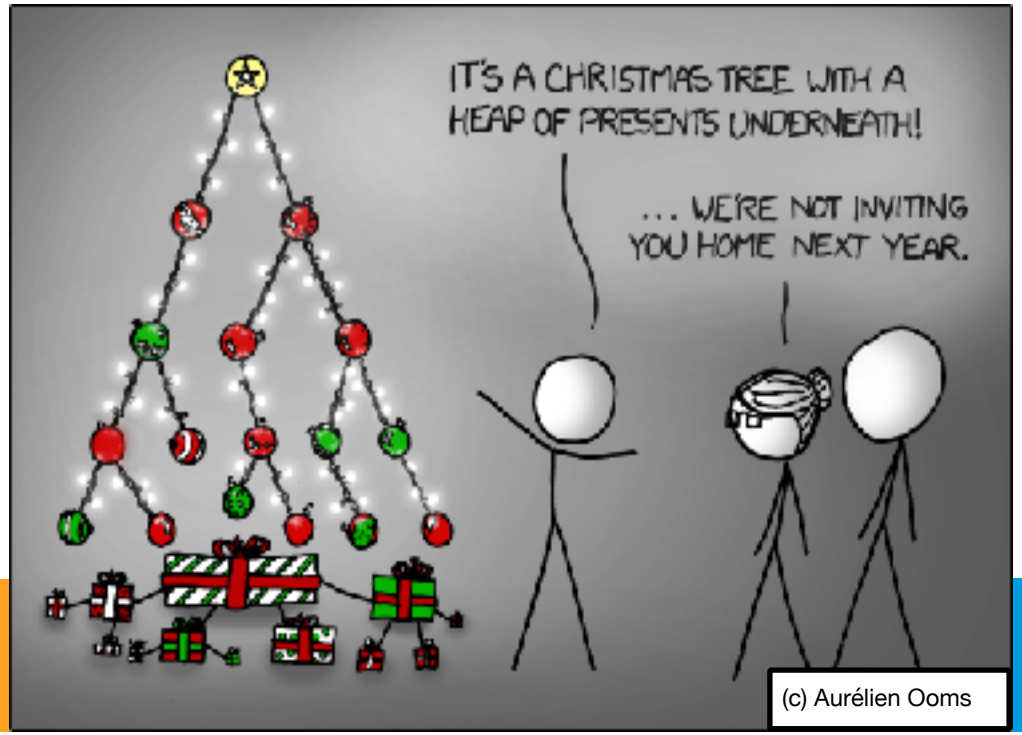
8. Collections « ordonnées »

Les structures arborescentes - partie 2

Arbre binaires de recherche

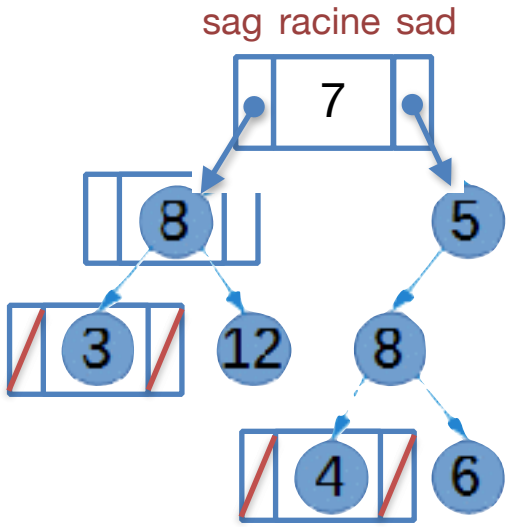


Rappels

Spécification fonctionnelle : **ArbBinT**

```
créer_ab: -> ArbBinT #crée un arbre binaire (vide)
ab_vide: ArbBinT -> booléen #vérifie si l'arbre est vide
racine: ArbBinT -> T #retourne la valeur du nœud racine
sag: ArbBinT -> ArbBinT #sous-arbre gauche de l'arbre binaire
sad: ArbBinT -> ArbBinT #sous-arbre droit de l'arbre binaire
enraciner: T x ArbBinT x ArbBinT -> ArbBinT #enracine les deux
arbres binaires à une racine de la valeur passée en paramètre
```

