Chapitre 4

Arbres binaires particuliers

Introduction

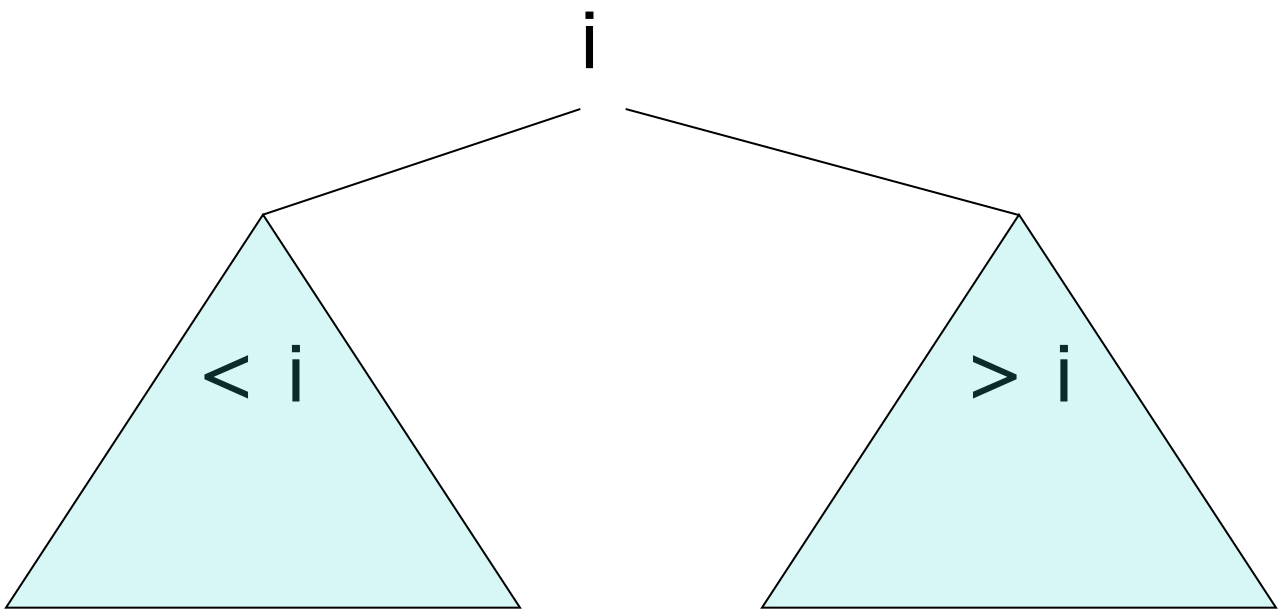
Certains arbres binaires possèdent des propriétés particulières qui peuvent être exploitées pour améliorer l'efficacité des traitements (recherche, insertion) ou pour effectuer des algorithmes spécifiques.

Par exemple:

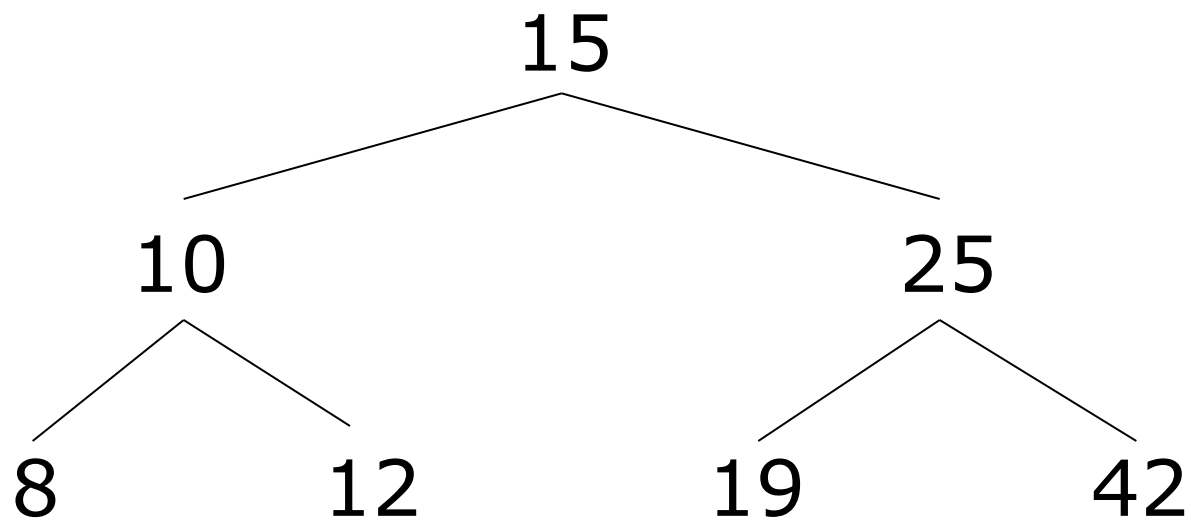
- les arbres binaires de recherche (ABR)
- les tas

1) Arbres binaires de recherche

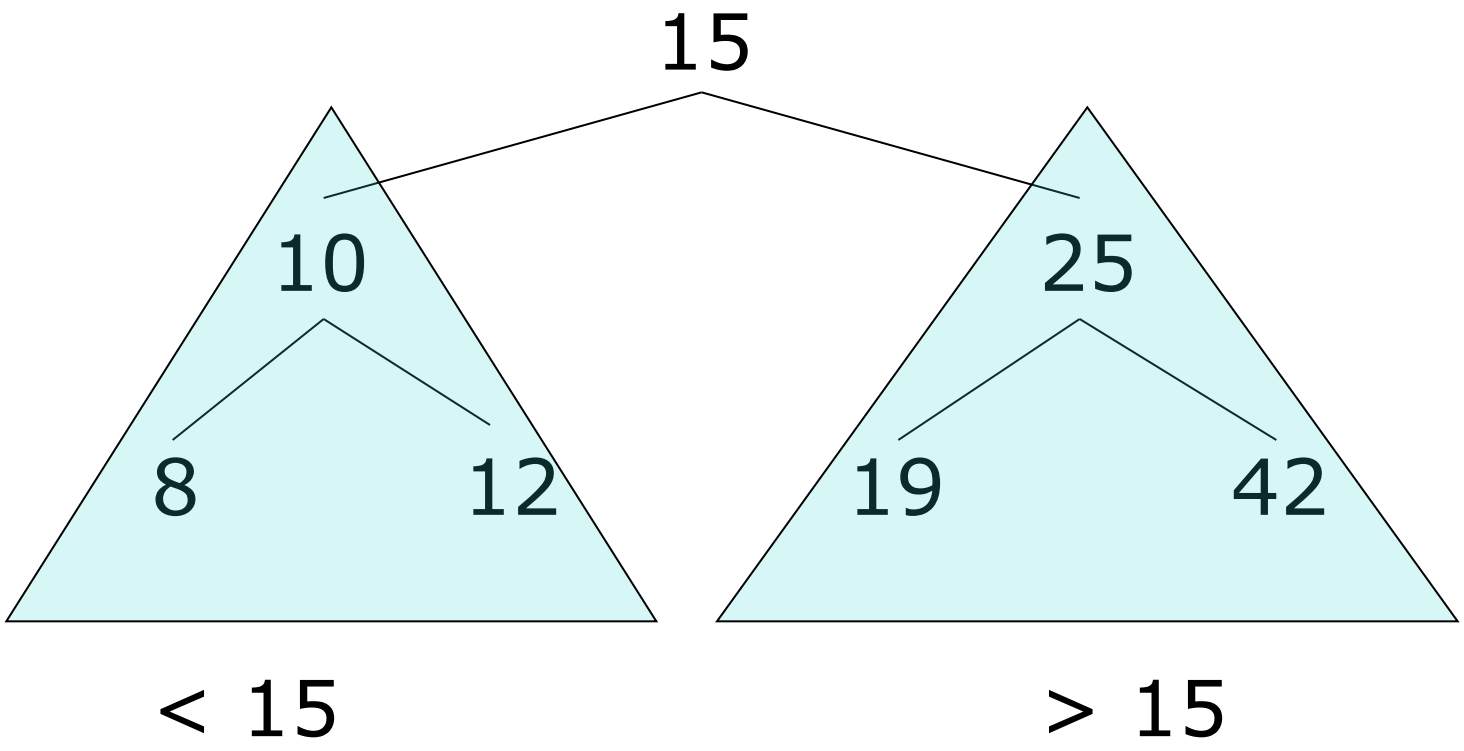
Un arbre dont les valeurs sont toutes différentes et tel que



