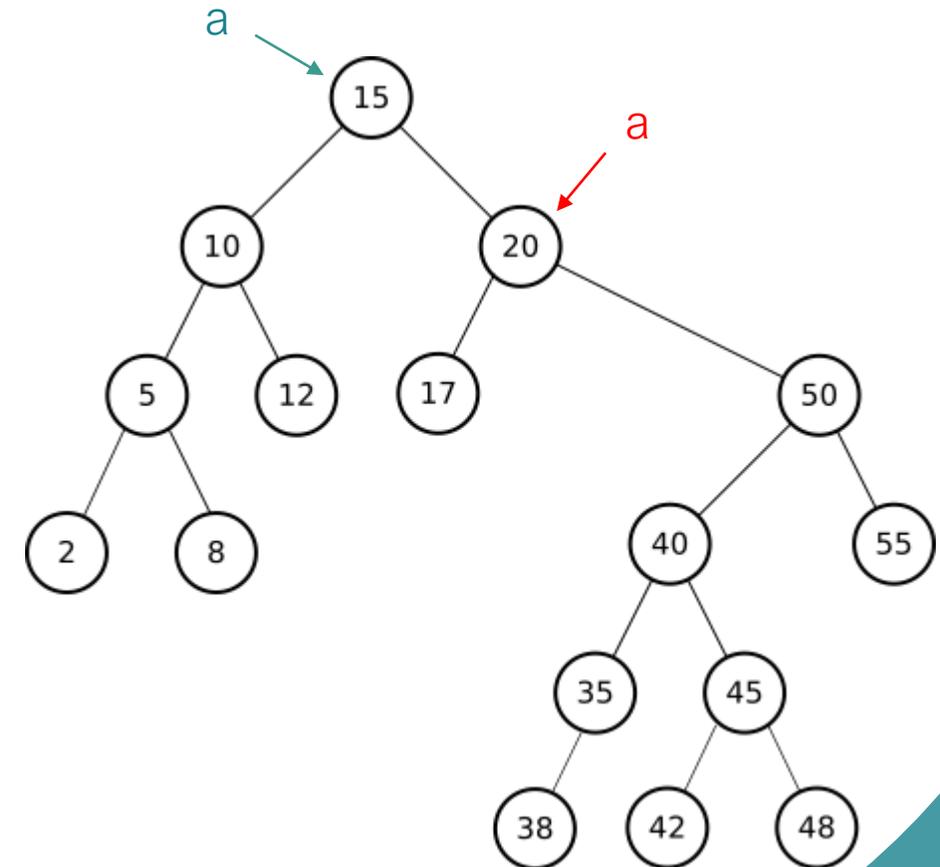
SINON

fg(a) ← suppMax(fg(a), adresse(element(a)))

FIN SI

RETOURNER a

FIN



ABR : opération de suppression

- Algorithme : suppression (a, 50)

FONCTION suppression(a: pointeur sur Arbre, e: element) :
pointeur sur Arbre

VARIABLE

tmp : ptr sur Arbre

DEBUT

// element non présent dans l'arbre

SI (a EST EGAL A NULL) Alors

RETOURNER a

// parcours récursif de l'arbre

➔ SINON SI (e SUP. STRICT. A element(a))

➔ fd(a) ← suppression(fd(a), e)

SINON SI (e INF. STRICT. A element(a))

fg(a) ← suppression(fg(a), e)

// élément trouvé : remplacement par fils unique

SINON SI NON (existeFilsGauche(a))

tmp ← a

a ← fd(a)

libérer(tmp)

// élément trouvé : remplacement par prédécesseur

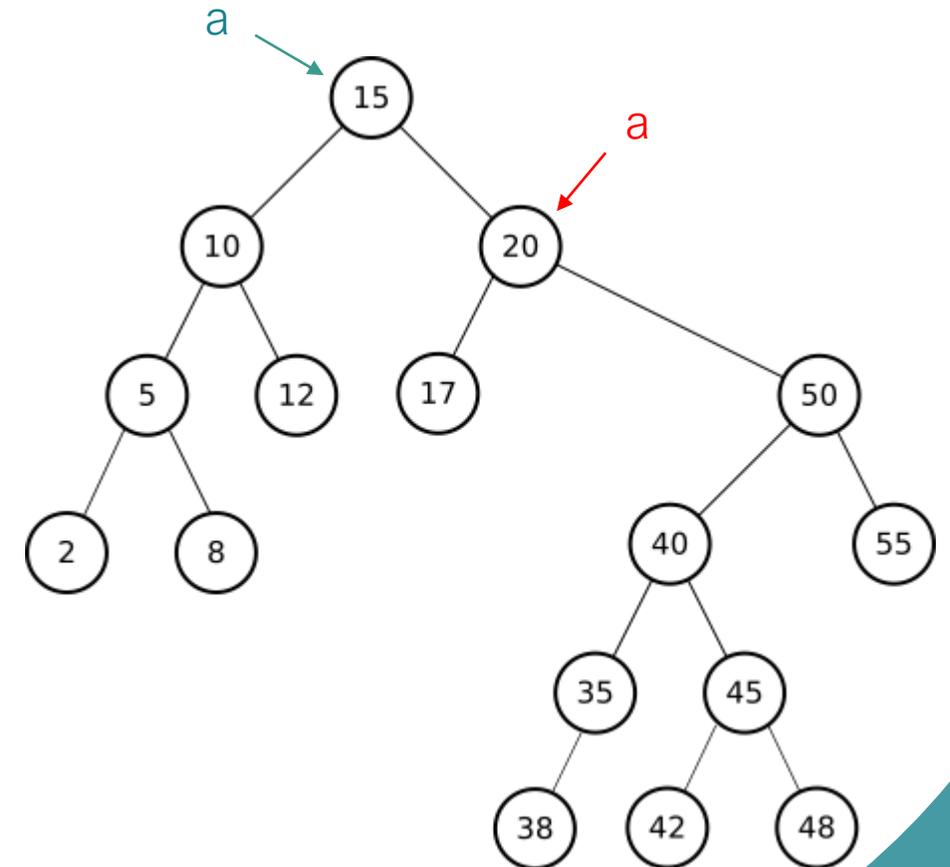
SINON

fg(a) ← suppMax(fg(a), adresse(element(a)))

FIN SI

RETOURNER a

FIN



ABR : opération de suppression

- Algorithme : suppression (a, 50)

FONCTION suppression(a: pointeur sur Arbre, e: element) :
pointeur sur Arbre

VARIABLE

tmp : ptr sur Arbre

DEBUT

// element non présent dans l'arbre

SI (a EST EGAL A NULL) Alors

RETOURNER a

// parcours récursif de l'arbre

SINON SI (e SUP. STRICT. A element(a))

➔ ➡ **fd(a)** ← suppression(fd(a), e)

SINON SI (e INF. STRICT. A element(a))

fg(a) ← suppression(fg(a), e)

// élément trouvé : remplacement par fils unique

SINON SI NON (existeFilsGauche(a))

tmp ← a

a ← **fd(a)**

libérer(tmp)

// élément trouvé : remplacement par prédécesseur

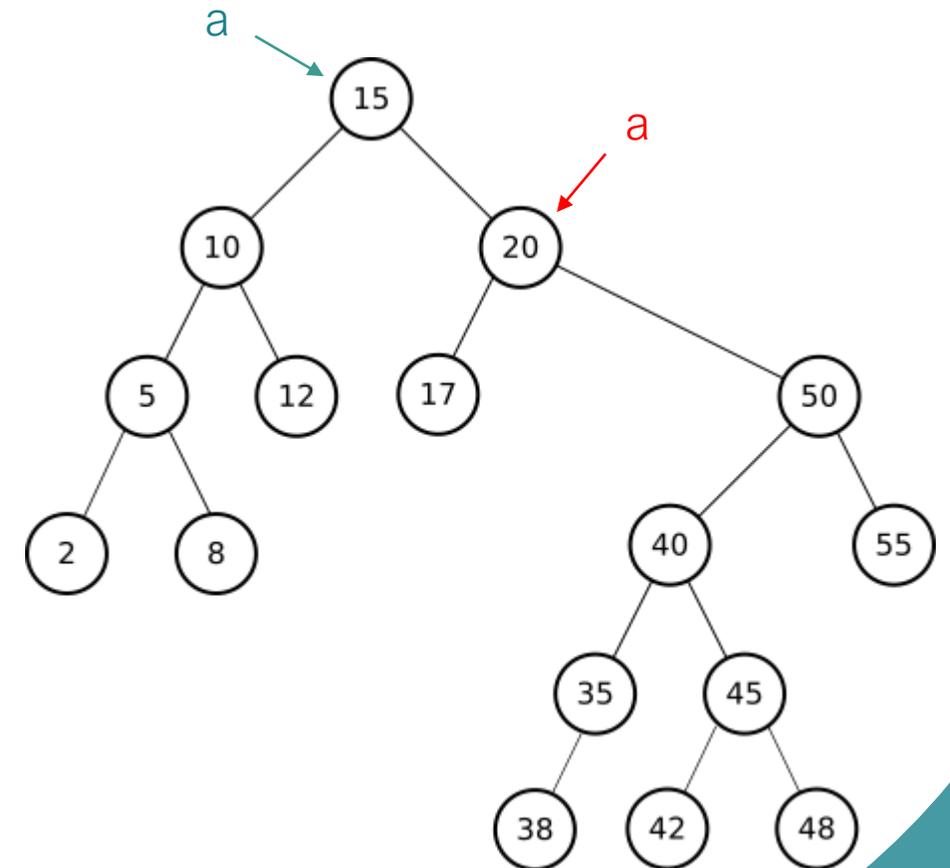
SINON

fg(a) ← supMax(fg(a), adresse(element(a)))

FIN SI

RETOURNER a

FIN



ABR : opération de suppression

- Algorithme : suppression (a, 50)

FONCTION suppression(a: pointeur sur Arbre, e: element) :
pointeur sur Arbre

VARIABLE

tmp : ptr sur Arbre

DEBUT

// element non présent dans l'arbre

SI (a EST EGAL A NULL) Alors

RETOURNER a

// parcours récursif de l'arbre

SINON SI (e SUP. STRICT. A element(a))

➡ ➡ **fd(a)** ← suppression(fd(a), e)

SINON SI (e INF. STRICT. A element(a))

fg(a) ← suppression(fg(a), e)

// élément trouvé : remplacement par fils unique

SINON SI NON (existeFilsGauche(a))

tmp ← a

a ← **fd(a)**

libérer(tmp)

// élément trouvé : remplacement par prédécesseur

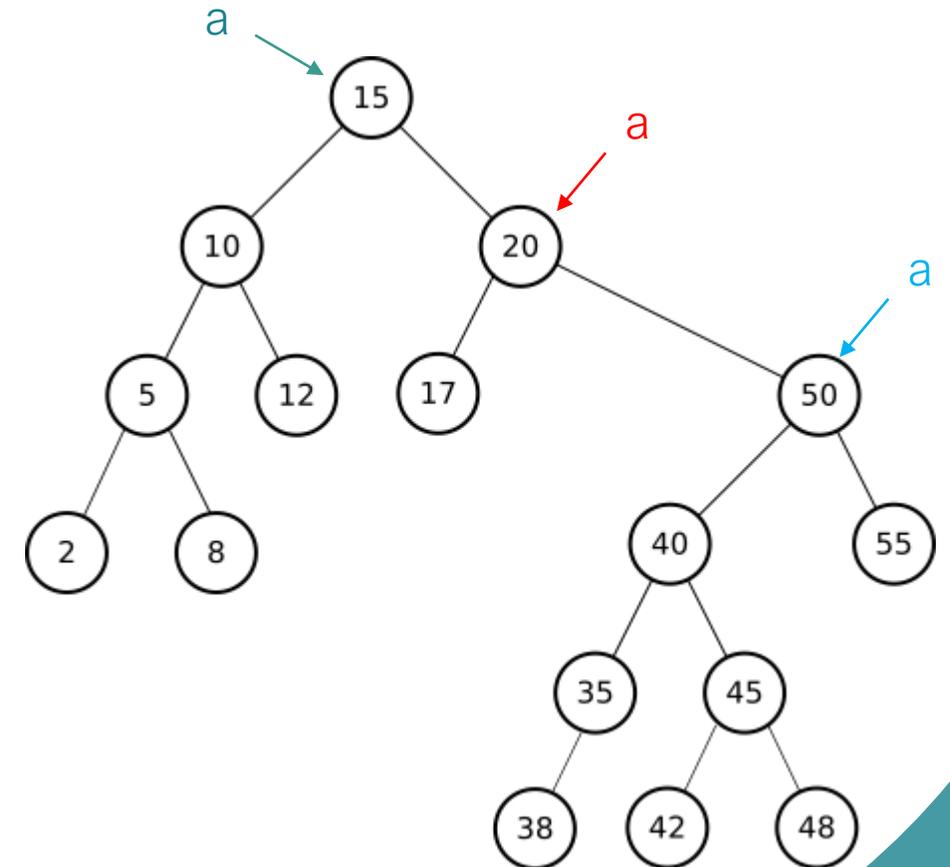
SINON

fg(a) ← supMax(fg(a), adresse(element(a)))

FIN SI

RETOURNER a

FIN



ABR : opération de suppression

- Algorithme : suppression (a, 50)

FONCTION suppression(a: pointeur sur Arbre, e: element) :
pointeur sur Arbre

VARIABLE

tmp : ptr sur Arbre

DEBUT

// element non présent dans l'arbre

➔ SI (a EST EGAL A NULL) Alors

RETOURNER a

// parcours récursif de l'arbre

SINON SI (e SUP. STRICT. A element(a))

