Arbres binaires (suite)

Alix Munier-Kordon et Maryse Pelletier

LIP6
Université P. et M. Curie
Paris

Algorithmique élémentaire

Plan du cours

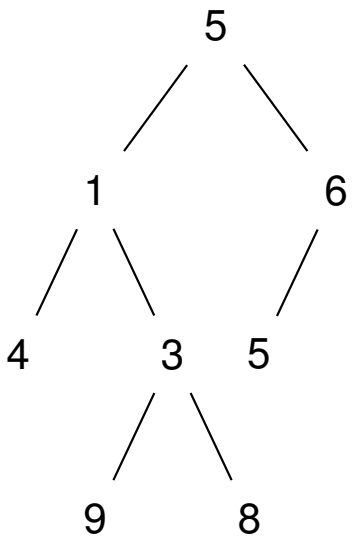
Arbres H-équilibrés : définition

Definition

Un arbre binaire est *H-équilibré* si, pour tout nœud x, la différence entre la hauteur du fils gauche de x et la hauteur du fils droit de x vaut 1, 0 ou -1.

Arbres H-équilibrés : exemples

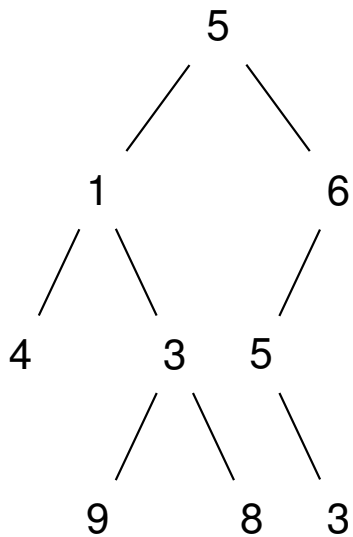
Arbre H-équilibré :



difference de 1 ou 0 étage entre les fils

= arbre H-équilibré

Arbre non H-équilibré :



Arbres H-équilibrés : propriété

Theorem

La hauteur d'un arbre H-équilibré de taille n est en $\Theta(\log n)$.

Meilleur cas : arbres *parfaits* (voir section suivante)
Pire cas : arbres *de Fibonacci* (voir TD)