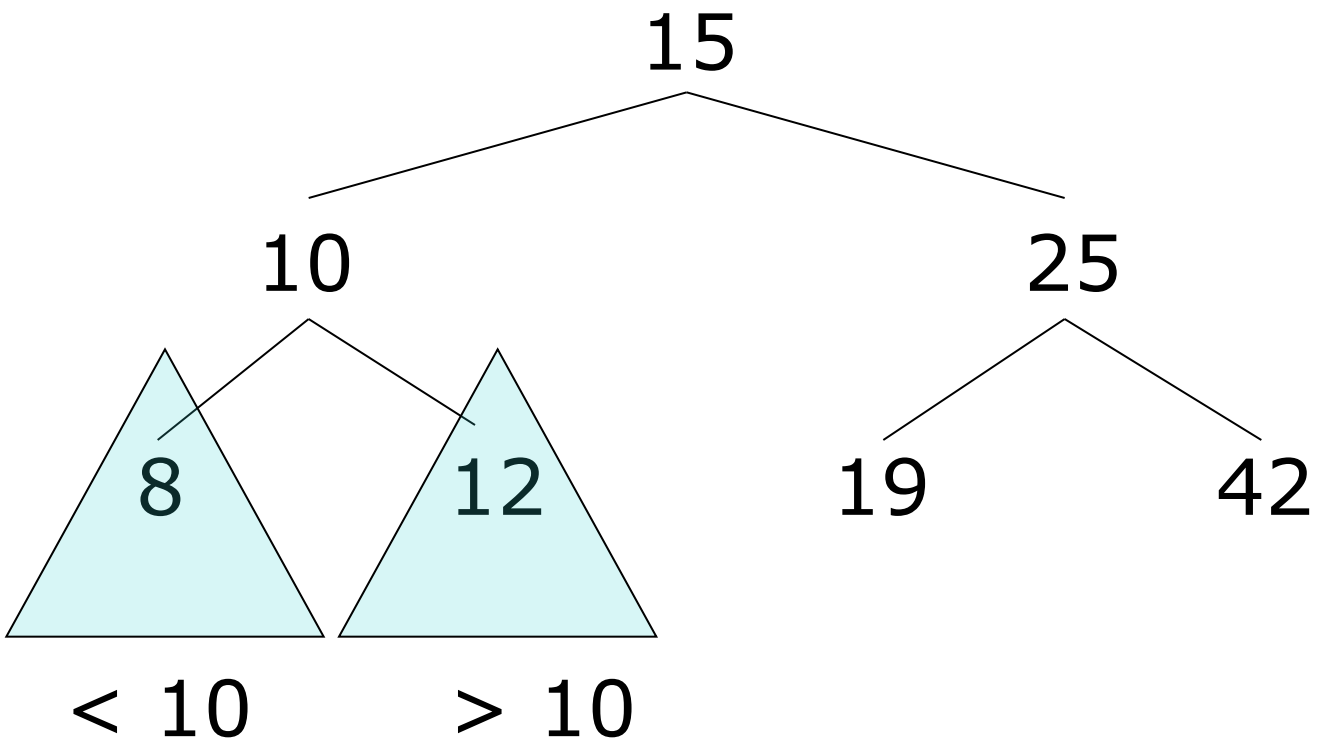
Exemple : un ABR d'entiers



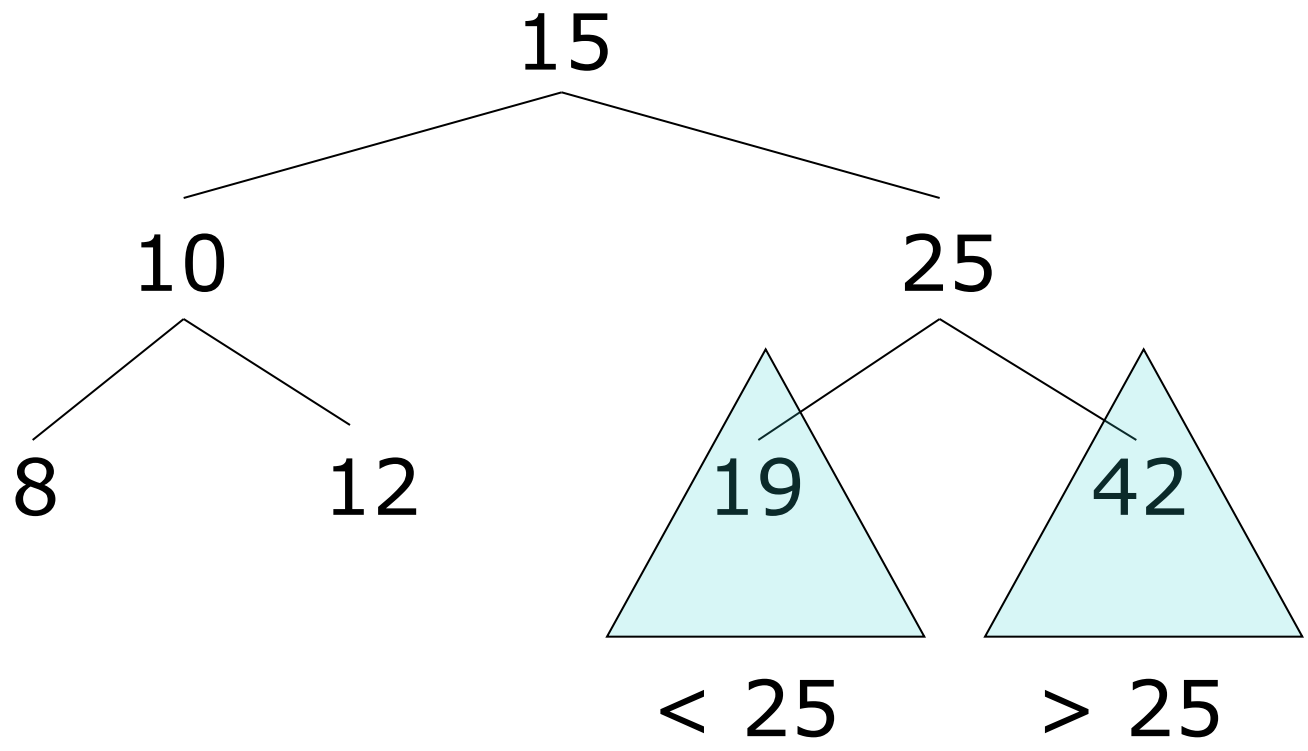
Exemple



Exemple



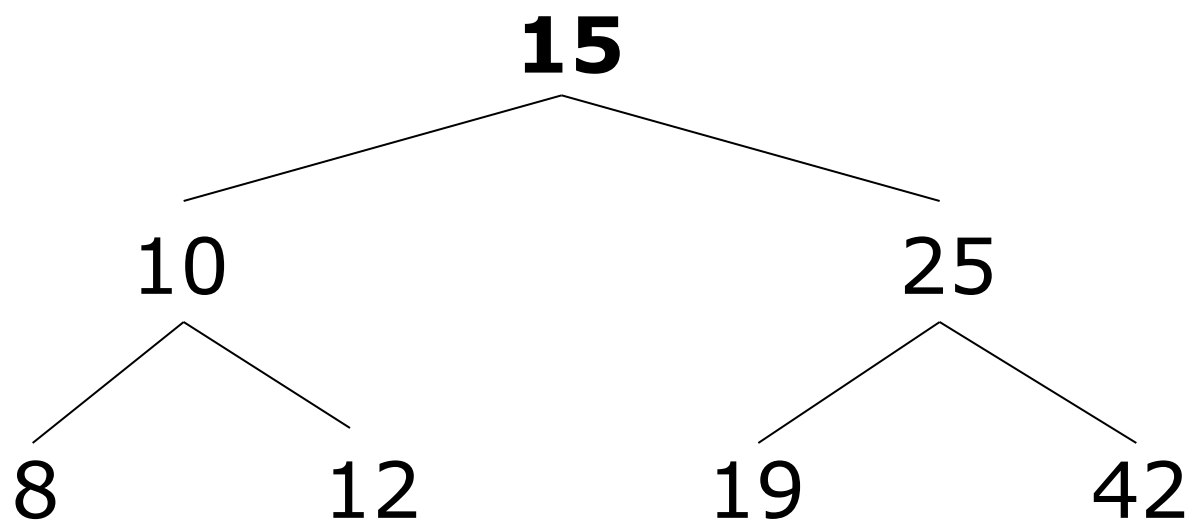
Exemple



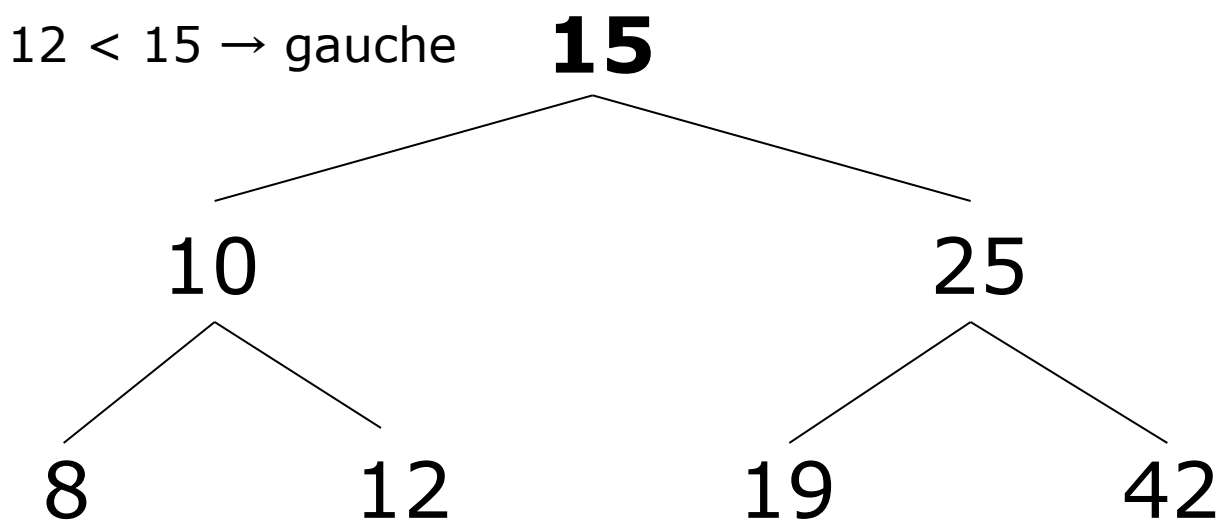
Intérêt des ABR

La recherche et l'insertion d'une valeur nécessitent de parcourir *une seule branche* de l'arbre au lieu de sa totalité