➔➔ **fd(a)** ← suppression(fd(a), e)

SINON SI (e INF. STRICT. A element(a))

fg(a) ← suppression(fg(a), e)

// élément trouvé : remplacement par fils unique

SINON SI NON (existeFilsGauche(a))

tmp ← a

a ← **fd(a)**

libérer(tmp)

// élément trouvé : remplacement par prédécesseur

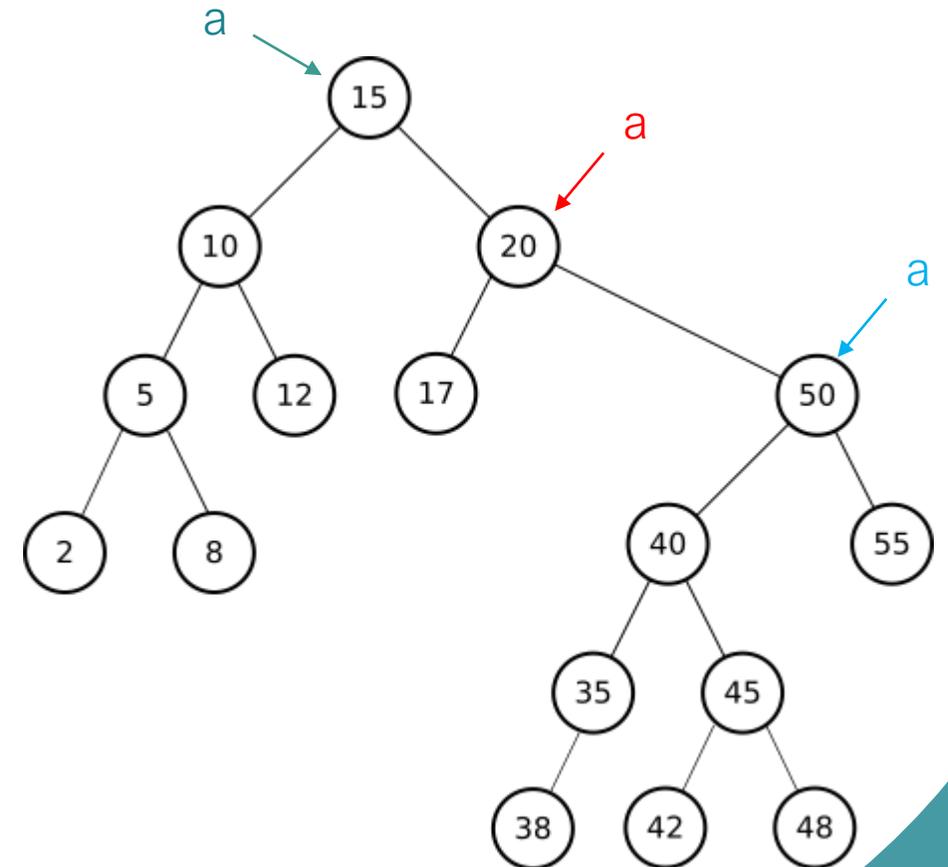
SINON

fg(a) ← supMax(fg(a), adresse(element(a)))

FIN SI

RETOURNER a

FIN



ABR : opération de suppression

- Algorithme : suppression (a, 50)

FONCTION suppression(a: pointeur sur Arbre, e: element) :
pointeur sur Arbre

VARIABLE

tmp : ptr sur Arbre

DEBUT

// element non présent dans l'arbre

SI (a EST EGAL A NULL) Alors

RETOURNER a

// parcours récursif de l'arbre

➡ SINON SI (e SUP. STRICT. A element(a))

➡➡ fd(a) ← suppression(fd(a), e)

SINON SI (e INF. STRICT. A element(a))

fg(a) ← suppression(fg(a), e)

// élément trouvé : remplacement par fils unique

SINON SI NON (existeFilsGauche(a))

tmp ← a

a ← fd(a)

libérer(tmp)

// élément trouvé : remplacement par prédécesseur

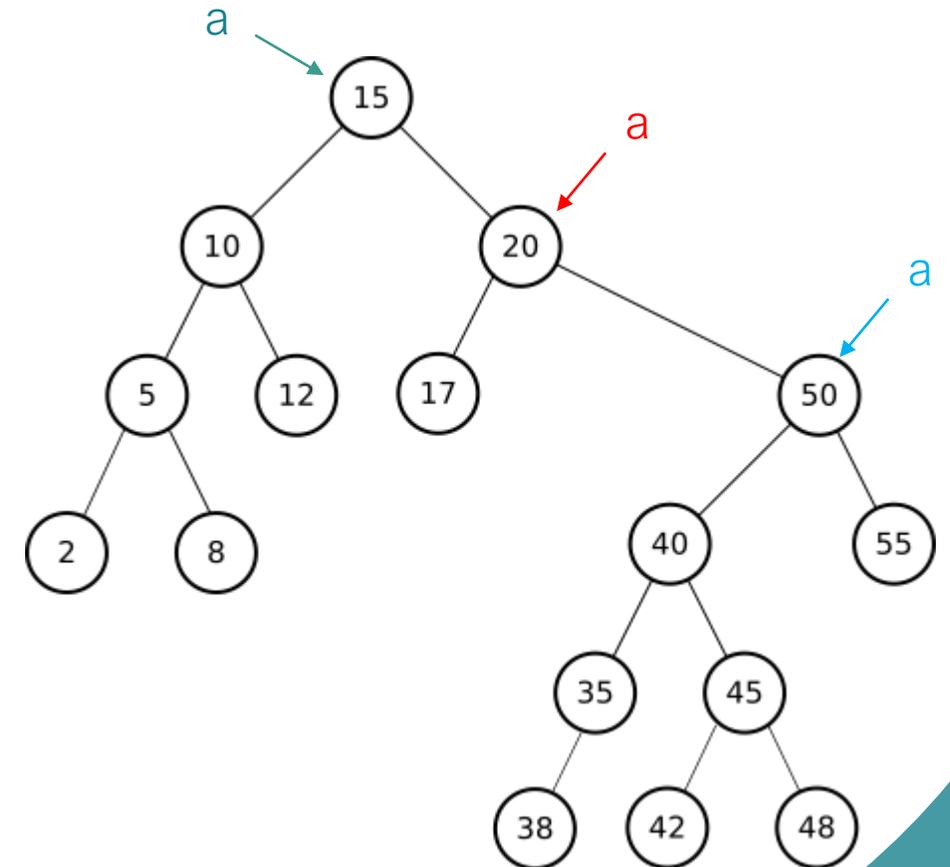
SINON

fg(a) ← suppMax(fg(a), adresse(element(a)))

FIN SI

RETOURNER a

FIN



ABR : opération de suppression

- Algorithme : suppression (a, 50)

FONCTION suppression(a: pointeur sur Arbre, e: element) :
pointeur sur Arbre

VARIABLE

tmp : ptr sur Arbre

DEBUT

// element non présent dans l'arbre

SI (a EST EGAL A NULL) Alors

RETOURNER a

// parcours récursif de l'arbre

SINON SI (e SUP. STRICT. A element(a))

➡ fd(a) ← suppression(fd(a), e)

➡ SINON SI (e INF. STRICT. A element(a))

fg(a) ← suppression(fg(a), e)

// élément trouvé : remplacement par fils unique

SINON SI NON (existeFilsGauche(a))

tmp ← a

a ← fd(a)

libérer(tmp)

// élément trouvé : remplacement par prédécesseur

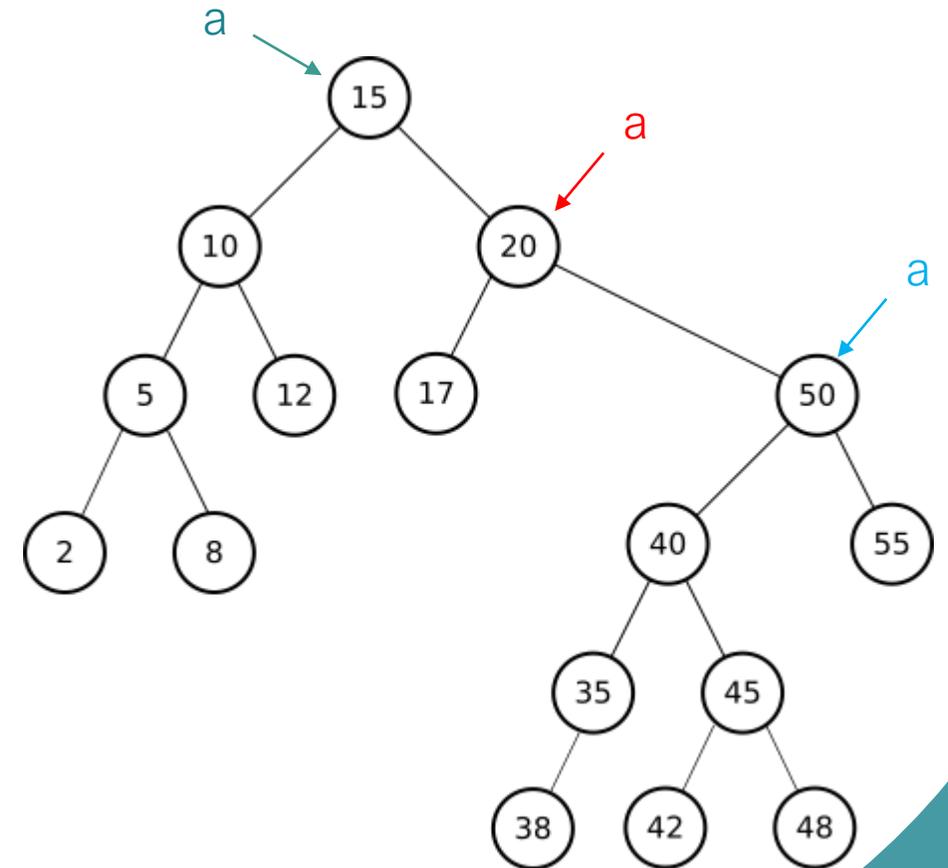
SINON

fg(a) ← suppMax(fg(a), adresse(element(a)))

FIN SI

RETOURNER a

FIN



ABR : opération de suppression

- Algorithme : suppression (a, 50)

FONCTION suppression(a: pointeur sur Arbre, e: element) :
pointeur sur Arbre

VARIABLE

tmp : ptr sur Arbre

DEBUT

// element non présent dans l'arbre

SI (a EST EGAL A NULL) Alors

RETOURNER a

// parcours récursif de l'arbre

SINON SI (e SUP. STRICT. A element(a))

➡➡ **fd(a)** ← suppression(fd(a), e)

SINON SI (e INF. STRICT. A element(a))

fg(a) ← suppression(fg(a), e)

// élément trouvé : remplacement par fils unique

➡ SINON SI NON (existeFilsGauche(a))

tmp ← a

a ← **fd(a)**

libérer(tmp)

// élément trouvé : remplacement par prédécesseur

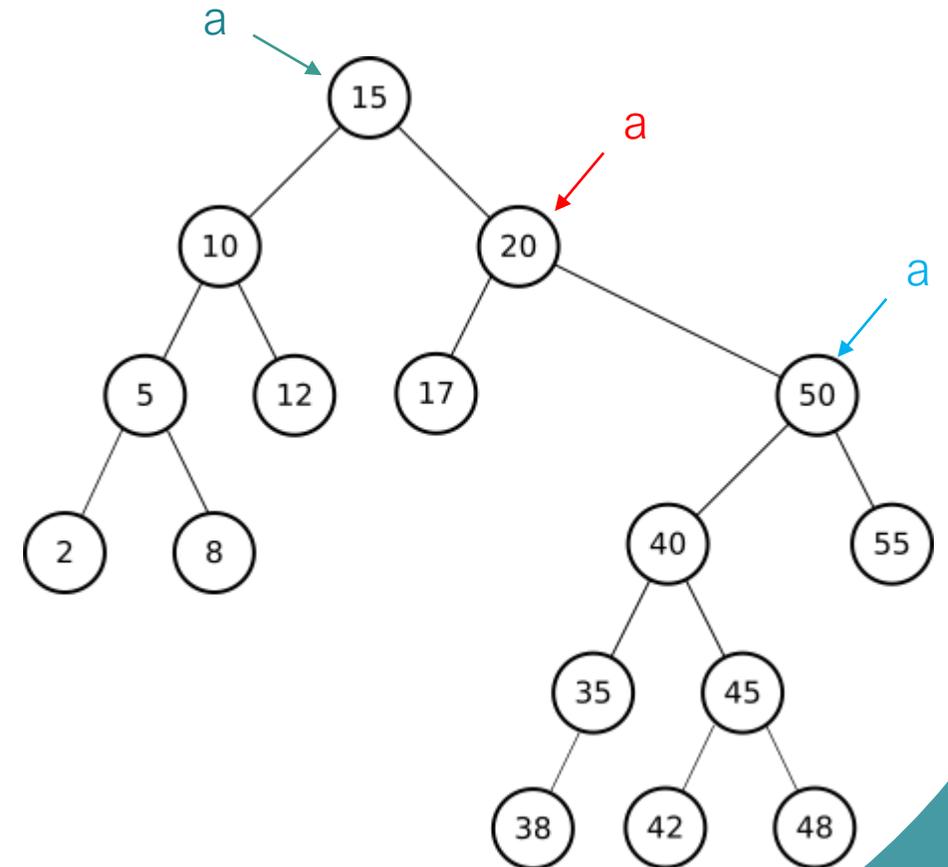
SINON

fg(a) ← supMax(fg(a), adresse(element(a)))