Arbres parfaits : définition

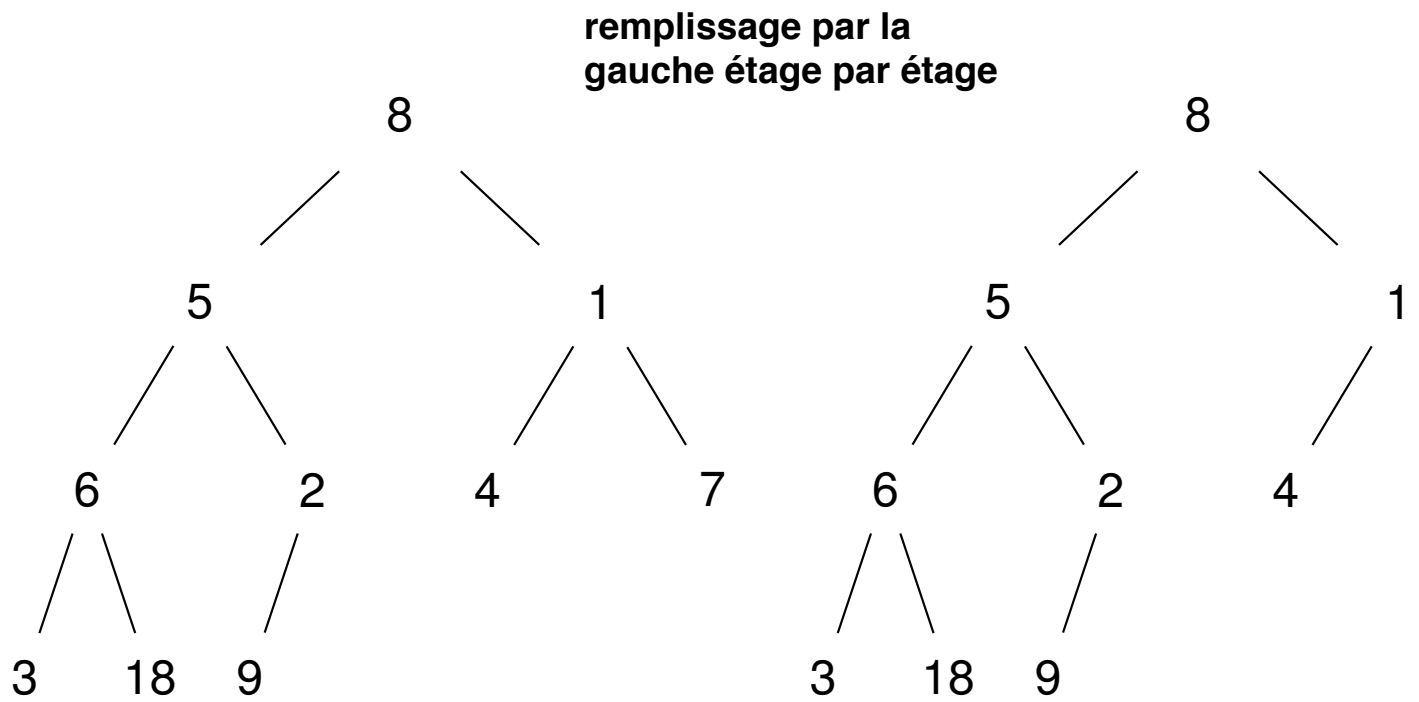
Definition

Un *arbre parfait* est un arbre dont tous les niveaux sont entièrement pleins, sauf éventuellement le dernier qui est rempli le plus à gauche.

Arbres parfaits : exemples

Arbre parfait :

Arbre non parfait :



Arbres parfaits : propriété

Theorem

La hauteur d'un arbre parfait de taille n est en $\Theta(\log n)$.

Preuve : $2^{h-1} \leq n < 2^h$ (voir TD)

Exemple : un arbre parfait de taille 1000 a une hauteur égale à 10 : les 9 premiers niveaux sont entièrement remplis et le 10-ème niveau contient 489 nœuds.

Arbres parfaits : propriété

Theorem

La hauteur d'un arbre parfait de taille n est en $\Theta(\log n)$.

Preuve : $2^{h-1} \leq n < 2^h$ (voir TD)

Exemple : un arbre parfait de taille 1000 a une hauteur égale à 10 : les 9 premiers niveaux sont entièrement remplis et le 10-ème niveau contient 489 nœuds.

Arbres parfaits : représentation

Un arbre parfait de taille n peut être représenté au moyen d'un tableau $A[0..N]$, avec $N \geq n$, tel que :

- $A[0]$ contient la taille n de T
- les cases $A[1..n]$ sont remplies en parcourant T de gauche à droite, niveau par niveau.

Représentation : exemple

Arbre parfait :

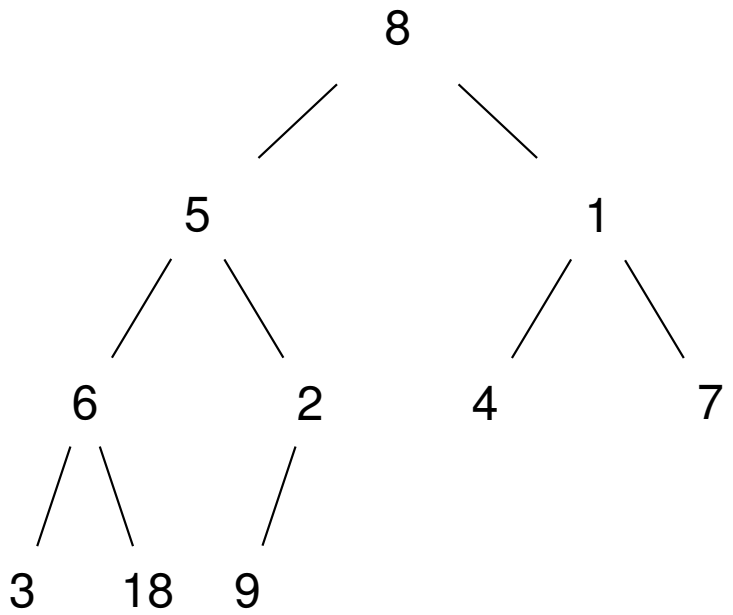


Tableau : [**10**, 8, 5, 1, 6, 2, 4, 7, 3, 18, 9]