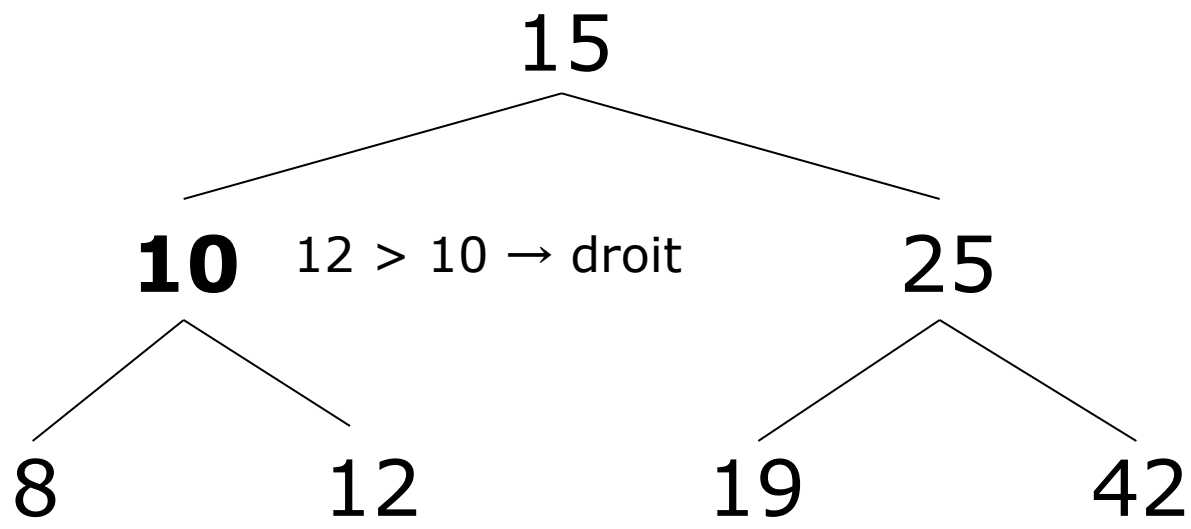
Exemple : recherche de 12



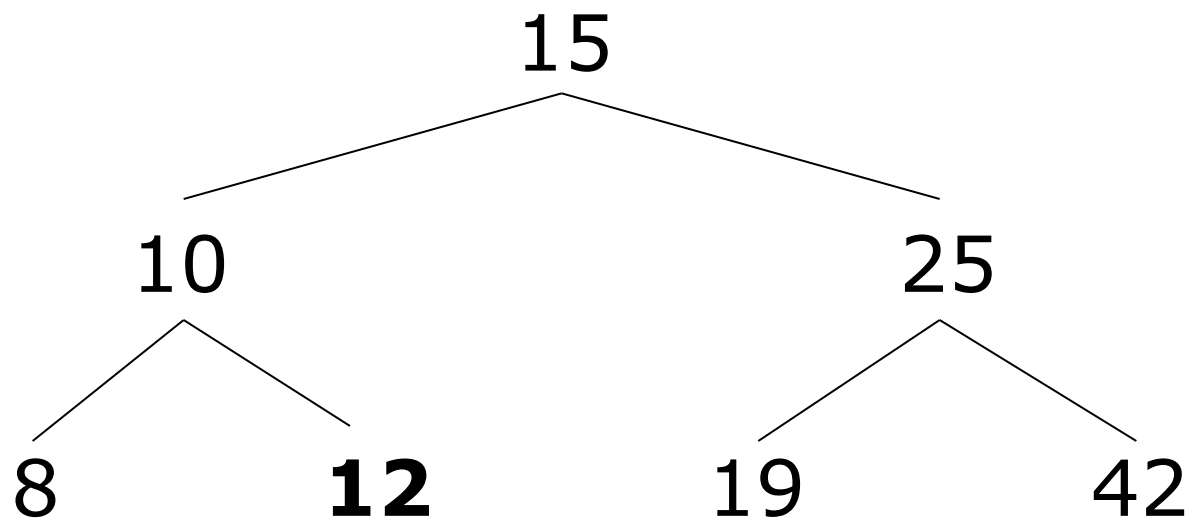
Exemple : recherche de 12



Exemple : recherche de 12

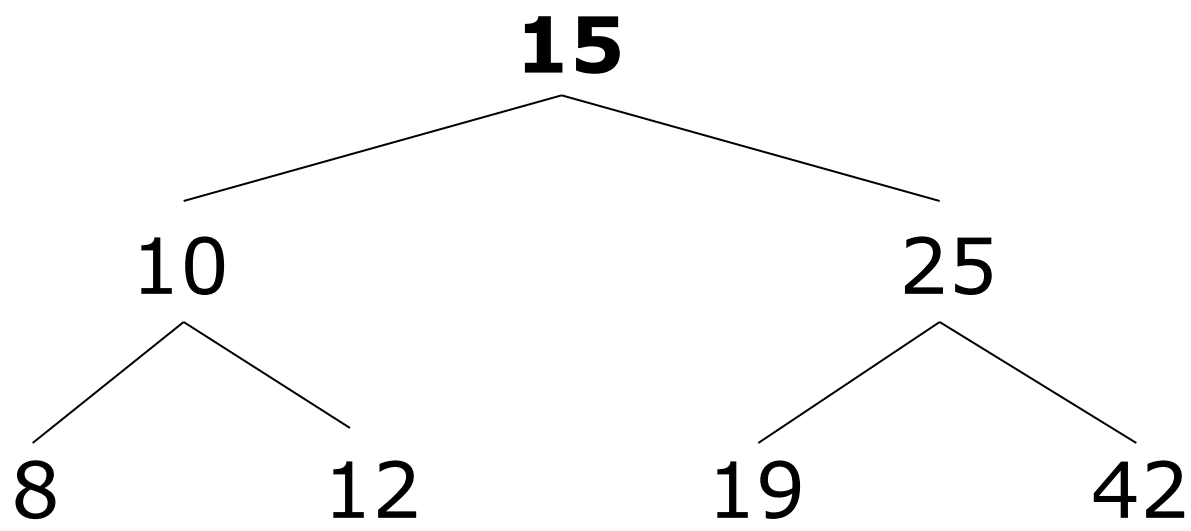


Exemple : recherche de 12