FIN SI

RETOURNER a

FIN



ABR : opération de suppression

- Algorithme : suppression (a, 50)

FONCTION suppression(a: pointeur sur Arbre, e: element) :
pointeur sur Arbre

VARIABLE

tmp : ptr sur Arbre

DEBUT

// element non présent dans l'arbre

SI (a EST EGAL A NULL) Alors

RETOURNER a

// parcours récursif de l'arbre

SINON SI (e SUP. STRICT. A element(a))

➡➡ **fd(a)** ← suppression(fd(a), e)

SINON SI (e INF. STRICT. A element(a))

fg(a) ← suppression(fg(a), e)

// élément trouvé : remplacement par fils unique

SINON SI NON (existeFilsGauche(a))

tmp ← a

a ← **fd(a)**

libérer(tmp)

// élément trouvé : remplacement par prédécesseur

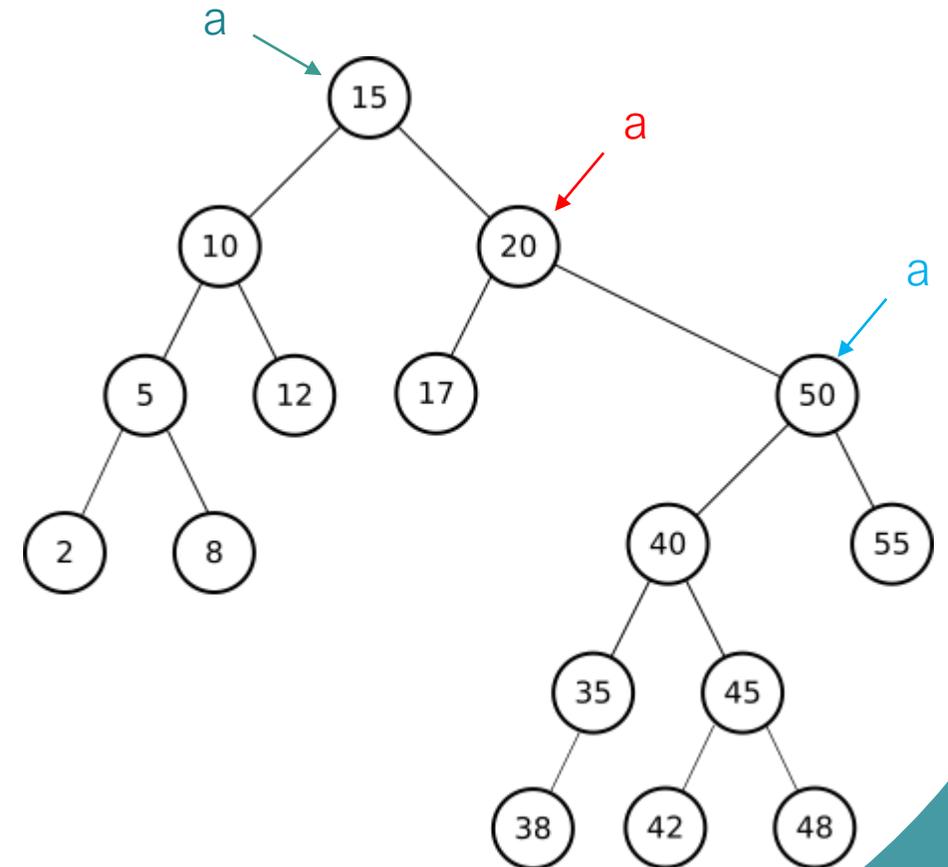
SINON

➡➡ **fg(a)** ← supMax(fg(a), adresse(element(a)))

FIN SI

RETOURNER a

FIN

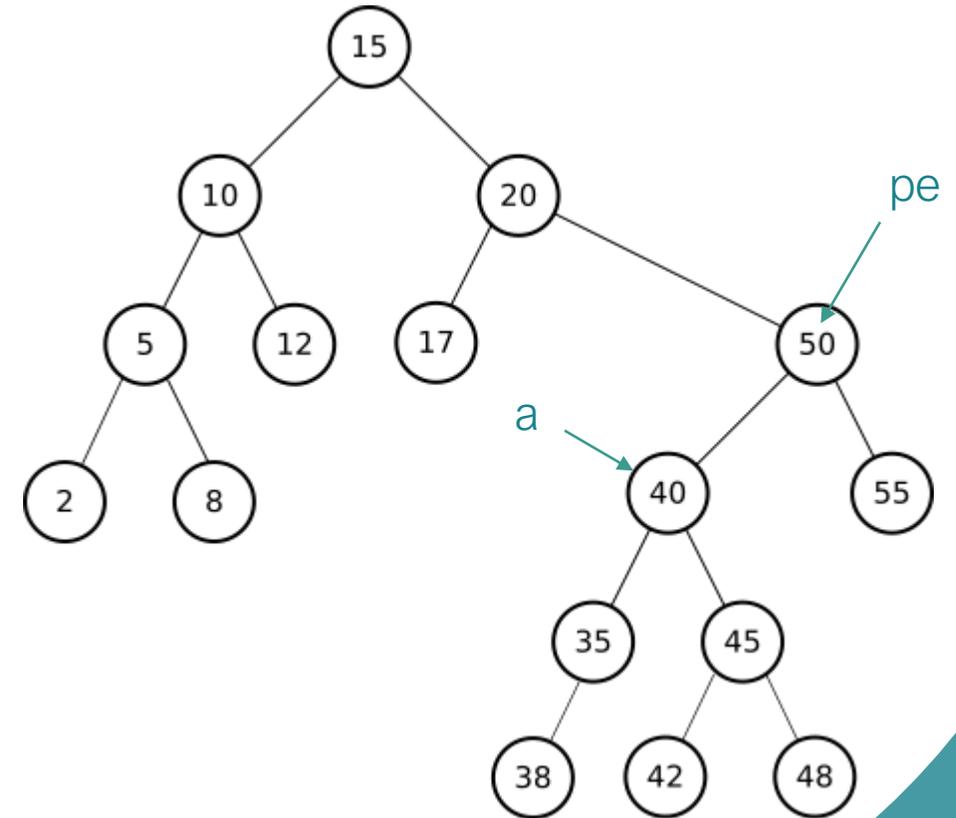


ABR : opération de suppression

- Algorithme : suppression (a,50)

```

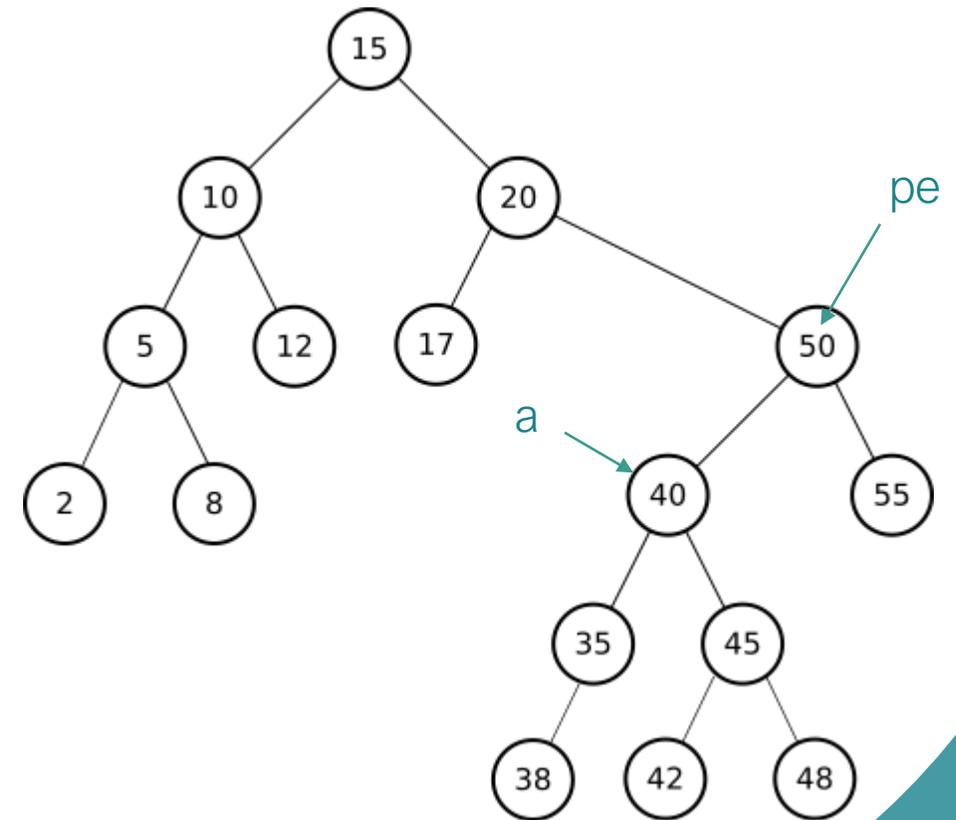
FONCTION suppMax(a: arbre, pe: Ptr sur Element) :
ptr sur Arbre
VARIABLE
    tmp : ptr sur Arbre
→ DEBUT
    // on rappelle la fonction avec le fils droit
    SI (existeFilsDroit(a)) Alors
        fd(a) ← suppMax(fd(a), pe)
    // Si plus de fils droit, on a le successeur
    SINON
        *pe ← element(a)
        tmp ← a
        a ← fg(a)
        libérer(tmp)
    FIN SI
    RETOURNER a
FIN
```



ABR : opération de suppression

- Algorithme : suppression (a,50)

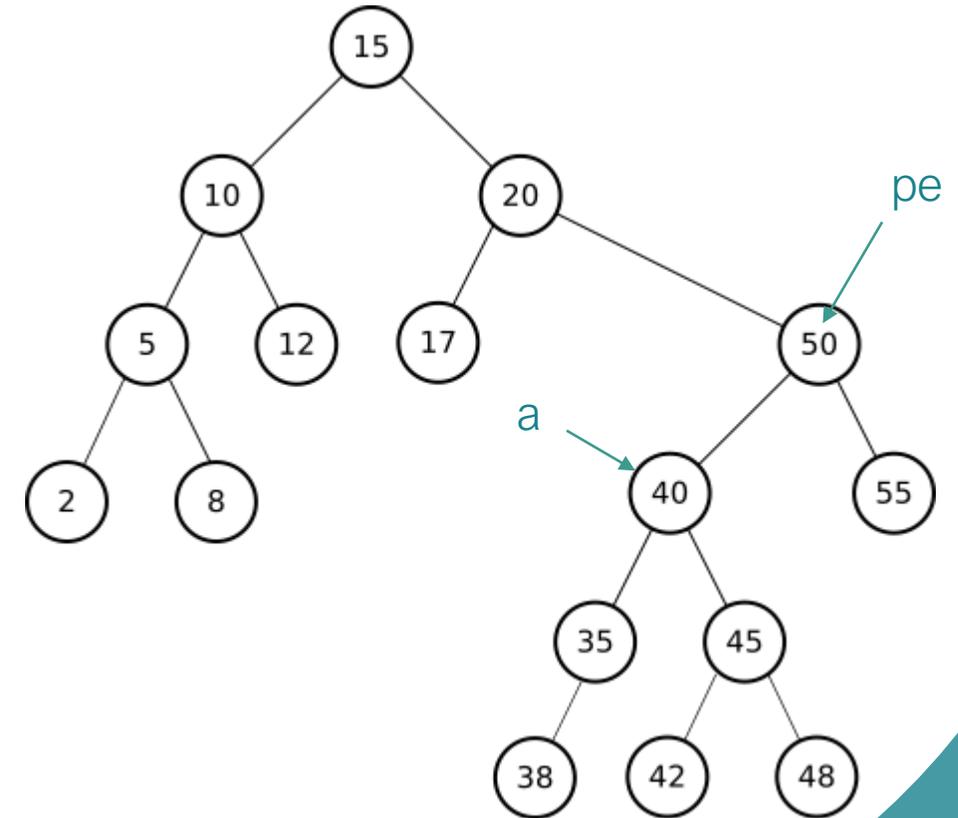
```
FONCTION suppMax(a: arbre, pe: Ptr sur Element) :  
ptr sur Arbre  
VARIABLE  
    tmp : ptr sur Arbre  
DEBUT  
    // on rappelle la fonction avec le fils droit  
    → SI (existeFilsDroit(a)) Alors  
        fd(a) ← suppMax(fd(a), pe)  
    // Si plus de fils droit, on a le successeur  
    SINON  
        *pe ← element(a)  
        tmp ← a  
        a ← fg(a)  
        libérer(tmp)  
    FIN SI  
    RETOURNER a  
FIN
```



ABR : opération de suppression

- Algorithme : suppression (a,50)

```
FONCTION suppMax(a: arbre, pe: Ptr sur Element) :  
ptr sur Arbre  
VARIABLE  
    tmp : ptr sur Arbre  
DEBUT  
    // on rappelle la fonction avec le fils droit  
    SI (existeFilsDroit(a)) Alors  
        ➡ fd(a) ← suppMax(fd(a), pe)  
    // Si plus de fils droit, on a le successeur  
    SINON  
        *pe ← element(a)  
        tmp ← a  
        a ← fg(a)  
        libérer(tmp)  
    FIN SI  
    RETOURNER a  
FIN
```



ABR : opération de suppression

- Algorithme : suppression (a,50)

```
FONCTION suppMax(a: arbre, pe: Ptr sur Element) :  
ptr sur Arbre
```

```
VARIABLE
```

```
    tmp : ptr sur Arbre
```

```
→ DEBUT
```

```
    // on rappelle la fonction avec le fils droit  
    SI (existeFilsDroit(a)) Alors
```

```
        → fd(a) ← suppMax(fd(a), pe)
```

```
    // Si plus de fils droit, on a le successeur
```

```
SINON
```

```
    *pe ← element(a)
```

```
    tmp ← a
```