Arbres parfaits : représentation

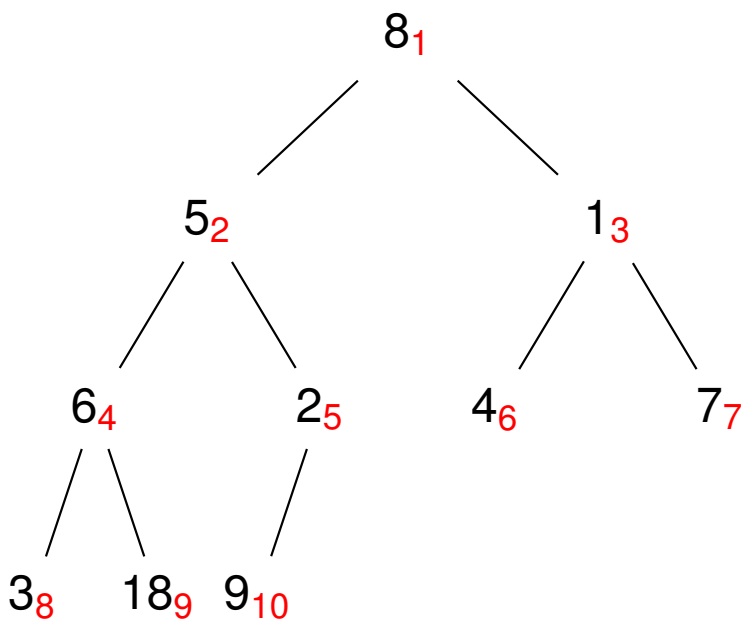
Dans le tableau $A[0..N]$ représentant un arbre parfait T :

- $A[1]$ est la racine de T
- si $2i \leq n$ alors le fils gauche de $A[i]$ est $A[2i]$
- si $2i + 1 \leq n$ alors le fils droit de $A[i]$ est $A[2i + 1]$
- si $i > 1$ alors le père de $A[i]$ est $A[i \div 2]$.

$\div$ est la division entière

Représentation : exemple

Numérotation d'un arbre parfait :



Insertion dans un arbre parfait

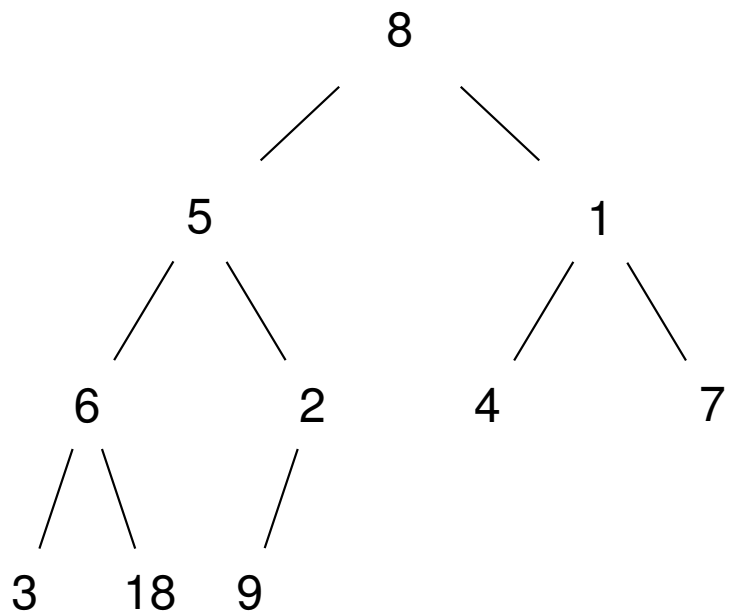
Insertion d'un élément x dans un arbre parfait de taille n.
Si $n < N$:

- insérer x dans la case $A[n+1]$
- augmenter $A[0]$ de 1.

Et si $n = N$? On peut décider de ne plus faire d'insertion ou bien doubler la taille du tableau.