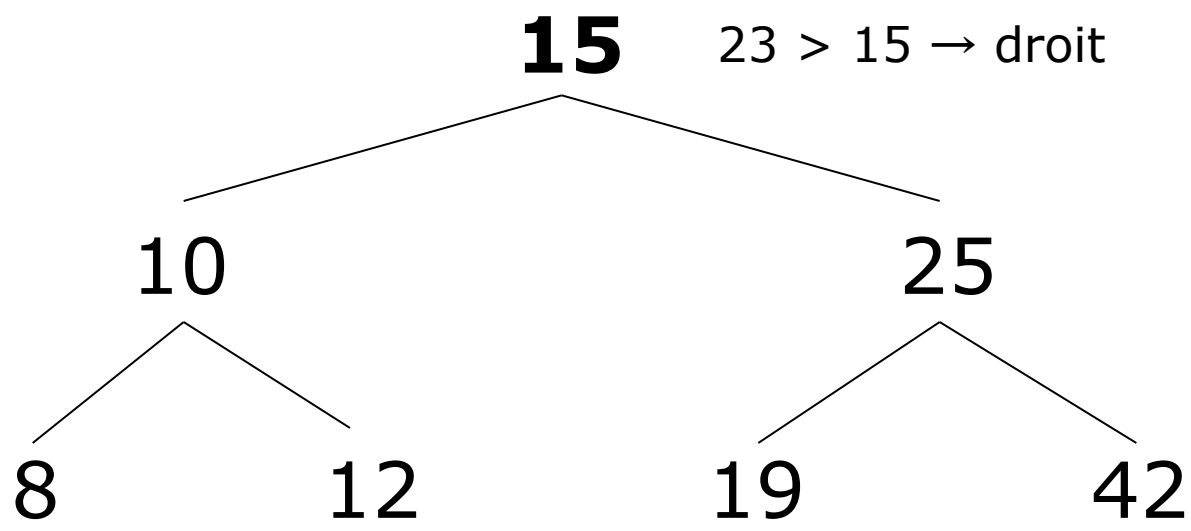


12 = 12 → présent

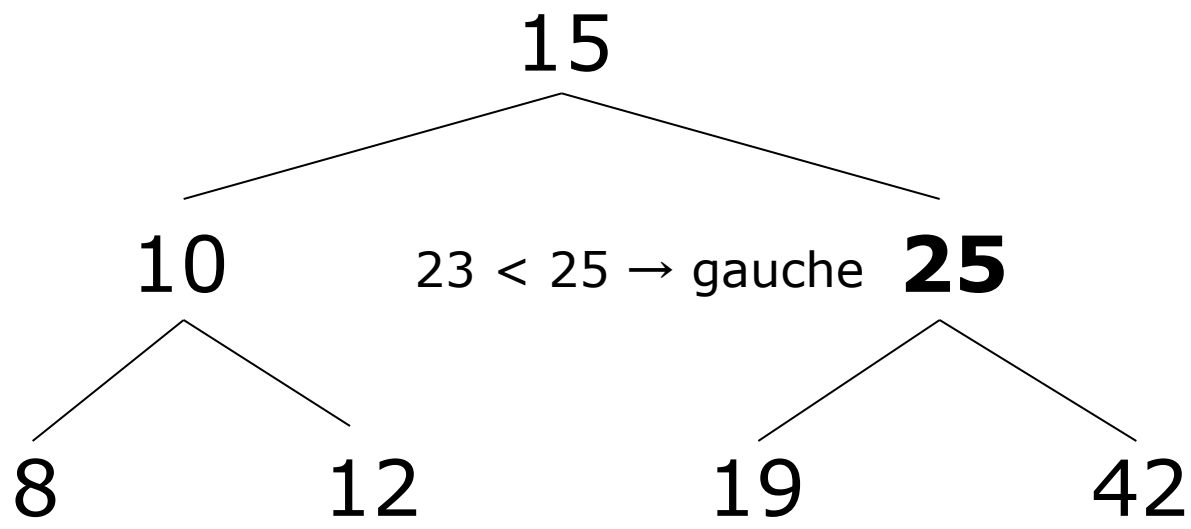
Exemple : recherche de 23



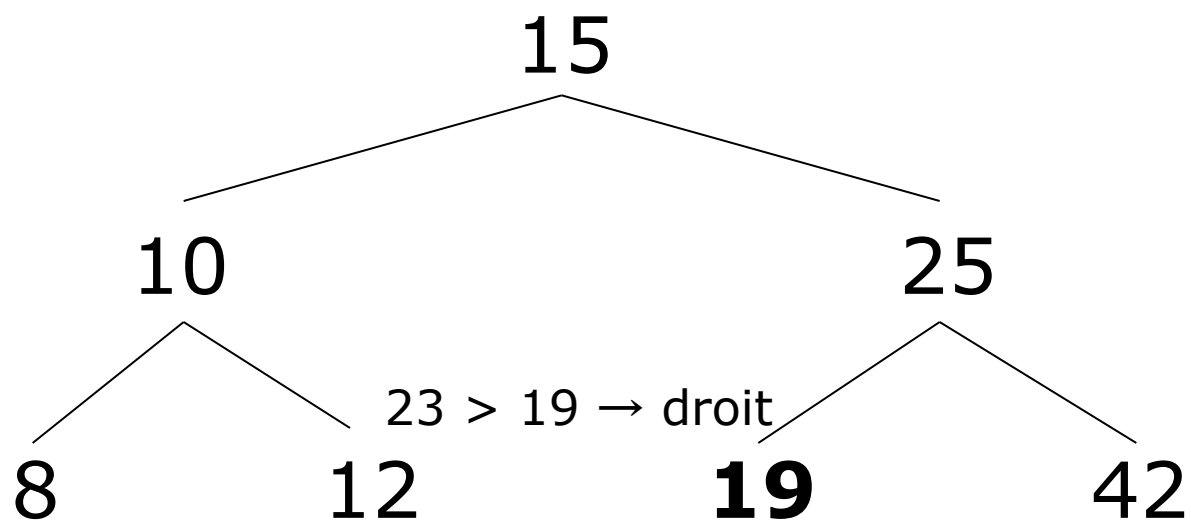
Exemple : recherche de 23



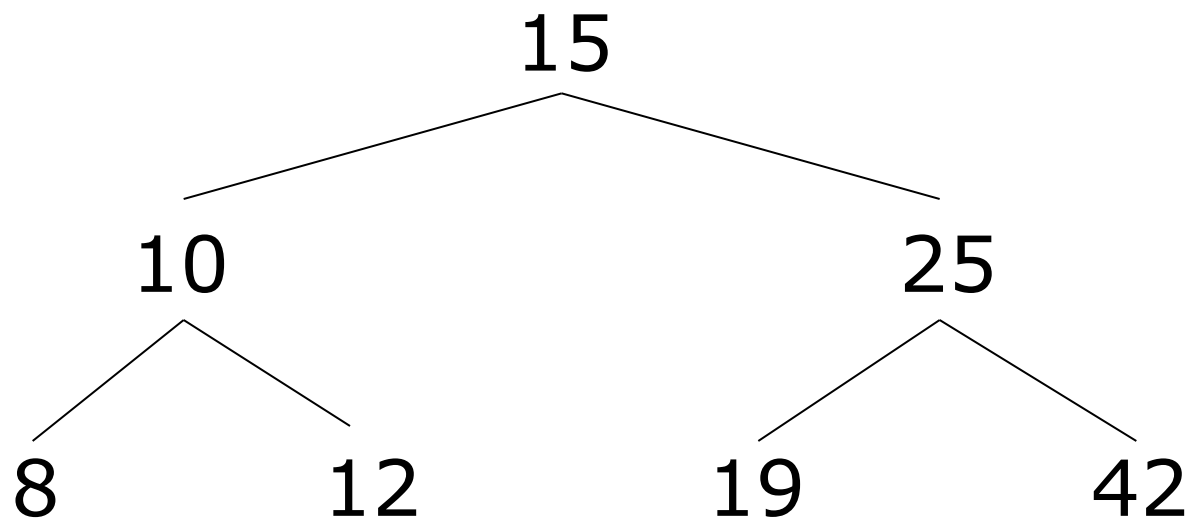