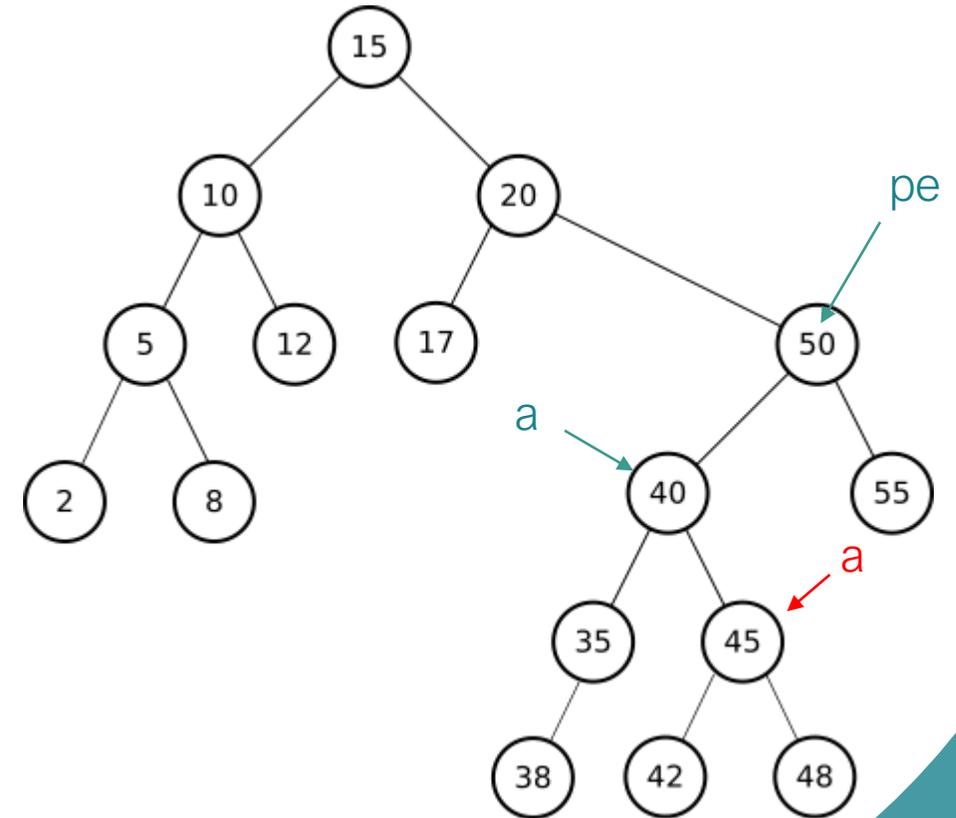
```
    a ← fg(a)
```

```
    libérer(tmp)
```

```
FIN SI
```

```
RETOURNER a
```

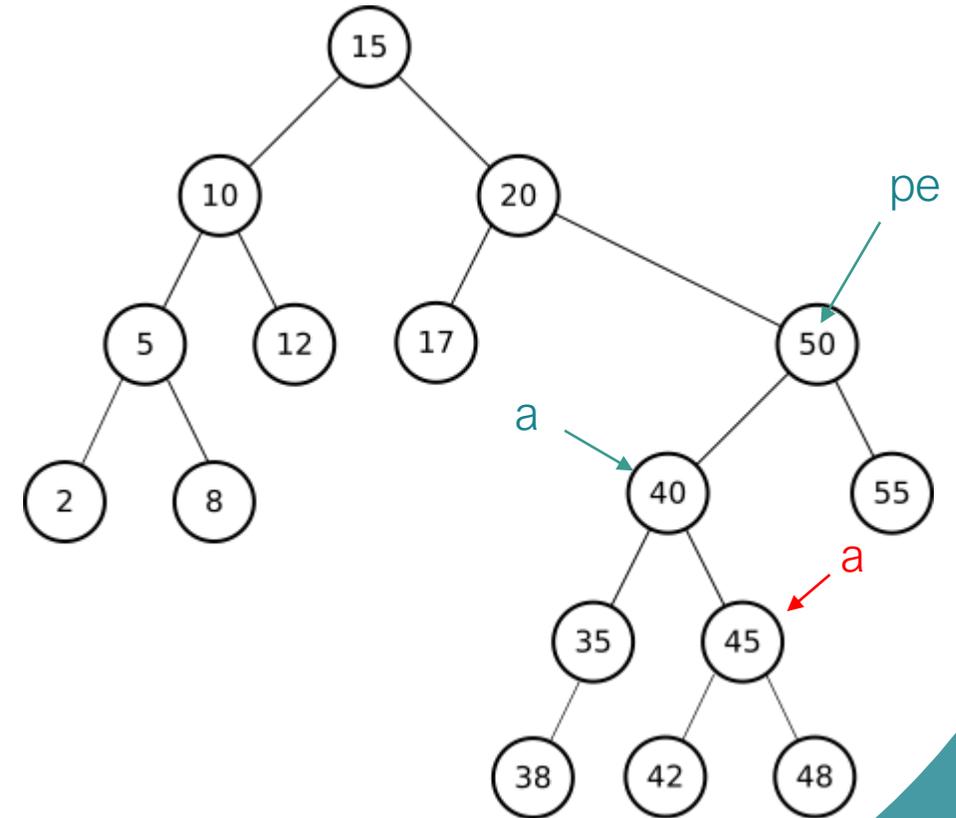
```
FIN
```



ABR : opération de suppression

- Algorithme : suppression (a,50)

```
FONCTION suppMax(a: arbre, pe: Ptr sur Element) :  
ptr sur Arbre  
VARIABLE  
    tmp : ptr sur Arbre  
DEBUT  
    // on rappelle la fonction avec le fils droit  
    → SI (existeFilsDroit(a)) Alors  
        → fd(a) ← suppMax(fd(a), pe)  
        // Si plus de fils droit, on a le successeur  
    SINON  
        *pe ← element(a)  
        tmp ← a  
        a ← fg(a)  
        libérer(tmp)  
    FIN SI  
    RETOURNER a  
FIN
```

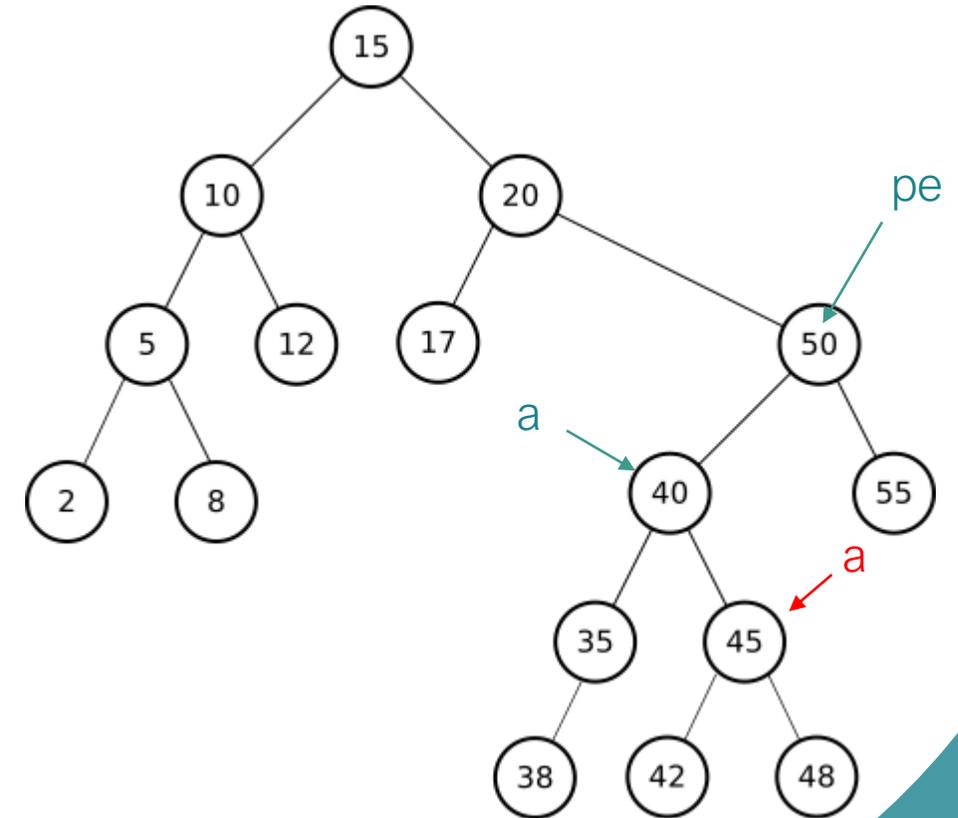


ABR : opération de suppression

- Algorithme : suppression (a,50)

```

FONCTION suppMax(a: arbre, pe: Ptr sur Element) :
ptr sur Arbre
VARIABLE
    tmp : ptr sur Arbre
DEBUT
    // on rappelle la fonction avec le fils droit
    SI (existeFilsDroit(a)) Alors
    ➔ ➔ fd(a) ← suppMax(fd(a), pe)
    // Si plus de fils droit, on a le successeur
    SINON
        *pe ← element(a)
        tmp ← a
        a ← fg(a)
        libérer(tmp)
    FIN SI
    RETOURNER a
FIN
```



ABR : opération de suppression

- Algorithme : suppression (a,50)

```
FONCTION suppMax(a: arbre, pe: Ptr sur Element) :  
ptr sur Arbre
```

```
VARIABLE
```

```
    tmp : ptr sur Arbre
```

```
➔ DEBUT
```

```
    // on rappelle la fonction avec le fils droit  
    SI (existeFilsDroit(a)) Alors
```

```
➔➔ fd(a) ← suppMax(fd(a), pe)
```

```
    // Si plus de fils droit, on a le successeur
```

```
SINON
```

```
    *pe ← element(a)
```

```
    tmp ← a
```

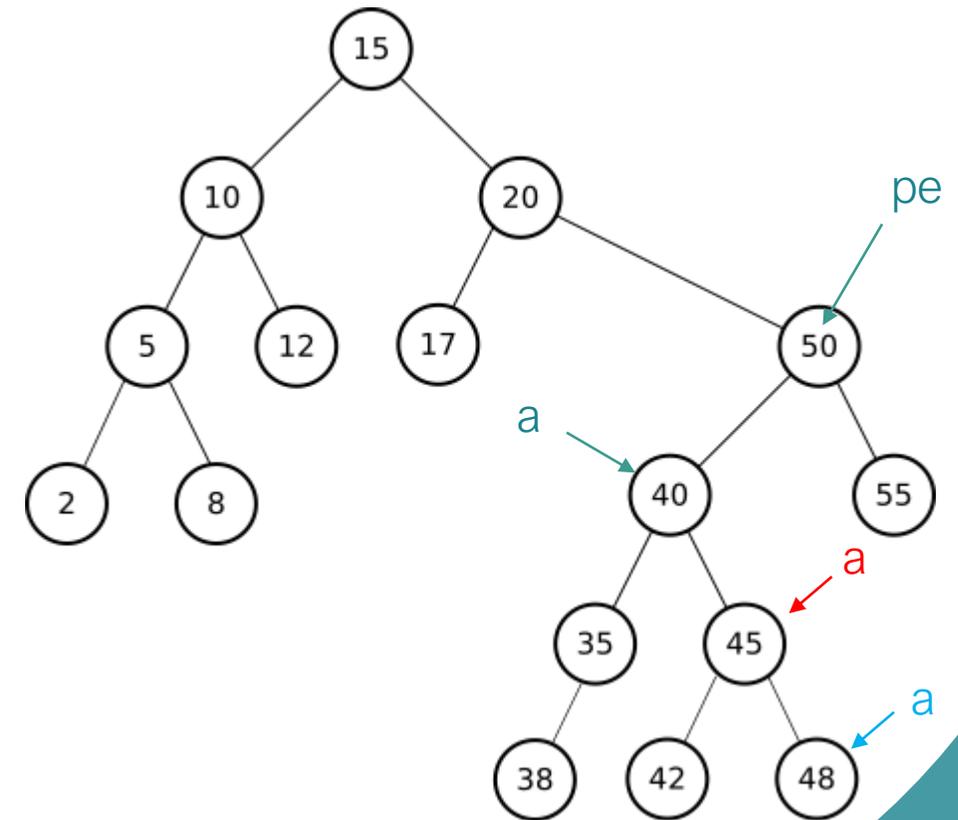
```
    a ← fg(a)
```

```
    libérer(tmp)
```

```
FIN SI
```

```
RETOURNER a
```

```
FIN
```