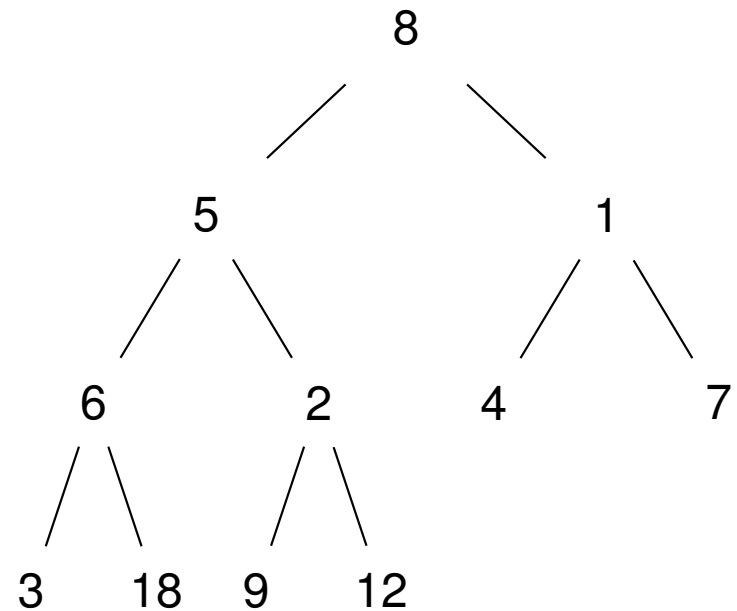
Insertion : exemple

Avant :



[**10**, 8, 5, 1, 6, 2, 4, 7, 3, 18, 9]

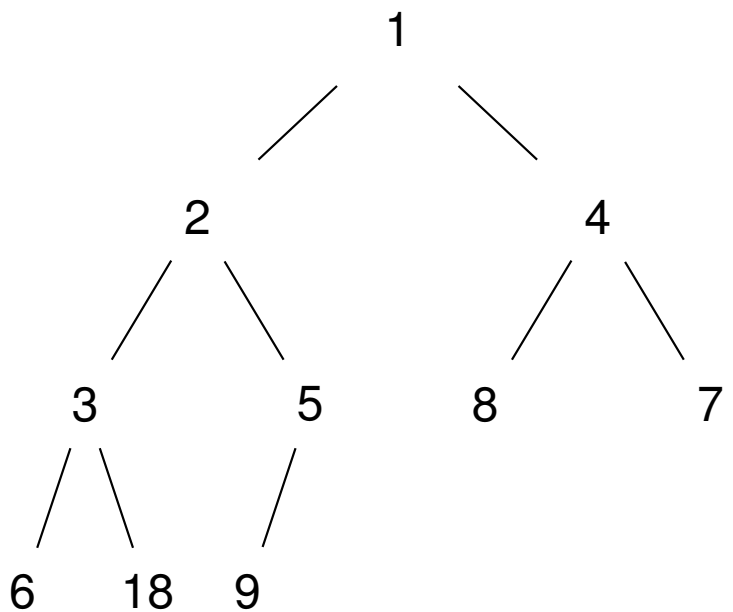
Après :



[**11**, 8, 5, 1, 6, 2, 4, 7, 3, 18, 9, 12]

Utilité des arbres parfaits : tas

Tas : arbre parfait croissant.



Les étiquettes croissent de la racine vers les feuilles.

Utilité des arbres parfaits : tas

- Insertion dans un tas : comme dans un arbre parfait, puis échanges avec le père tant que nécessaire.
  Complexité : $O(\log n)$.
- Suppression du minimum, avec obtention d'un nouveau tas. Complexité : $O(\log n)$.
- Tri par tas, par n insertions puis n suppressions de minimum.
  Complexité : $O(n \log n)$.

Le tri par tas sera vu en TD.

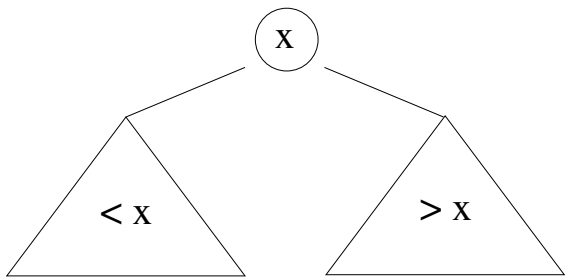
Arbres binaires de recherche : définition

L'ensemble E des clefs est totalement ordonné.