Exemple : recherche de 23



Exemple : recherche de 23

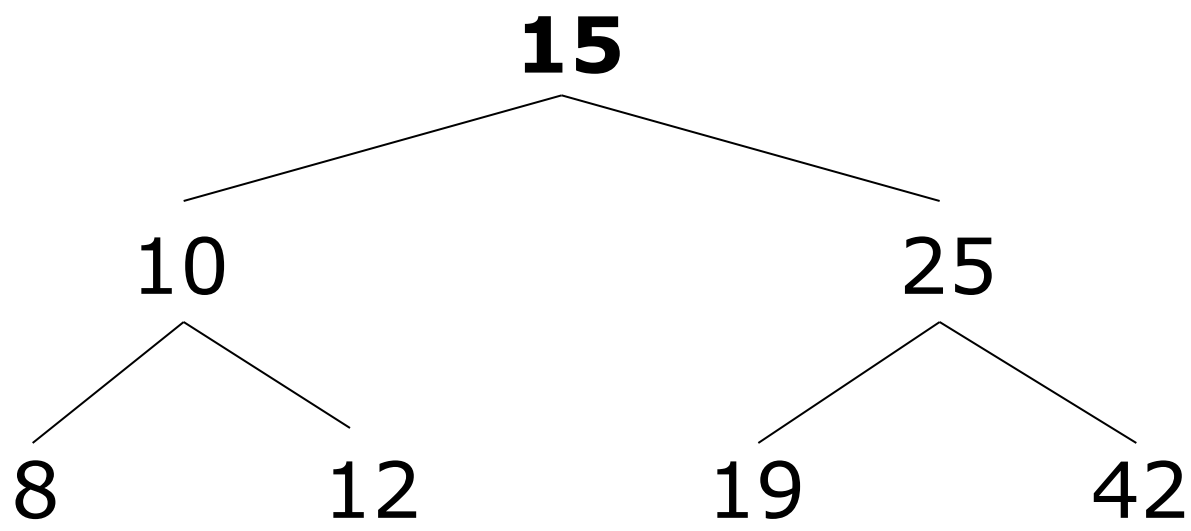


Exemple : recherche de 23

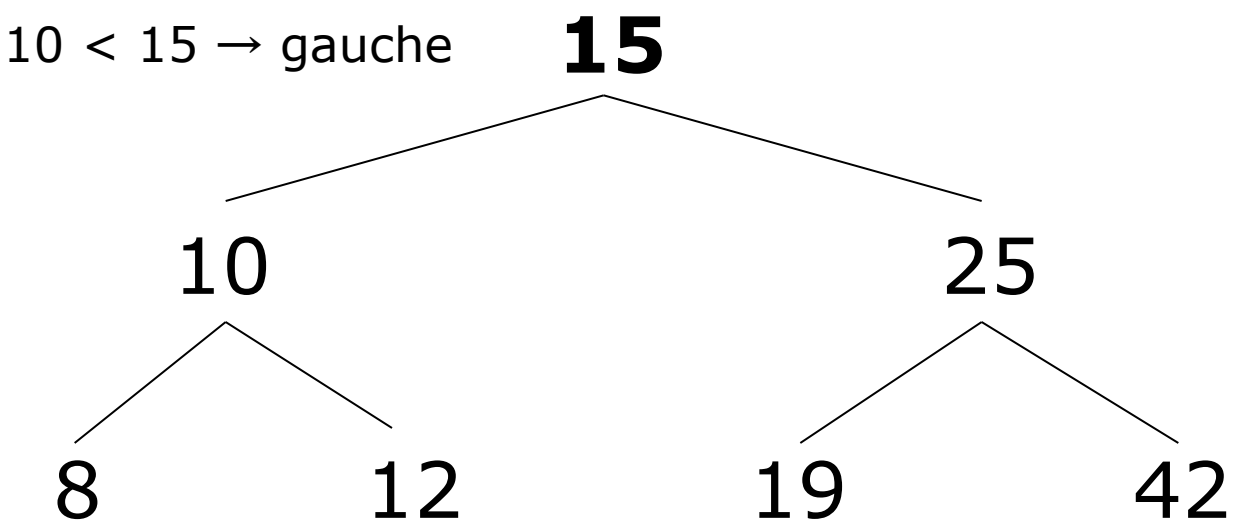


hors arbre → 23 absent

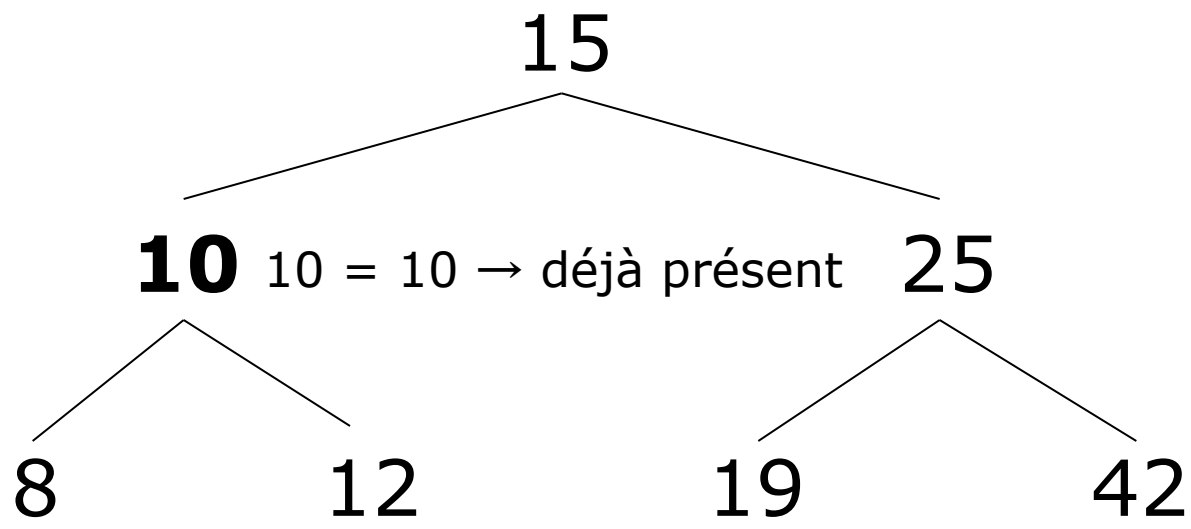