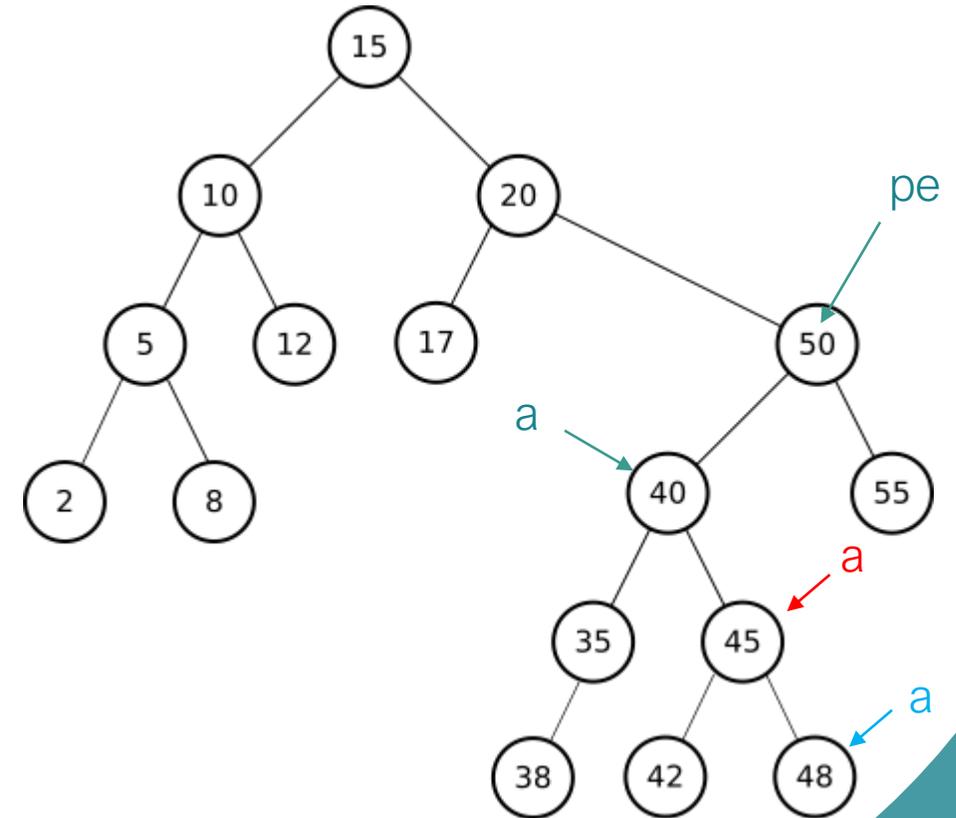


ABR : opération de suppression

- Algorithme : suppression (a,50)

```

FONCTION suppMax(a: arbre, pe: Ptr sur Element) :
ptr sur Arbre
VARIABLE
    tmp : ptr sur Arbre
DEBUT
    // on rappelle la fonction avec le fils droit
    → SI (existeFilsDroit(a)) Alors
        → fd(a) ← suppMax(fd(a), pe)
        // Si plus de fils droit, on a le successeur
    SINON
        *pe ← element(a)
        tmp ← a
        a ← fg(a)
        libérer(tmp)
    FIN SI
    RETOURNER a
FIN
```

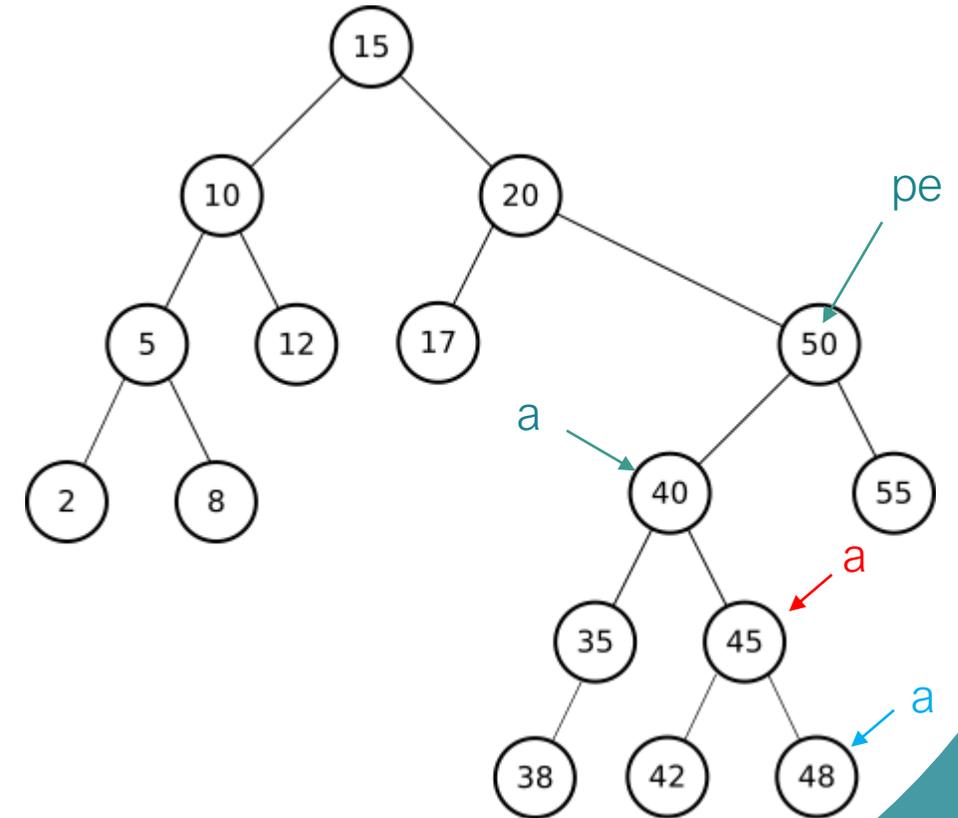


ABR : opération de suppression

- Algorithme : suppression (a,50)

```

FONCTION suppMax(a: arbre, pe: Ptr sur Element) :
ptr sur Arbre
VARIABLE
    tmp : ptr sur Arbre
DEBUT
    // on rappelle la fonction avec le fils droit
    SI (existeFilsDroit(a)) Alors
    ➔ ➔ fd(a) ← suppMax(fd(a), pe)
    // Si plus de fils droit, on a le successeur
    SINON
    ➔ *pe ← element(a)
    tmp ← a
    a ← fg(a)
    libérer(tmp)
    FIN SI
    RETOURNER a
FIN
```

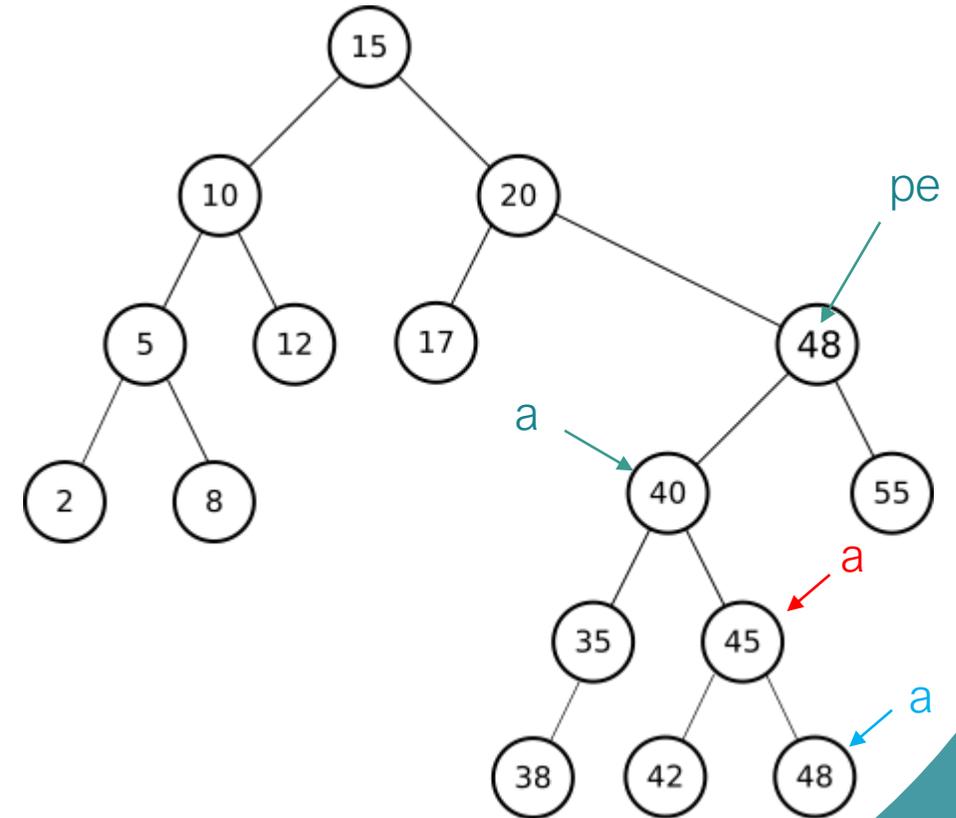


ABR : opération de suppression

- Algorithme : suppression (a,50)

```

FONCTION suppMax(a: arbre, pe: Ptr sur Element) :
ptr sur Arbre
VARIABLE
    tmp : ptr sur Arbre
DEBUT
    // on rappelle la fonction avec le fils droit
    SI (existeFilsDroit(a)) Alors
    ➔➔ fd(a) ← suppMax(fd(a), pe)
    // Si plus de fils droit, on a le successeur
    SINON
        *pe ← element(a)
        ➔ tmp ← a
        a ← fg(a)
        libérer(tmp)
    FIN SI
    RETOURNER a
FIN
```

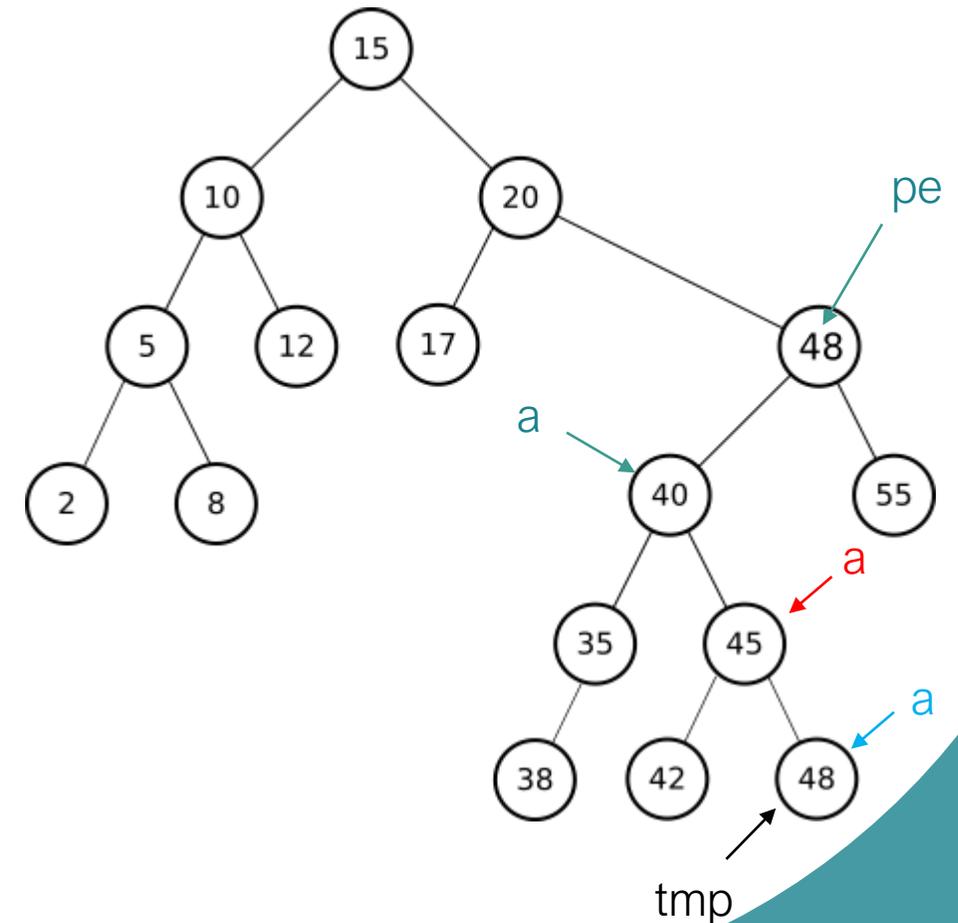


ABR : opération de suppression

- Algorithme : suppression (a,50)

```

FONCTION suppMax(a: arbre, pe: Ptr sur Element) :
ptr sur Arbre
VARIABLE
    tmp : ptr sur Arbre
DEBUT
    // on rappelle la fonction avec le fils droit
    SI (existeFilsDroit(a)) Alors
    ➡ ➡ fd(a) ← suppMax(fd(a), pe)
    // Si plus de fils droit, on a le successeur
    SINON
        *pe ← element(a)
        tmp ← a
        ➡ ➡ a ← fg(a)
        libérer(tmp)
    FIN SI
    RETOURNER a
FIN
```