Parcours infixe, préfixe, postfixe



8.4 Arbres binaires de recherche

L'objectif est de représenter un ensemble de valeurs disposant d'une relation d'ordre et pour lequel les opérations privilégiées sont :

- l'insertion d'une valeur
- la suppression d'une valeur
- la recherche d'une valeur
- le parcours dans l'ordre des valeurs

Spécification **fonctionnelle**

```
créer_abr: -> ABR #crée un ABR vide
abr_vide: ABR -> booléen #vérifie si l'ABR est vide
insère: ABR x T -> ABR #insert la valeur dans l'ABR
recherche: ABR x T -> Bool #renvoie vrai si la valeur appartient
à l'ABR.
supprime: ABR x T -> ABR #supprime la valeur de l'ABR
tri: ABR -> ABR #tri l'ensemble selon la relation d'ordre
```

Propriétés

```
P1: abr_vide(creer_abr()) == True
P2: supprime(a,t)!= a => recherche(a,t) == True
P3: recherche(a,t) == False => supprime(a,t) == a
P4: recherche(insère(a,t),t) == True
```

Description **logique**

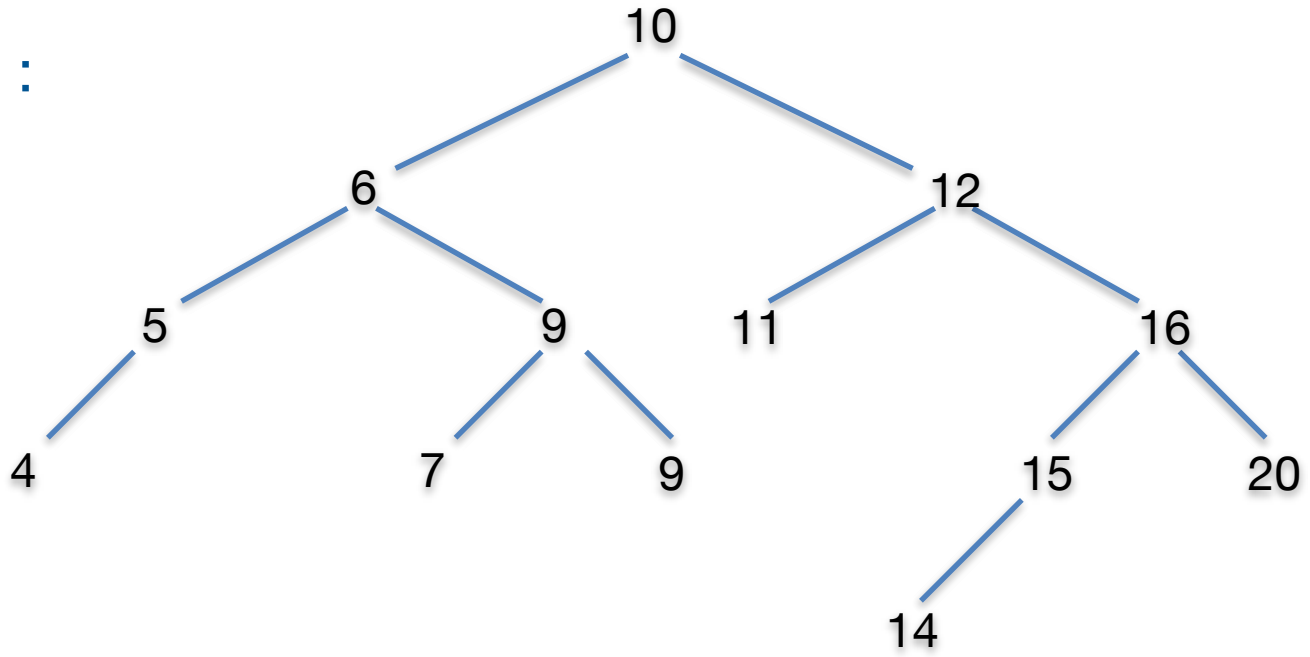
Une structure arborescente parait naturelle pour essayer de rendre ces opérations efficaces en temps calcul : elles semblent promettre de ne pas être obligé de parcourir tous les éléments