Exemple : insertion de 10



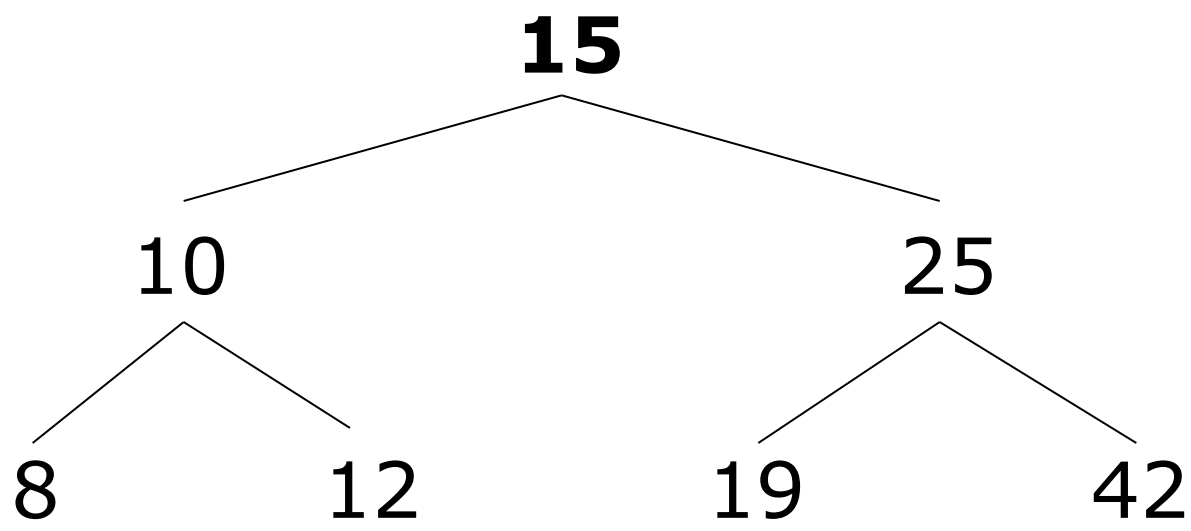
Exemple : insertion de 10



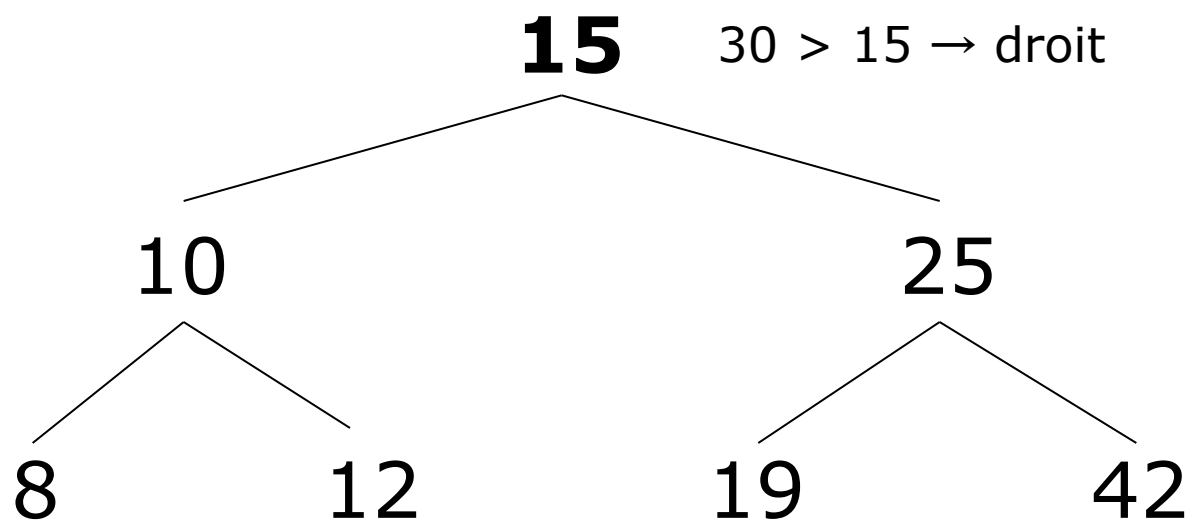
Exemple : insertion de 10



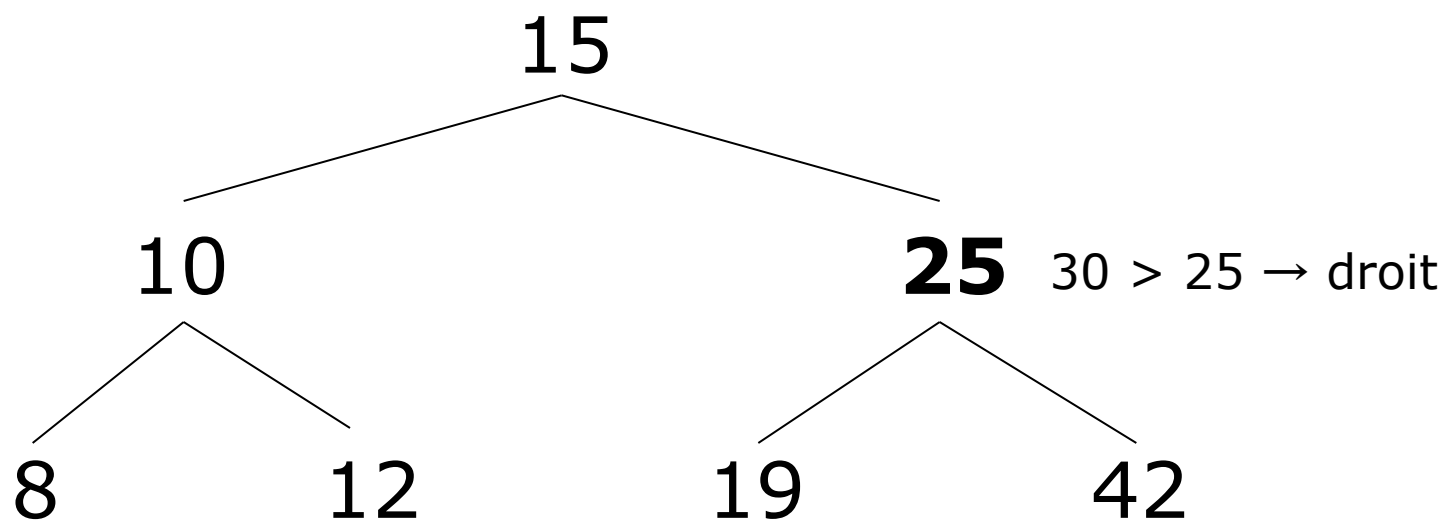
Exemple : insertion de 30