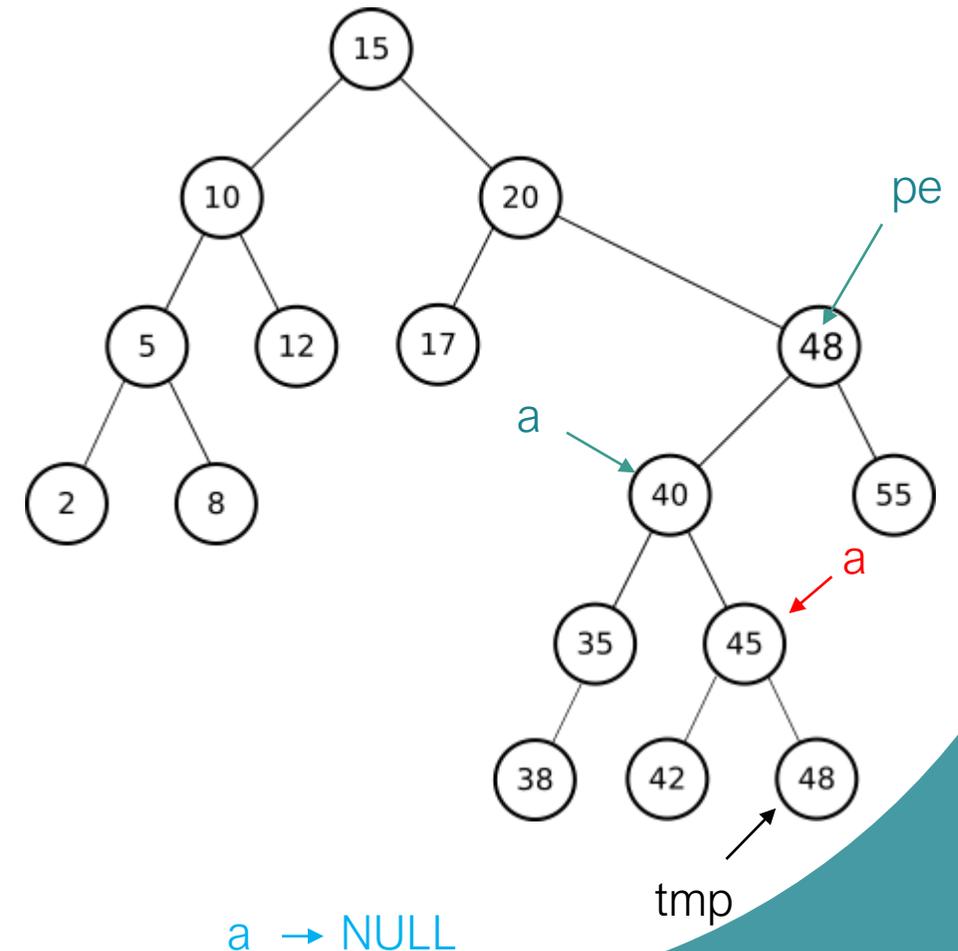


ABR : opération de suppression

- Algorithme : suppression (a,50)

```

FONCTION suppMax(a: arbre, pe: Ptr sur Element) :
ptr sur Arbre
VARIABLE
    tmp : ptr sur Arbre
DEBUT
    // on rappelle la fonction avec le fils droit
    SI (existeFilsDroit(a)) Alors
    ➡ ➡ fd(a) ← suppMax(fd(a), pe)
    // Si plus de fils droit, on a le successeur
    SINON
        *pe ← element(a)
        tmp ← a
        a ← fg(a)
        ➡ libérer(tmp)
    FIN SI
    RETOURNER a
FIN
```

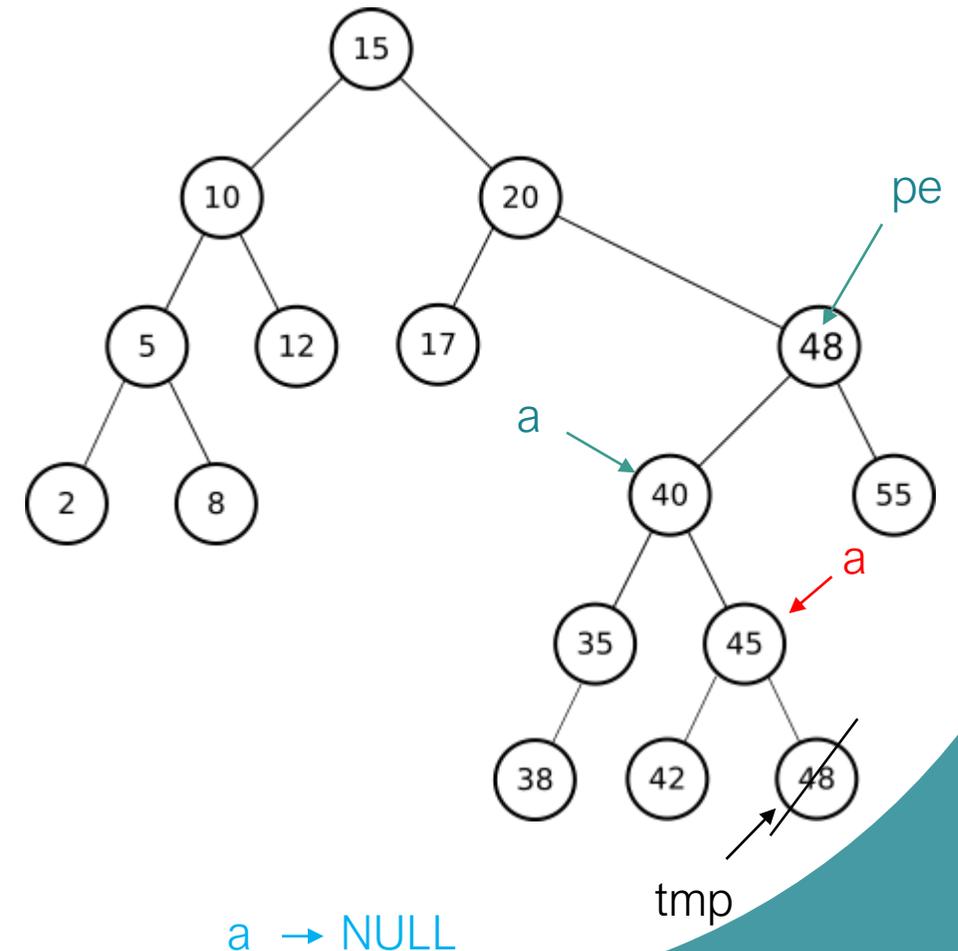


ABR : opération de suppression

- Algorithme : suppression (a,50)

```

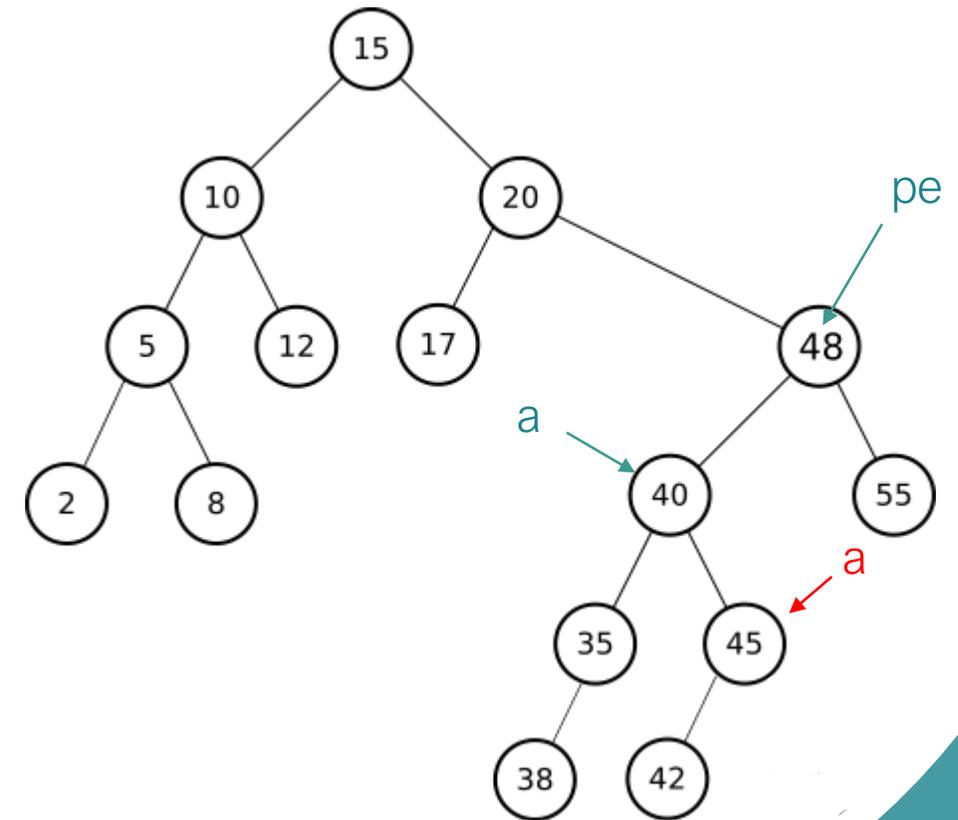
FONCTION suppMax(a: arbre, pe: Ptr sur Element) :
ptr sur Arbre
VARIABLE
    tmp : ptr sur Arbre
DEBUT
    // on rappelle la fonction avec le fils droit
    SI (existeFilsDroit(a)) Alors
    ➡ ➡ fd(a) ← suppMax(fd(a), pe)
    // Si plus de fils droit, on a le successeur
    SINON
        *pe ← element(a)
        tmp ← a
        a ← fg(a)
        libérer(tmp)
    FIN SI
    ➡ ➡ RETOURNER a
FIN
```



ABR : opération de suppression

- Algorithme : suppression (a,50)

```
FONCTION suppMax(a: arbre, pe: Ptr sur Element) :  
ptr sur Arbre  
VARIABLE  
    tmp : ptr sur Arbre  
DEBUT  
    // on rappelle la fonction avec le fils droit  
    SI (existeFilsDroit(a)) Alors  
        ➡ fd(a) ← suppMax(fd(a), pe)  
    // Si plus de fils droit, on a le successeur  
    SINON  
        *pe ← element(a)  
        tmp ← a  
        a ← fg(a)  
        libérer(tmp)  
    FIN SI  
    RETOURNER a  
FIN
```

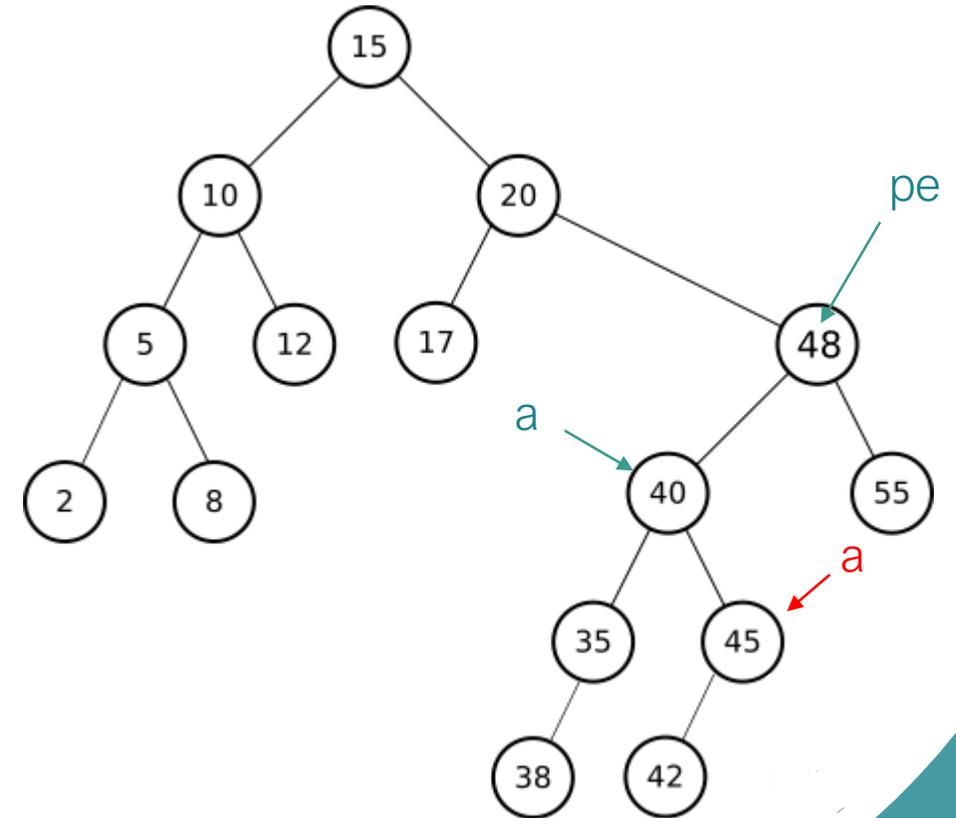


ABR : opération de suppression

- Algorithme : suppression (a,50)

```

FONCTION suppMax(a: arbre, pe: Ptr sur Element) :
ptr sur Arbre
VARIABLE
    tmp : ptr sur Arbre
DEBUT
    // on rappelle la fonction avec le fils droit
    SI (existeFilsDroit(a)) Alors
        ➡ fd(a) ← suppMax(fd(a), pe)
    // Si plus de fils droit, on a le successeur
    SINON
        *pe ← element(a)
        tmp ← a
        a ← fg(a)
        libérer(tmp)
    FIN SI
    ➡ RETOURNER a
FIN
```