Pour garantir **l'efficacité** de ces opérations, on définit une façon particulière de structurer les informations : **arbre binaire de recherche**

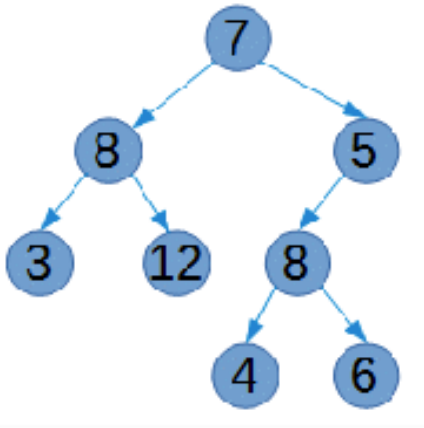
Un arbre binaire de recherche est un un **arbre binaire** avec deux propriétés importantes :

- toutes les valeurs du sous-arbre gauche d'un nœud sont inférieures à la valeur de ce nœud
- toutes les valeurs du sous-arbre droit d'un nœud sont supérieures ou égales à la valeur de ce nœud.

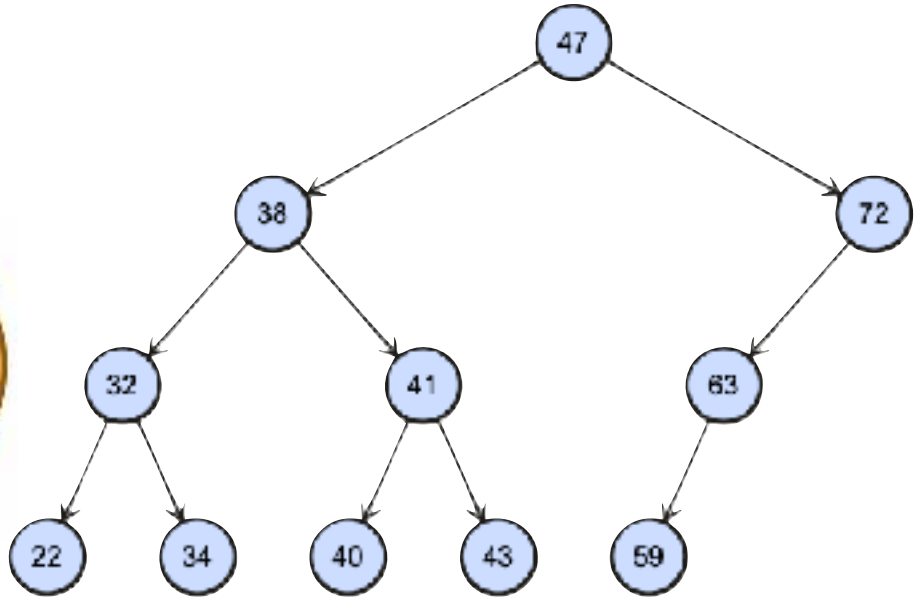
Exemple d'ABR :



Pourquoi cet arbre n'est-il pas un ABR ?



Cet arbre est-il un ABR ?