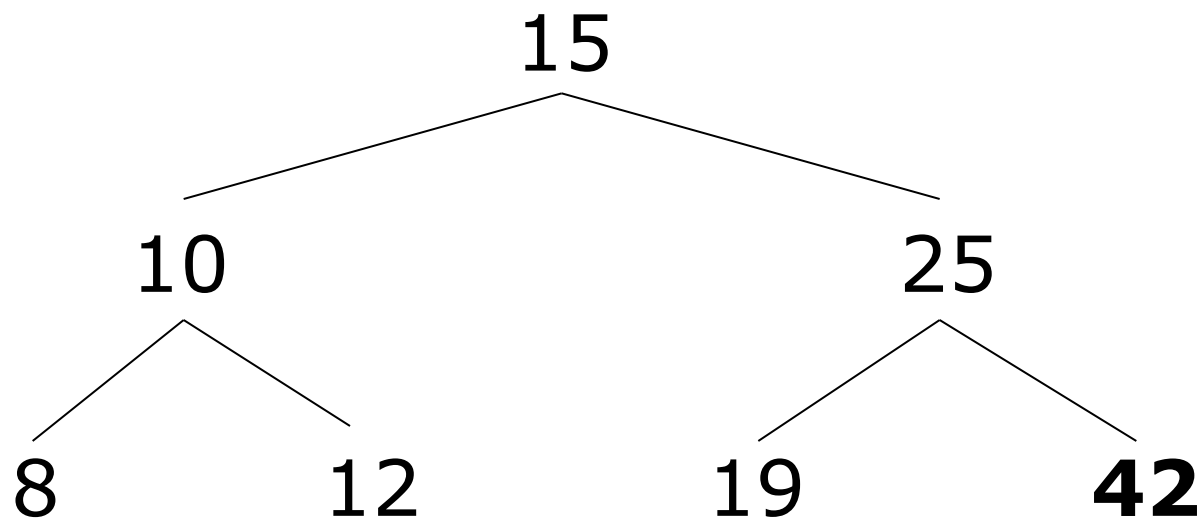
Exemple : insertion de 30



Exemple : insertion de 30

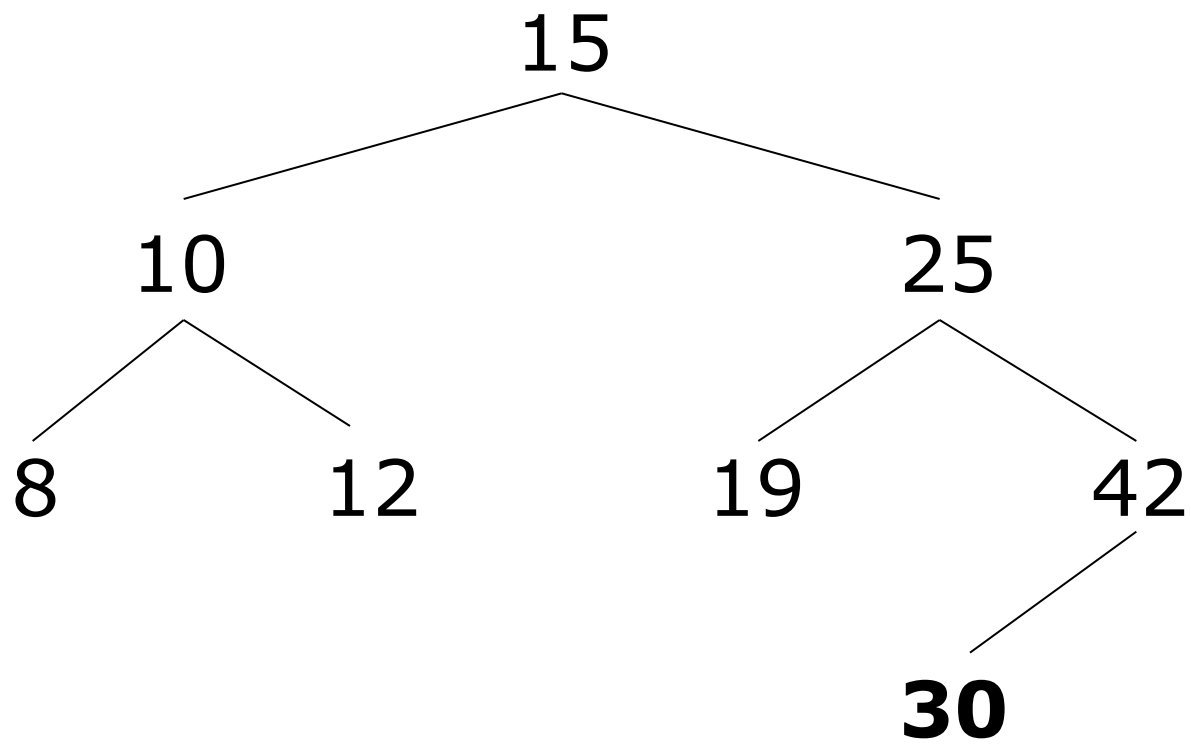


Exemple : insertion de 30



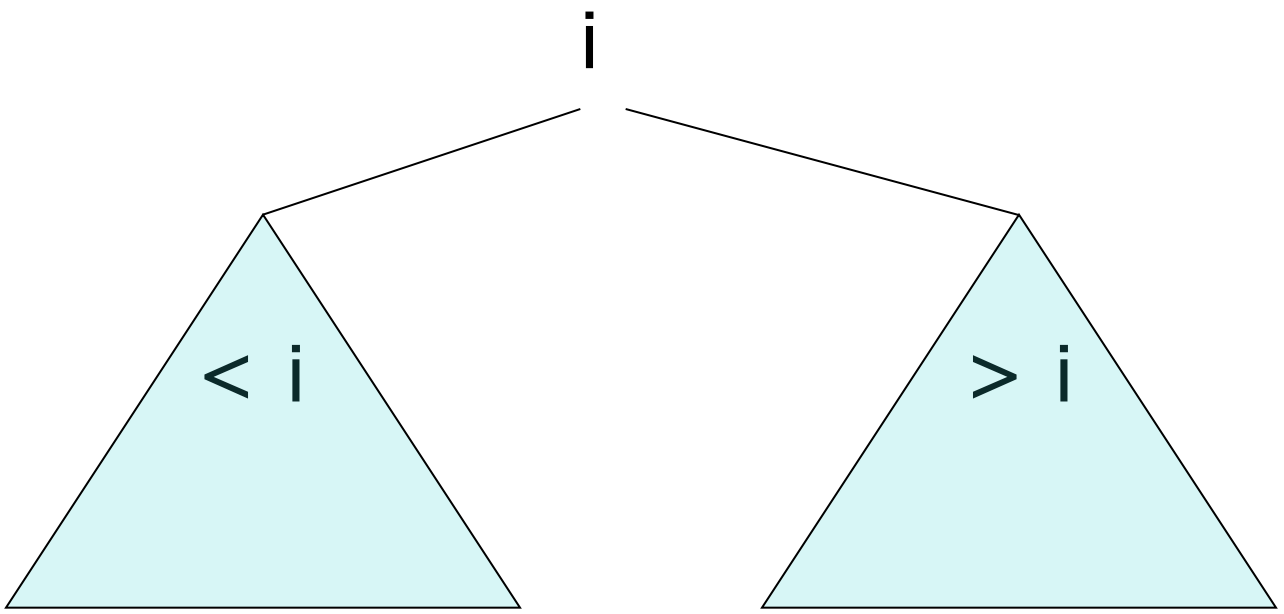
30 < 42 & pas de filsg
→ insertion à gauche