Definition (ABR(E))

Un *arbre binaire de recherche* sur E est un arbre binaire dans lequel tout nœud a une clef qui est supérieure à toutes les clefs de son sous-arbre gauche et inférieure à toutes les clefs de son sous-arbre droit.

Pour tout nœud x:



Abréviation : ABR pour arbre binaire de recherche.
Remarque : les clefs d'un ABR sont deux à deux distinctes.

Arbres binaires de recherche : définition inductive

Definition ($\mathcal{ABR}$(E))

$T \in \mathcal{ABR}$(E) si :

Base $T = \emptyset$;

Ind. $T = (x, G, D)$ avec $x \in E$ et tel que,

- G et D sont dans $\mathcal{ABR}$(E);
- toutes les clefs de G sont inférieures à x;
- toutes les clefs de D sont supérieures à x.

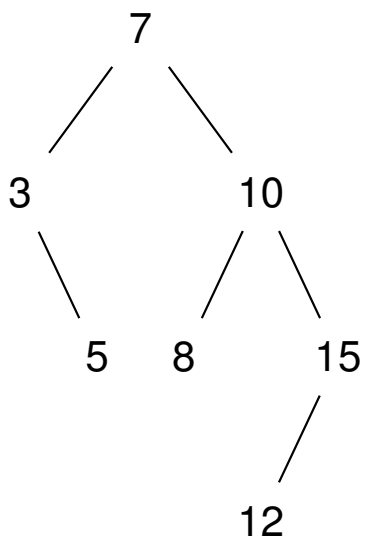
Theorem

Pour tout ensemble E ordonné, $\mathcal{ABR}(E) = ABR(E)$

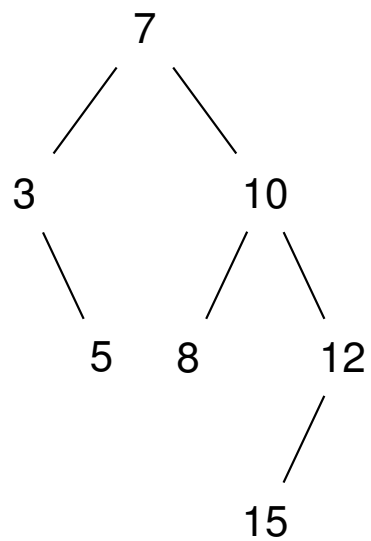
Preuve en TD

ABR : exemples

ABR :



Non ABR :



le problème est le 12

Propriété caractéristique

infixe :bac
prefixe :abc
suffixe :bca

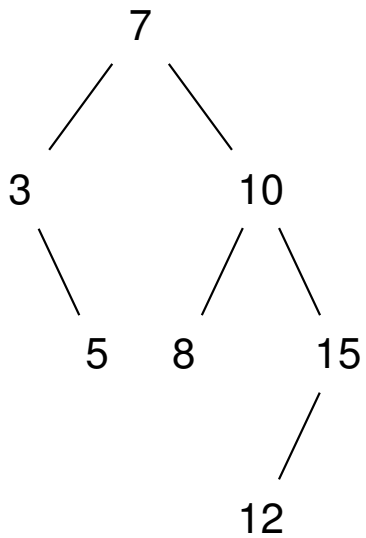
Theorem

Un arbre binaire est un ABR ssi son parcours infixe est rangé en ordre strictement croissant.

Preuve en TD

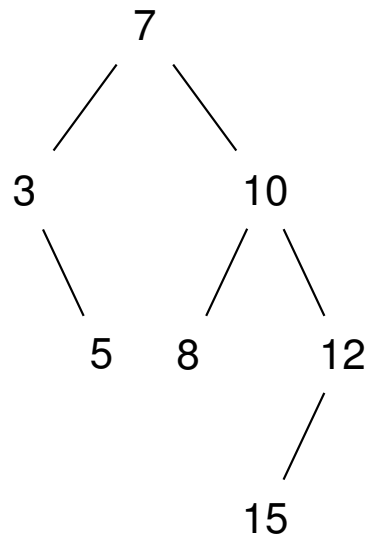
Propriété caractéristique : exemple

ABR :



(3, 5, 7, 8, 10, 12, 15)

Non ABR :



(3, 5, 7, 8, 10, 15, 12)

Recherche dans un ABR

Idée : comparer la clef cherchée et la racine pour savoir s'il faut descendre à gauche ou à droite.