Structure de données

```
Type ABR = (Vide | ABRnv)

Type ABRnv
  racine : T
  SAG : ABR
  SAD : ABR
endType
```

Comme on peut le voir, la structure de données d'un ABR est la même que celle d'un AB. Seules les fonctions et l'application des propriétés peut différencier un AB d'un ABR :

$$\forall x \in SAG(a), racine(x) < racine(a)$$

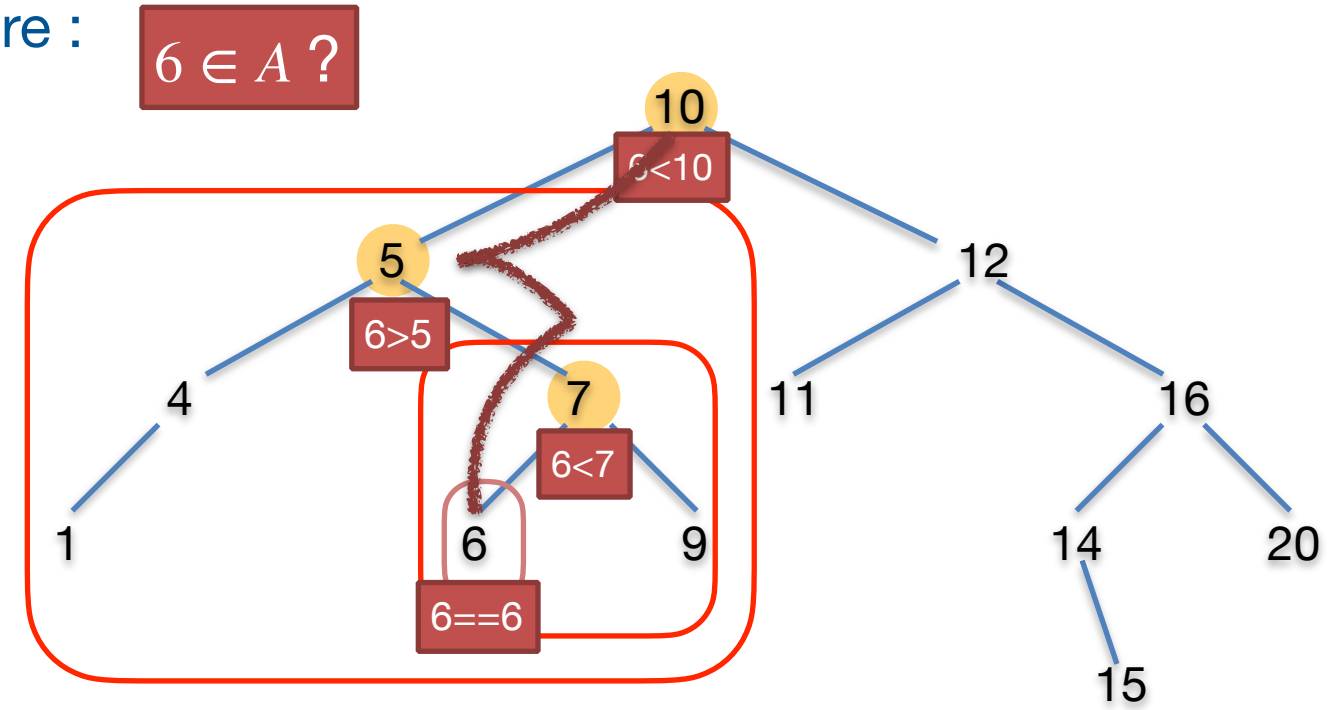
$$\forall x \in SAD(a), racine(x) \geq racine(a)$$

On spécifie ici les fonctions de la structure de données ; il suffit de reprendre celles de l'arbre binaire.

```
racine: ABR -> T #retourne la valeur du nœud racine
sag: ABR -> ABR #sous-arbre gauche de l'arbre binaire
sad: ABR -> ABR #sous-arbre droit de l'arbre binaire
enraciner: t x ABR x ABR -> ABR # crée un arbre dont la
racine a pour valeur t et qui a pour sous-arbres les deux
arbres passés en paramètres
enraciner_abr: ABR x ABR x ABR -> ABR # crée un arbre dont
la racine est la racine du premier arbre passé en paramètre
et qui a pour sous-arbres les deux derniers arbres passés
en paramètres
set_racine: ABR x T -> ABR #remplace la valeur de la racine
par celle passée en paramètre et retourne l'arbre ainsi
modifié
```

Algorithme des fonctions sur les ABR : **Recherche**

Grâce aux propriétés de l'ABR, on explore seulement une partie de l'arbre :



Intérêt : la complexité est en O(log(n)) si l'arbre est équilibré, en O(n) dans le pire des cas.