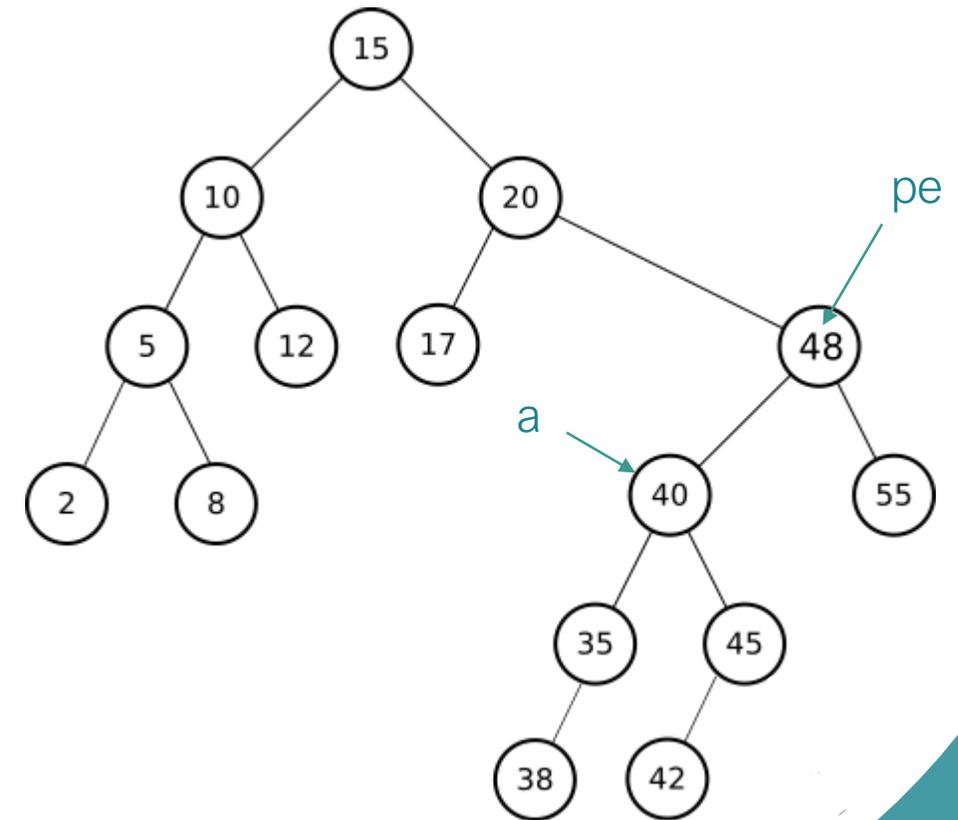


ABR : opération de suppression

- Algorithme : suppression (a,50)

```

FONCTION suppMax(a: arbre, pe: Ptr sur Element) :
ptr sur Arbre
VARIABLE
    tmp : ptr sur Arbre
DEBUT
    // on rappelle la fonction avec le fils droit
    SI (existeFilsDroit(a)) Alors
        fd(a) ← suppMax(fd(a), pe)
    // Si plus de fils droit, on a le successeur
    SINON
        *pe ← element(a)
        tmp ← a
        a ← fg(a)
        libérer(tmp)
    FIN SI
    ➡ RETOURNER a
FIN
```



ABR : opération de suppression

- Algorithme : suppression (a, 50)

FONCTION suppression(a: pointeur sur Arbre, e: element) :
pointeur sur Arbre

VARIABLE

tmp : ptr sur Arbre

DEBUT

// element non présent dans l'arbre

SI (a EST EGAL A NULL) Alors

RETOURNER a

// parcours récursif de l'arbre

SINON SI (e SUP. STRICT. A element(a))

➡➡ **fd(a)** ← suppression(fd(a), e)

SINON SI (e INF. STRICT. A element(a))

fg(a) ← suppression(fg(a), e)

// élément trouvé : remplacement par fils unique

SINON SI NON (existeFilsGauche(a))

tmp ← a

a ← **fd(a)**

libérer(tmp)

// élément trouvé : remplacement par prédécesseur

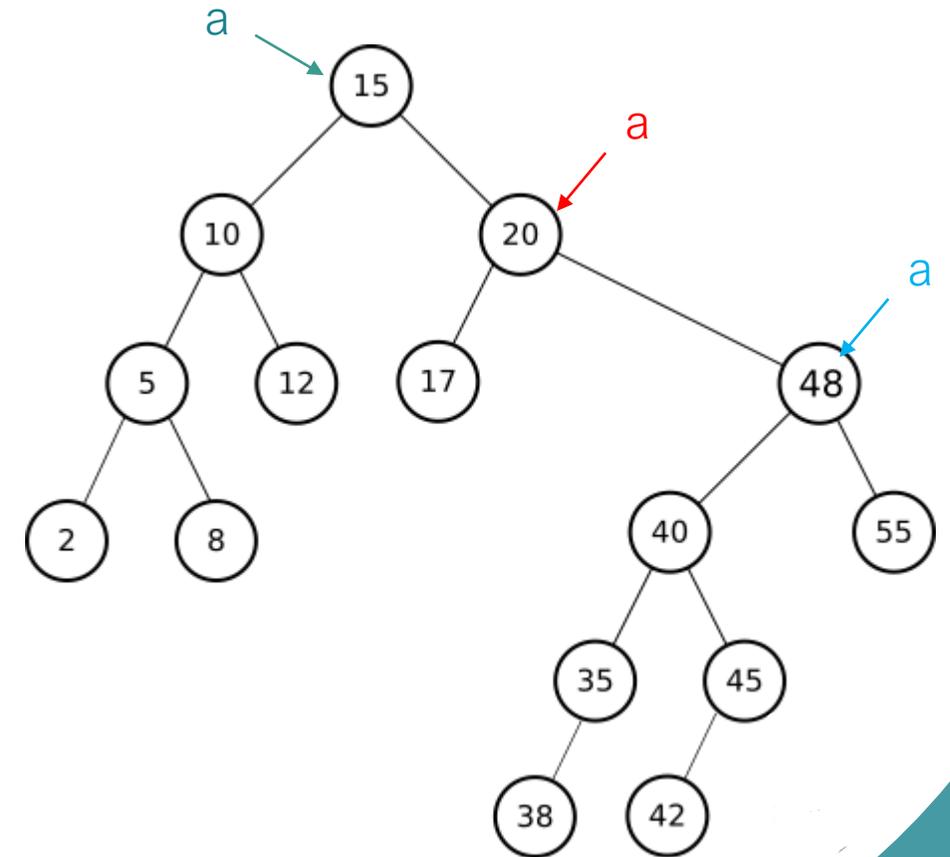
SINON

➡➡ **fg(a)** ← supMax(fg(a), adresse(element(a)))

FIN SI

RETOURNER a

FIN



ABR : opération de suppression

- Algorithme : suppression (a, 50)

FONCTION suppression(a: pointeur sur Arbre, e: element) :
pointeur sur Arbre

VARIABLE

tmp : ptr sur Arbre

DEBUT

// element non présent dans l'arbre

SI (a EST EGAL A NULL) Alors

RETOURNER a

// parcours récursif de l'arbre

SINON SI (e SUP. STRICT. A element(a))

➡ ➡ **fd(a)** ← suppression(fd(a), e)

SINON SI (e INF. STRICT. A element(a))

fg(a) ← suppression(fg(a), e)

// élément trouvé : remplacement par fils unique

SINON SI NON (existeFilsGauche(a))

tmp ← a

a ← **fd(a)**

libérer(tmp)

// élément trouvé : remplacement par prédécesseur

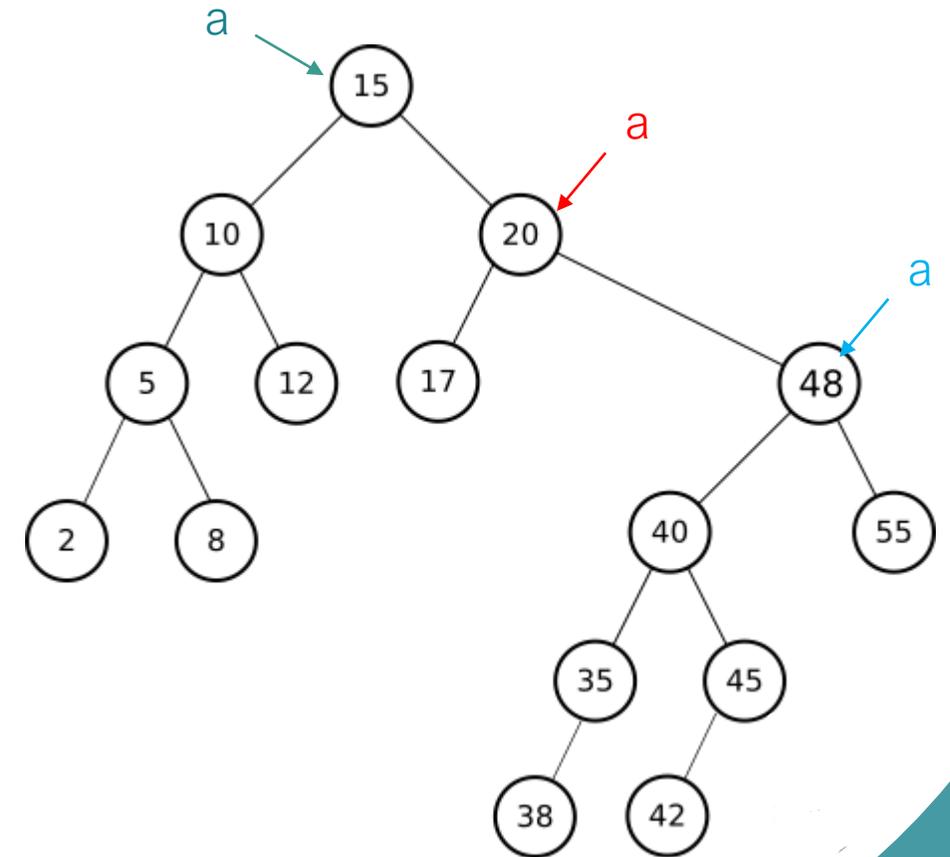
SINON

fg(a) ← supMax(fg(a), adresse(element(a)))

FIN SI

➡ RETOURNER a

FIN



ABR : opération de suppression

- Algorithme : suppression (a, 50)

FONCTION suppression(a: pointeur sur Arbre, e: element) :
pointeur sur Arbre

VARIABLE

tmp : ptr sur Arbre

DEBUT

// element non présent dans l'arbre

SI (a EST EGAL A NULL) Alors

RETOURNER a

// parcours récursif de l'arbre

SINON SI (e SUP. STRICT. A element(a))

➔ ➡ **fd(a)** ← suppression(fd(a), e)

SINON SI (e INF. STRICT. A element(a))

fg(a) ← suppression(fg(a), e)

// élément trouvé : remplacement par fils unique

SINON SI NON (existeFilsGauche(a))

tmp ← a

a ← **fd(a)**

libérer(tmp)

// élément trouvé : remplacement par prédécesseur

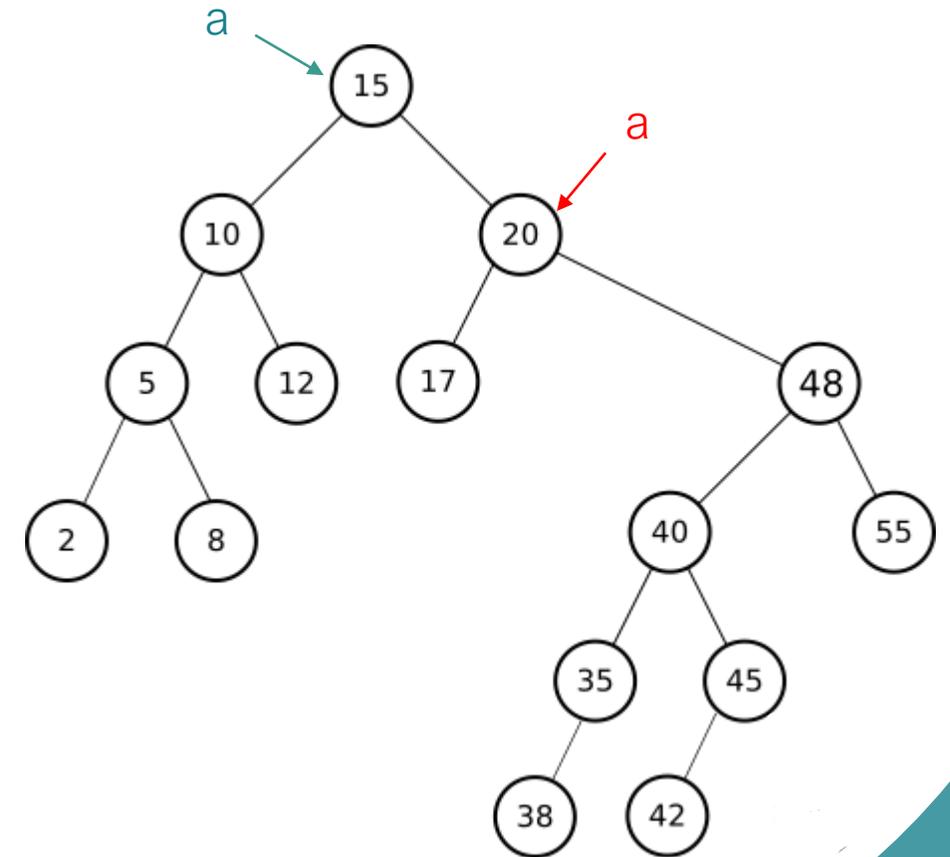
SINON

fg(a) ← supMax(fg(a), adresse(element(a)))

FIN SI

RETOURNER a

FIN



ABR : opération de suppression

- Algorithme : suppression (a, 50)

FONCTION suppression(a: pointeur sur Arbre, e: element) :
pointeur sur Arbre

VARIABLE

tmp : ptr sur Arbre

DEBUT

// element non présent dans l'arbre

SI (a EST EGAL A NULL) Alors

RETOURNER a

// parcours récursif de l'arbre

SINON SI (e SUP. STRICT. A element(a))

➡ **fd(a)** ← suppression(fd(a), e)

SINON SI (e INF. STRICT. A element(a))

fg(a) ← suppression(fg(a), e)

// élément trouvé : remplacement par fils unique

SINON SI NON (existeFilsGauche(a))

tmp ← a

a ← **fd(a)**

libérer(tmp)