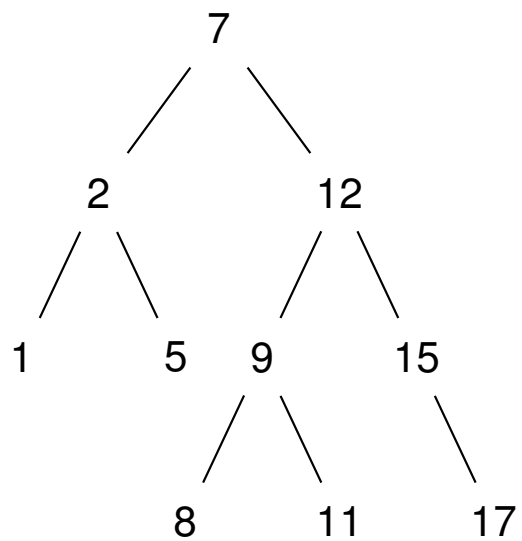
Algorithme : recherche de x dans l'ABR T

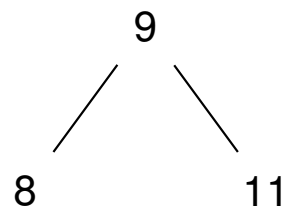
```
def ABRcherche(x,T):
    if estABvide(T):
        return False
    if x == T.clef:
        return T
    if x < T.clef:
        return ABRcherche(x,T.gauche)
    return ABRcherche(x,T.droit)
```

Complexité : la hauteur de l'ABR.

Recherche dans un ABR : exemple

`ABRcherche(9,T)` renvoie





Insertion dans un ABR

Idée : comparer la nouvelle clef à la racine pour savoir s'il faut l'insérer à gauche ou à droite.

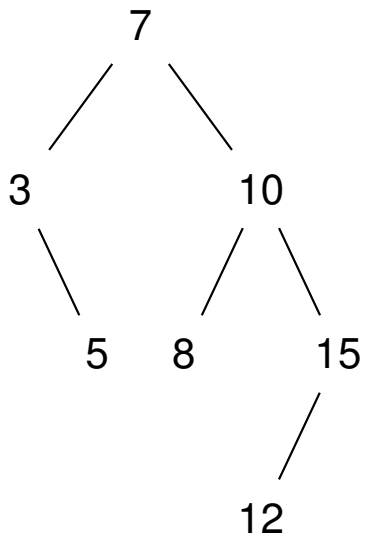
Algorithme : insertion de x dans l'ABR T

```
def ABRinsertion(x,T):
    if estABvide(T):
        return ABfeuille(x)
    if x == T.clef:
        return T
    if x < T.clef:
        return AB(T.clef,ABRinsertion(x,T.gauche),T.droit)
    return AB(T.clef,T.gauche,ABRinsertion(x,T.droit))
```

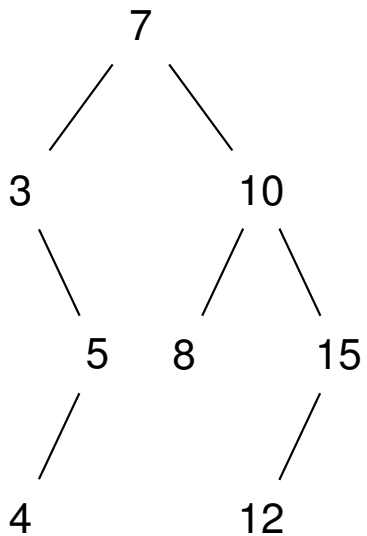
soit x = a une clef de l'arbre
soit x sera une feuille de l'arbre

Complexité : la hauteur de l'ABR.