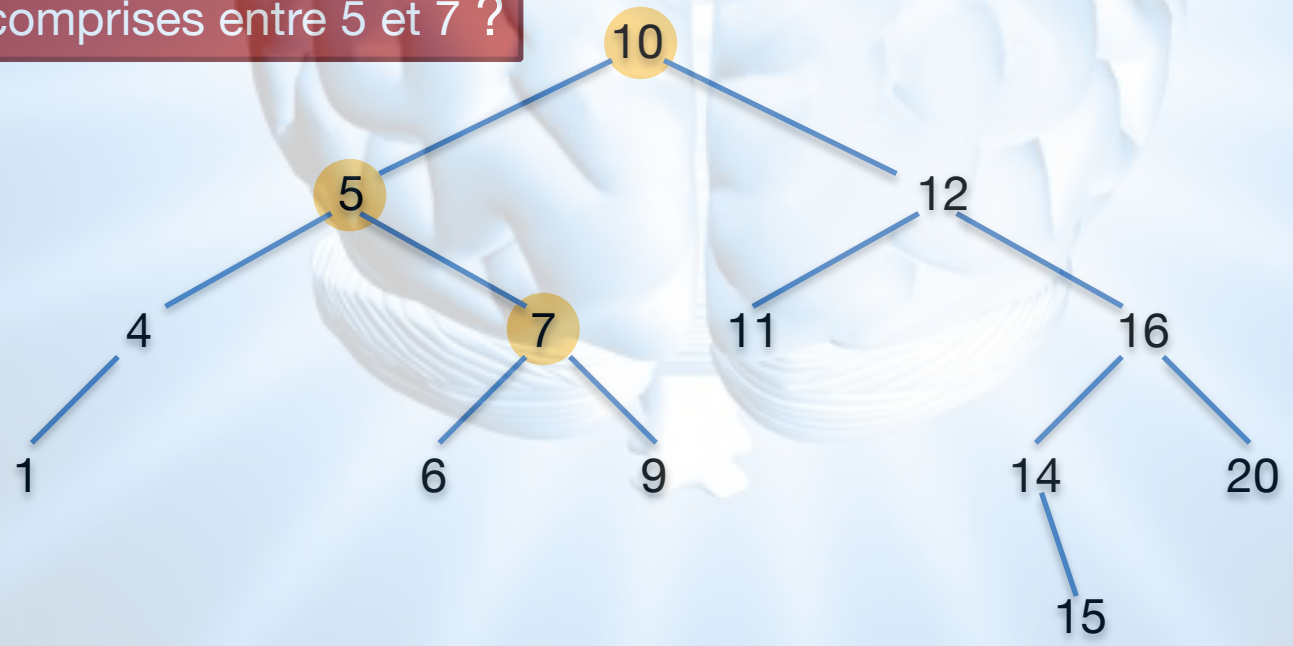
Algorithme des fonctions sur les ABR : **Recherche**

Activité Web : essayons quelques recherches sur des ABR pour vérifier votre compréhension de l'algorithme de recherche

Algorithme des fonctions sur les ABR : RechercheRange

`Exercice*` : à effectuer **en autonomie** en dehors des séances avec l'aide de cette activité Moodle

Valeurs comprises entre 5 et 7 ?



Algorithme des fonctions sur les ABR : **Insertion**

Pour insérer une valeur, il suffit d'effectuer une recherche et de rajouter la valeur lorsqu'on atteint une feuille de l'arbre.