// élément trouvé : remplacement par prédécesseur

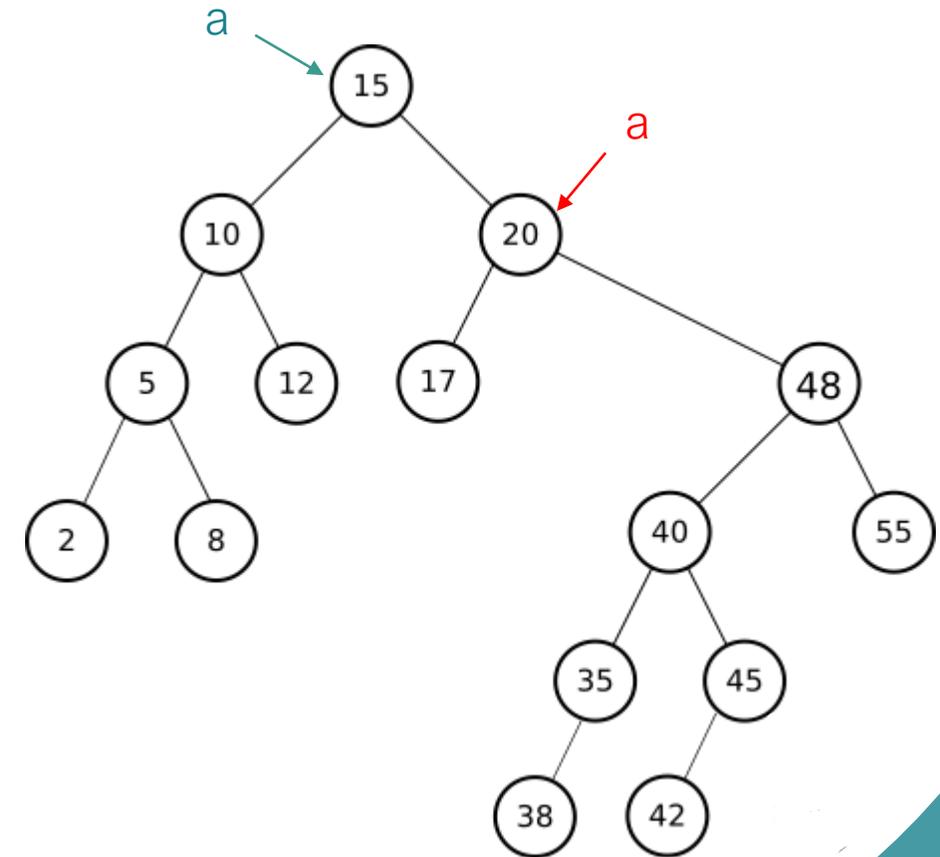
SINON

fg(a) ← supMax(fg(a), adresse(element(a)))

FIN SI

➡ RETOURNER a

FIN



ABR : opération de suppression

- Algorithme : suppression (a, 50)

FONCTION suppression(a: pointeur sur Arbre, e: element) :
pointeur sur Arbre

VARIABLE

tmp : ptr sur Arbre

DEBUT

// element non présent dans l'arbre

SI (a EST EGAL A NULL) Alors

RETOURNER a

// parcours récursif de l'arbre

SINON SI (e SUP. STRICT. A element(a))

➡ **fd(a)** ← suppression(fd(a), e)

SINON SI (e INF. STRICT. A element(a))

fg(a) ← suppression(fg(a), e)

// élément trouvé : remplacement par fils unique

SINON SI NON (existeFilsGauche(a))

tmp ← a

a ← **fd(a)**

libérer(tmp)

// élément trouvé : remplacement par prédécesseur

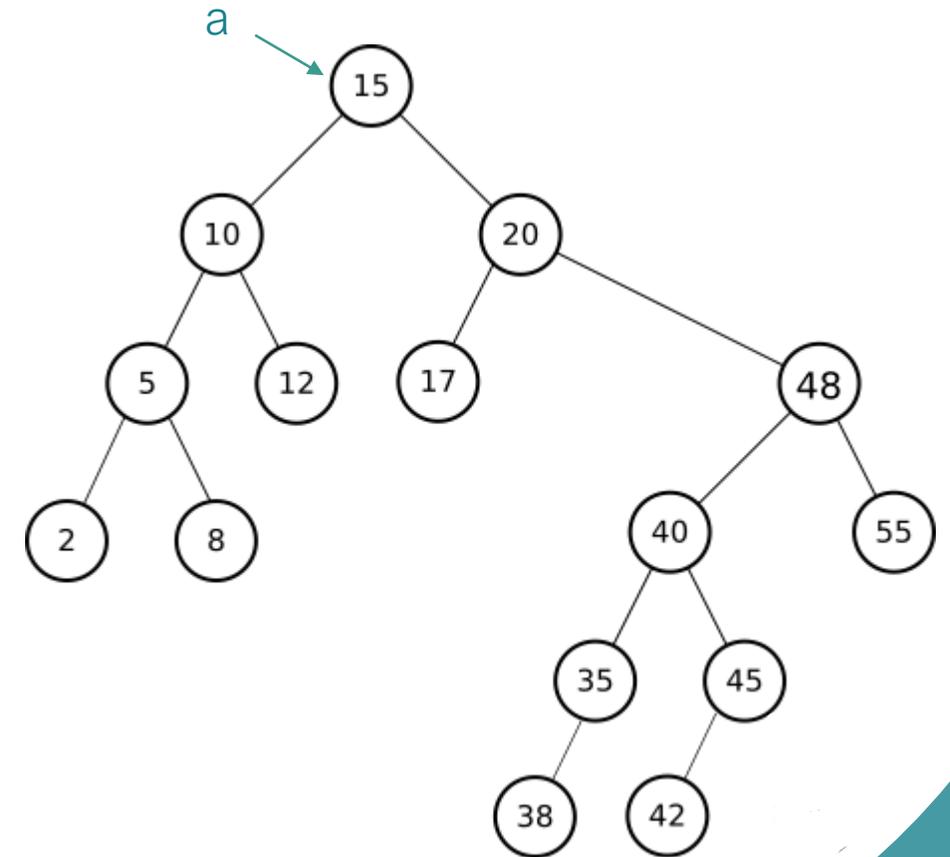
SINON

fg(a) ← supMax(fg(a), adresse(element(a)))

FIN SI

RETOURNER a

FIN



ABR : opération de suppression

- Algorithme : suppression (a, 50)

FONCTION suppression(a: pointeur sur Arbre, e: element) :
pointeur sur Arbre

VARIABLE

tmp : ptr sur Arbre

DEBUT

// element non présent dans l'arbre

SI (a EST EGAL A NULL) Alors

RETOURNER a

// parcours récursif de l'arbre

SINON SI (e SUP. STRICT. A element(a))

fd(a) ← suppression(fd(a), e)

SINON SI (e INF. STRICT. A element(a))

fg(a) ← suppression(fg(a), e)

// élément trouvé : remplacement par fils unique

SINON SI NON (existeFilsGauche(a))

tmp ← a

a ← fd(a)

libérer(tmp)

// élément trouvé : remplacement par prédécesseur

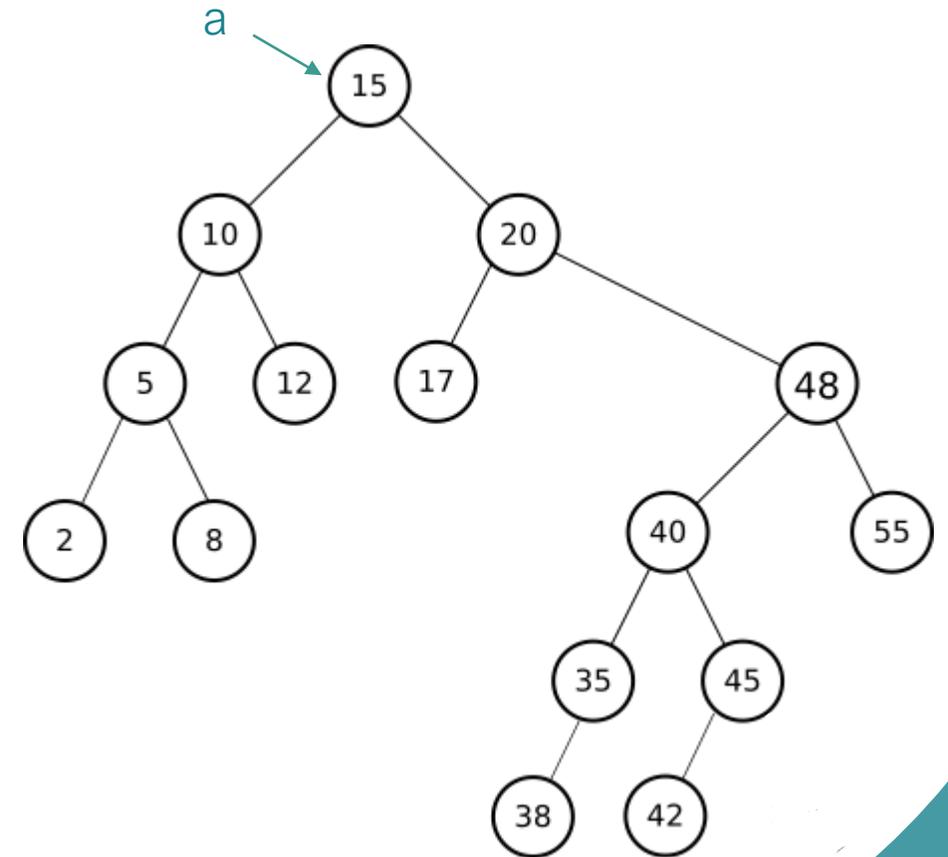
SINON

fg(a) ← suppMax(fg(a), adresse(element(a)))

FIN SI

➡ RETOURNER a

FIN



ABR : opération de suppression

- Algorithme : suppression (a, 50)

```
FONCTION suppression(a: pointeur sur Arbre, e: element) :  
pointeur sur Arbre
```

```
VARIABLE
```

```
    tmp : ptr sur Arbre
```

```
DEBUT
```

```
    // element non présent dans l'arbre
```

```
    SI (a EST EGAL A NULL) Alors
```

```
        RETOURNER a
```

```
    // parcours récursif de l'arbre
```

```
    SINON SI (e SUP. STRICT. A element(a))
```

```
        fd(a) ← suppression(fd(a), e)
```

```
    SINON SI (e INF. STRICT. A element(a))
```

```
        fg(a) ← suppression(fg(a), e)
```

```
    // élément trouvé : remplacement par fils unique
```

```
    SINON SI NON (existeFilsGauche(a))
```

```
        tmp ← a
```

```
        a ← fd(a)
```

```
        libérer(tmp)
```

```
    // élément trouvé : remplacement par prédécesseur
```

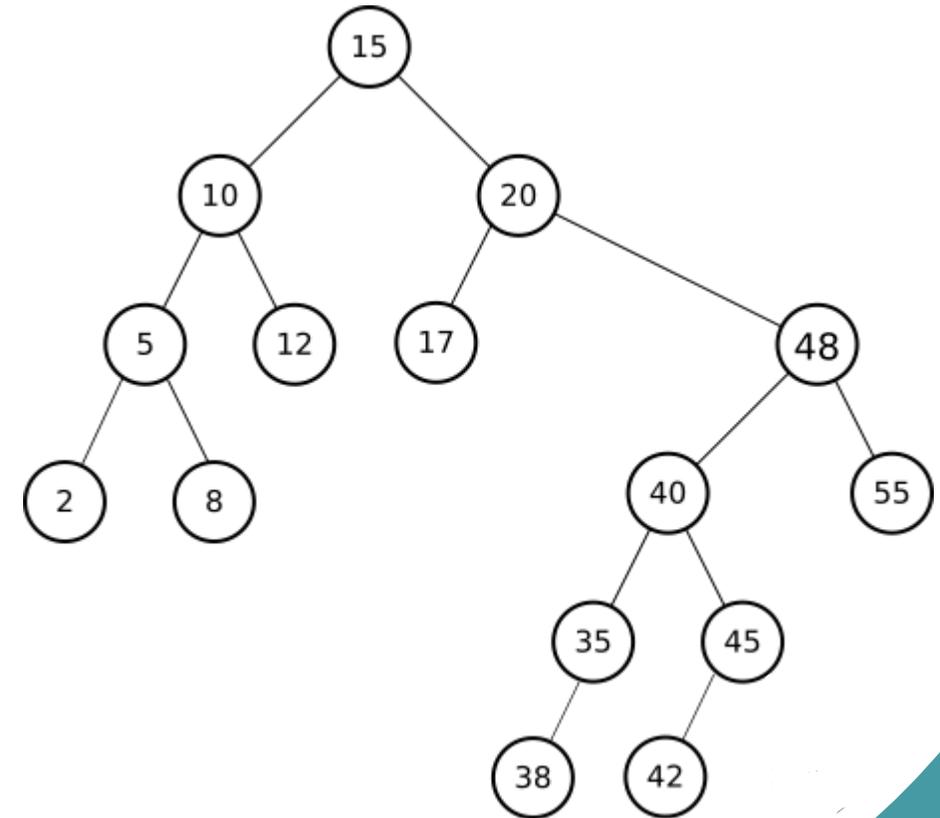
```
    SINON
```

```
        fg(a) ← suppMax(fg(a), adresse(element(a)))
```

```
    FIN SI
```

```
    RETOURNER a
```

```
FIN
```



Résumé

- Les arbres binaires de recherche sont des arbres binaires dont les éléments respectent une relation d'ordre :
 - tout élément d'un sous-arbre gauche doit être inférieur strictement à la racine
 - tout élément d'un sous-arbre droit doit être supérieur strictement à la racine
 - Dans notre cours, nous imposons un cas particulier : l'arbre ne contient que des valeurs distinctes d'éléments
- Cette structure permet d'accélérer les opérations de recherche ($O(\log_2)$) plutôt que $O(n)$ pour les arbres binaires classiques.
- Les opérations d'insertions et de suppressions doivent maintenir la relation d'ordre.
- Il existe encore d'autres moyens d'optimiser la recherche dans un ABR : c'est le sujet du prochain cours !