Insertion dans un ABR : exemple



`ABRinsertion(4,T)` renvoie

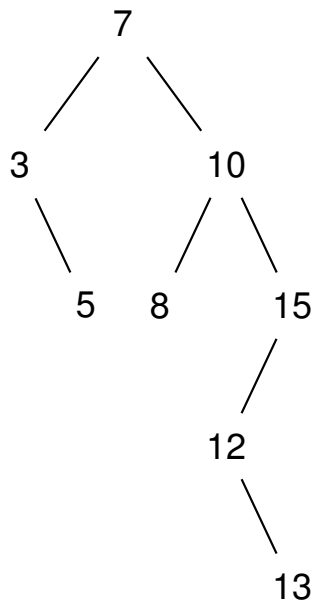


Recherche du maximum d'un ABR

Idée : le maximum est dans le nœud le plus à droite.

Algorithme : en TD

Complexité : la longueur de la *branche droite*.



Le maximum est 15.
Les nœuds de la branche droite sont 7, 10, 15.
La longueur de la branche droite est 3.

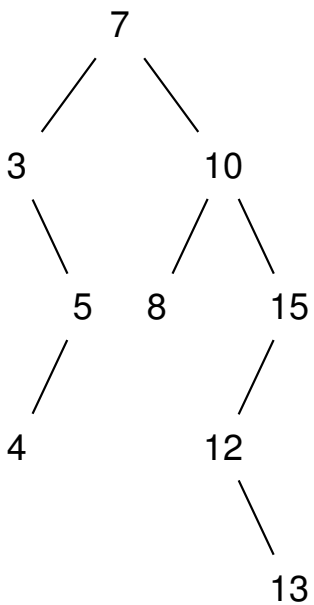
Suppression du maximum d'un ABR

Idée : remplacer le nœud le plus à droite par son fils gauche.

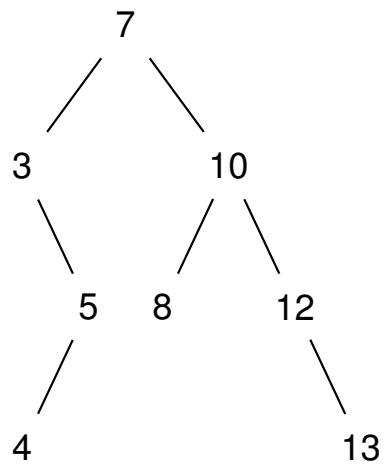
Algorithme : en TD

Complexité : la longueur de la branche droite.

Avant suppression du max



Après suppression du max



Suppression de la racine d'un ABR

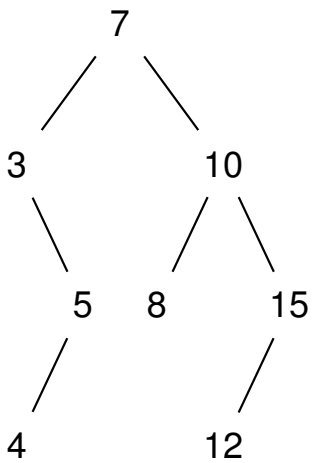
Idée : Si le sous-arbre gauche n'est pas vide :

- remplacer la racine par le max du sous-arbre gauche,
- supprimer le max du sous-arbre gauche.

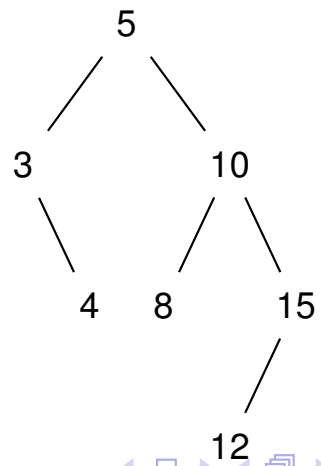
Algorithme : en TD

Complexité : la longueur de la branche droite du sous-arbre gauche.

Avant suppression racine



Après suppression racine



Suppression d'un élément dans un ABR

Algorithme : suppression de x dans l'ABR T

```
def ABRsuppression(x,T):
    if estABvide(T):
        return T
    if x == T.clef:
        return ABRmoinsRacine(T)
    if x < T.clef:
        return AB(T.clef,ABRsuppression(x,T.gauche),T.droit)
    return AB(T.clef,T.gauche,ABRsuppression(x,T.droit))
```

ici le travail est fait par la fonction ABRmoinsRacine(T) elle définit la condition d'arrêt

Complexité : la hauteur de l'ABR.