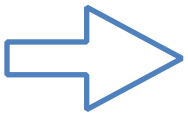
La complexité est la même que celle de la fonction recherche

Activité Web : essayons des insertions dans des ABR

Algorithme des fonctions sur les ABR : **Suppression**

Pour supprime une valeur, il faut distinguer 3 cas :

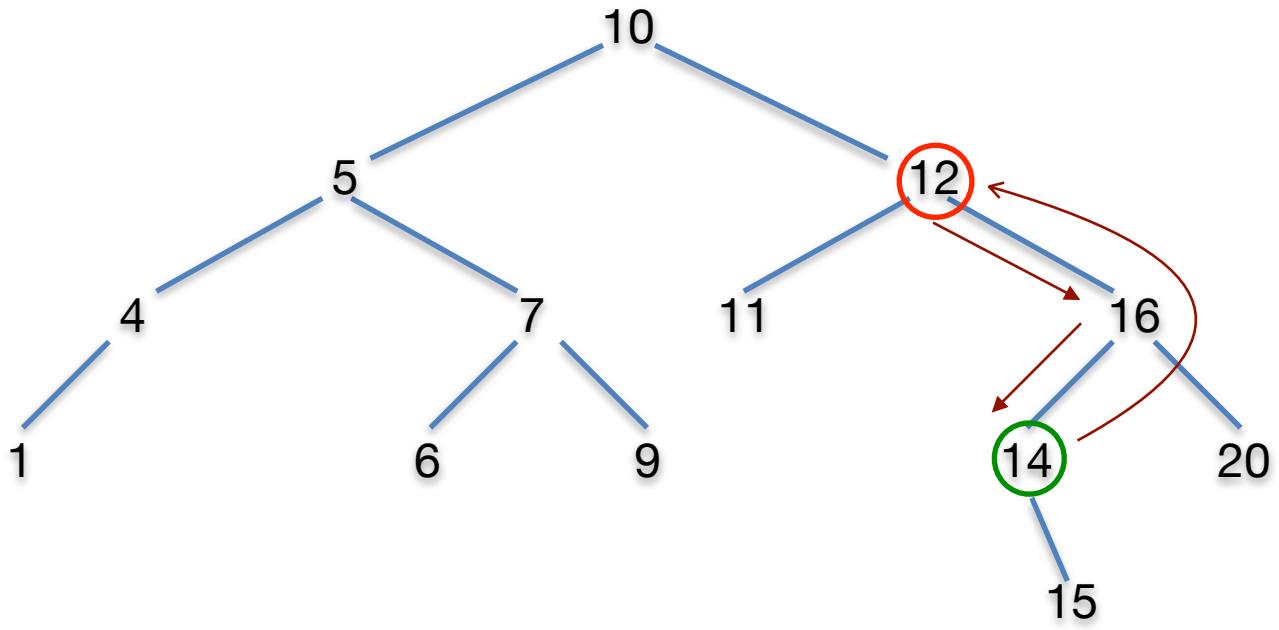
1. si la valeur est sur une feuille, il suffit de la supprimer
2. si la valeur est sur un nœud qui n'a qu'un enfant, il suffit de remplacer ce nœud par son enfant
3. si la valeur est sur un nœud avec deux enfants, alors il faut remplacer la valeur de la racine par la plus petite valeur des valeurs plus grandes que la valeur à supprimer (ou par la plus grande valeur des valeurs plus petites que celle à supprimer)



```
successeur_ab: ABR x T -> ABR
prédécesseur_ab: ABR x T -> ABR
```

Exemple : essayons de supprimer la valeur 12 :

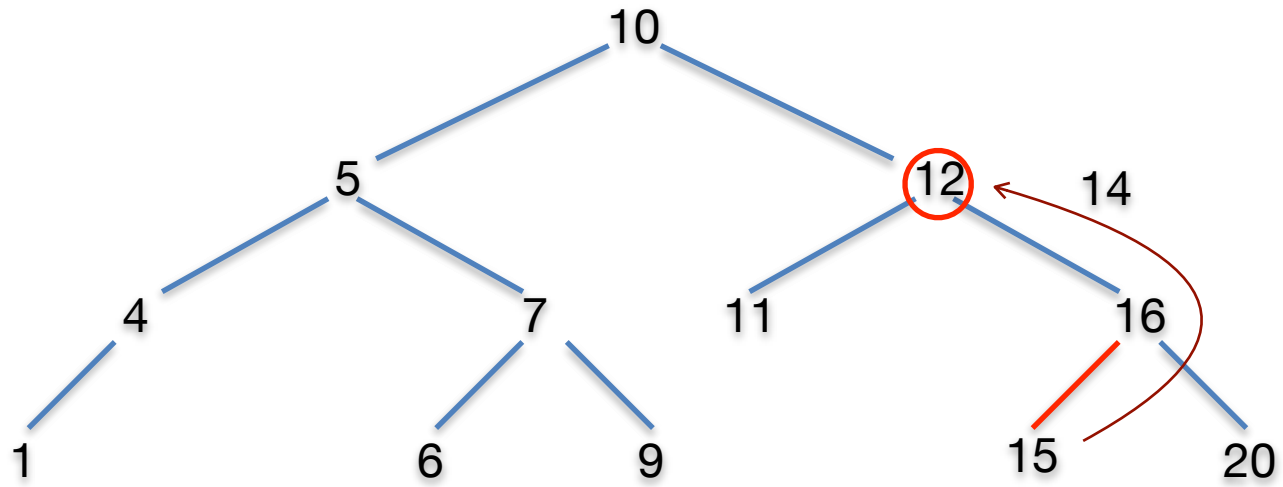
- … est la plus petite des valeurs plus grandes que 12
- elle doit venir prendre la place de 12 dans l'arbre : il faut donc la supprimer de son emplacement actuel (appel rec !)



Exemple : essayons de supprimer la valeur 12 :

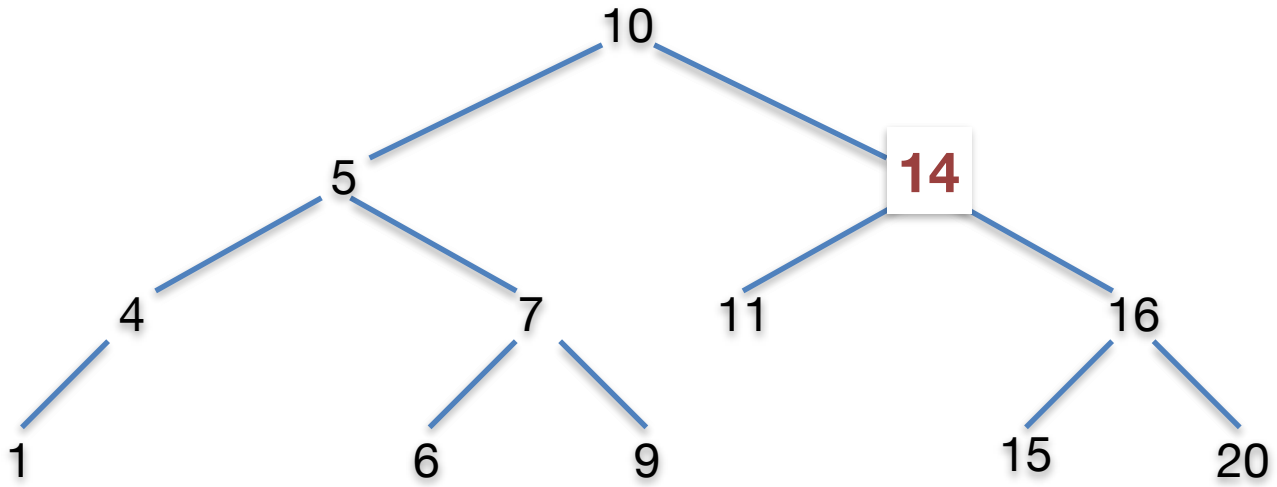
- Il faut donc supprimer … de sa place actuelle :
  - elle n'a qu'un fils : sa suppression est facile (cas 2)

(cas 2 : si la valeur est sur un nœud qui n'a qu'un enfant, il suffit de remplacer ce nœud par son enfant)

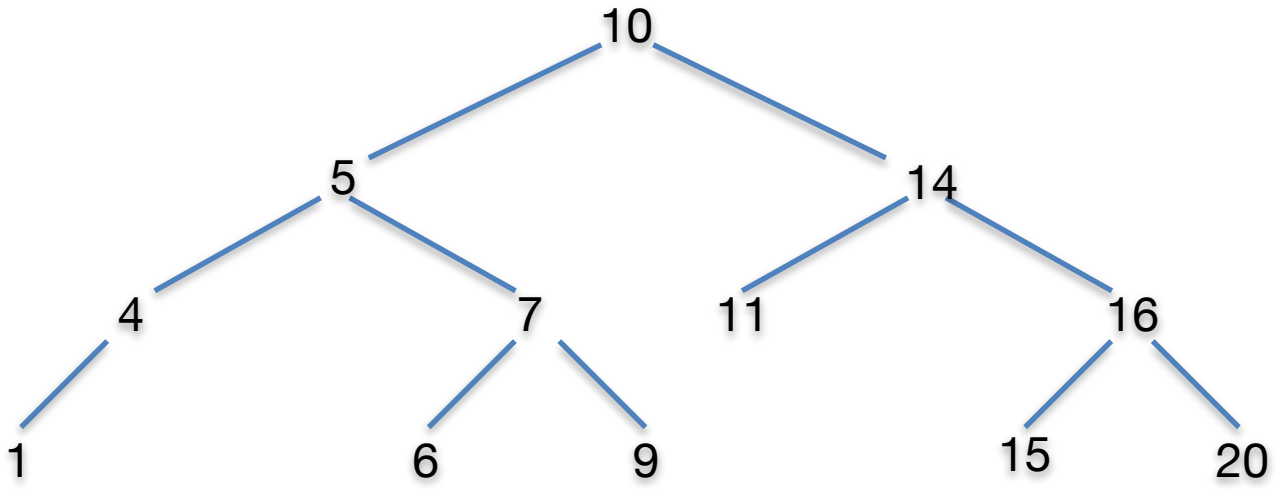


Exemple : essayons de supprimer la valeur 12 :

- une fois la valeur de remplacement (14) elle-même supprimée, on peut l'insérer à la place de 12



Activité Web : essayons des suppressions dans des ABR



wooclap

Question : est-ce que la suppression de la valeur 14 (pour remplacer 12) peut interférer avec la procédure en cours de remplacement de 12 ? Pourquoi ?

Question : pourquoi est-ce qu'on n'appliquera jamais le cas 3 à une valeur remplaçant la valeur qu'on cherche initialement à supprimer ?

Question : pouvez-vous en déduire la complexité des opérations restantes quand on vient de trouver la valeur de remplacement ?

Question : pouvez-vous en déduire le nombre maximum d'appels récursifs qui peut être nécessaire pour la suppression d'une valeur initiale ?

Exercice :écrivez l'algorithme de **suppression** d'une valeur dans un arbre binaire de recherche

Remerciements & Références

C. Fiorio

OpenDSA : https://opendsa-server.cs.vt.edu

Unisciel : http://ressources.unisciel.fr

Introduction à l'algorithmique, Cormen, Leiserson, Rivest, Dunod

Exercices et problèmes d'algorithmique, Baynat, Chrétienne, Hanen, Kedad-Sidhoum,Munier-Kordon,Picouleau, Dunod

Mastering Swift 5, Hoffman, Packt

Images : FAVPNG, …