Exemple : insertion de 30



Suppression d'un nœud

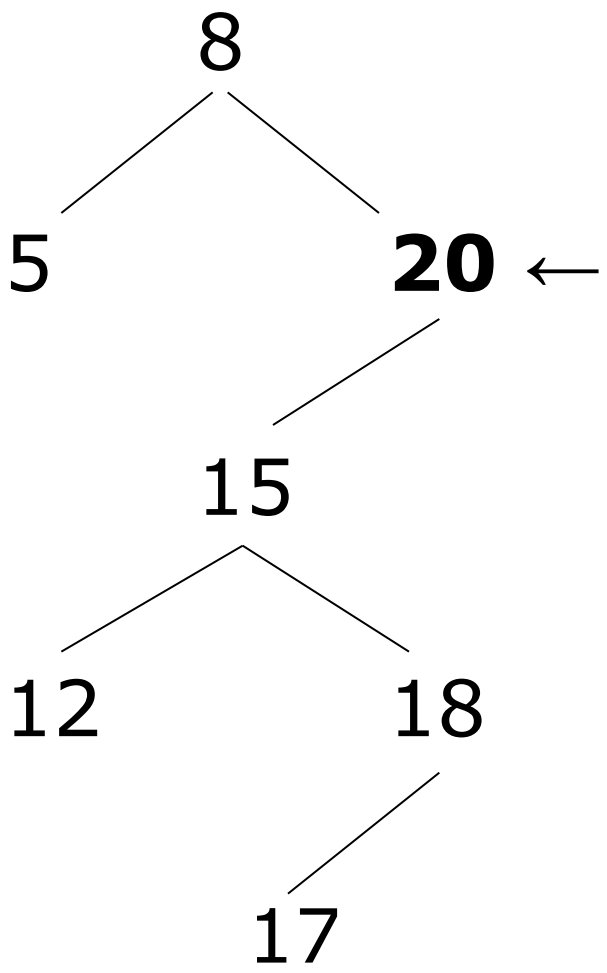
La suppression d'un nœud doit préserver les propriétés de l'arbre



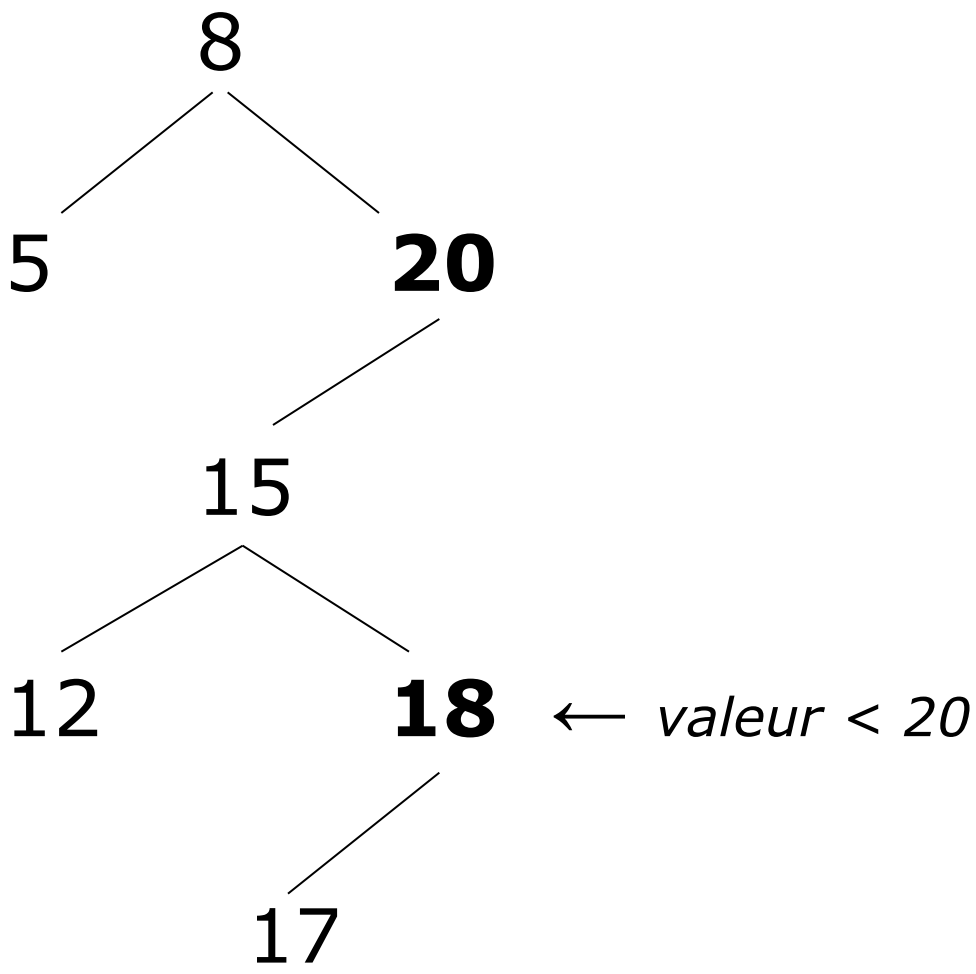
Suppression d'un nœud

- Le nœud est une feuille
  - Supprimer ce noeud
- Le nœud possède au moins un fils
  - Trouver la valeur immédiatement inférieure (ou supérieure)
  - Remplacer le nœud par cette valeur
  - Supprimer cette valeur de la même manière

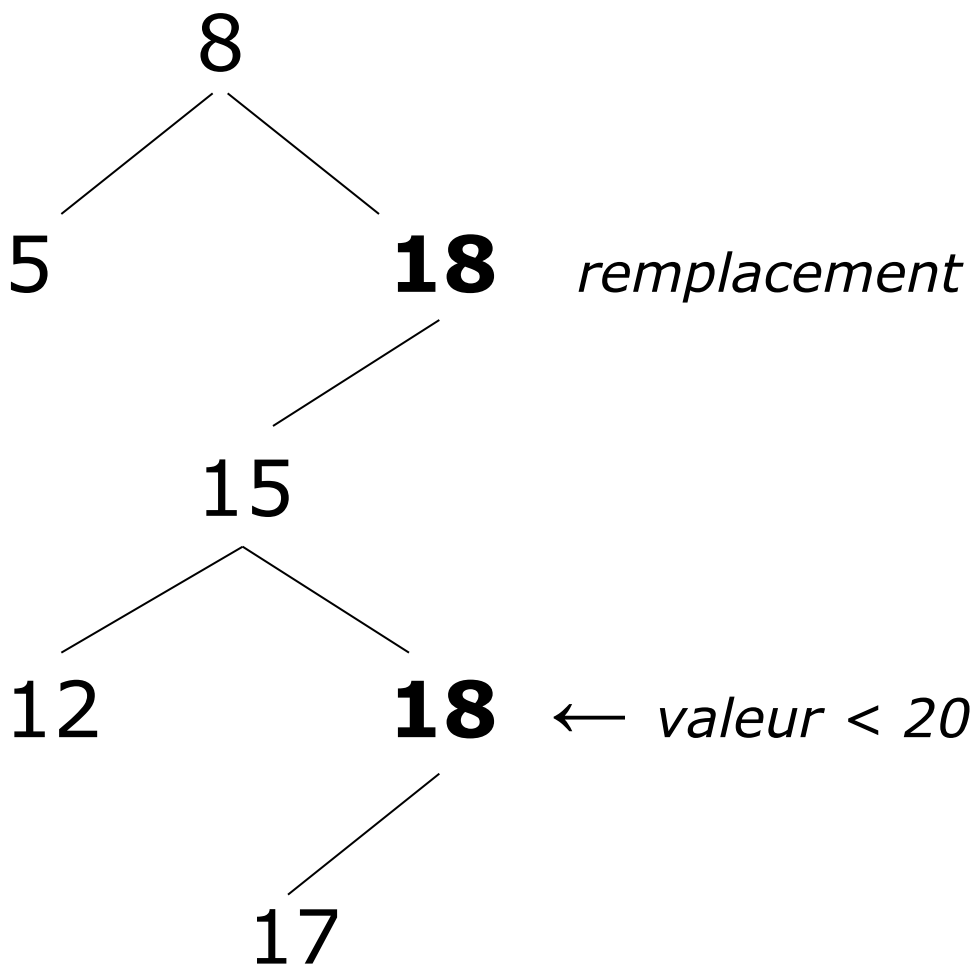
Exemple : suppression de 20



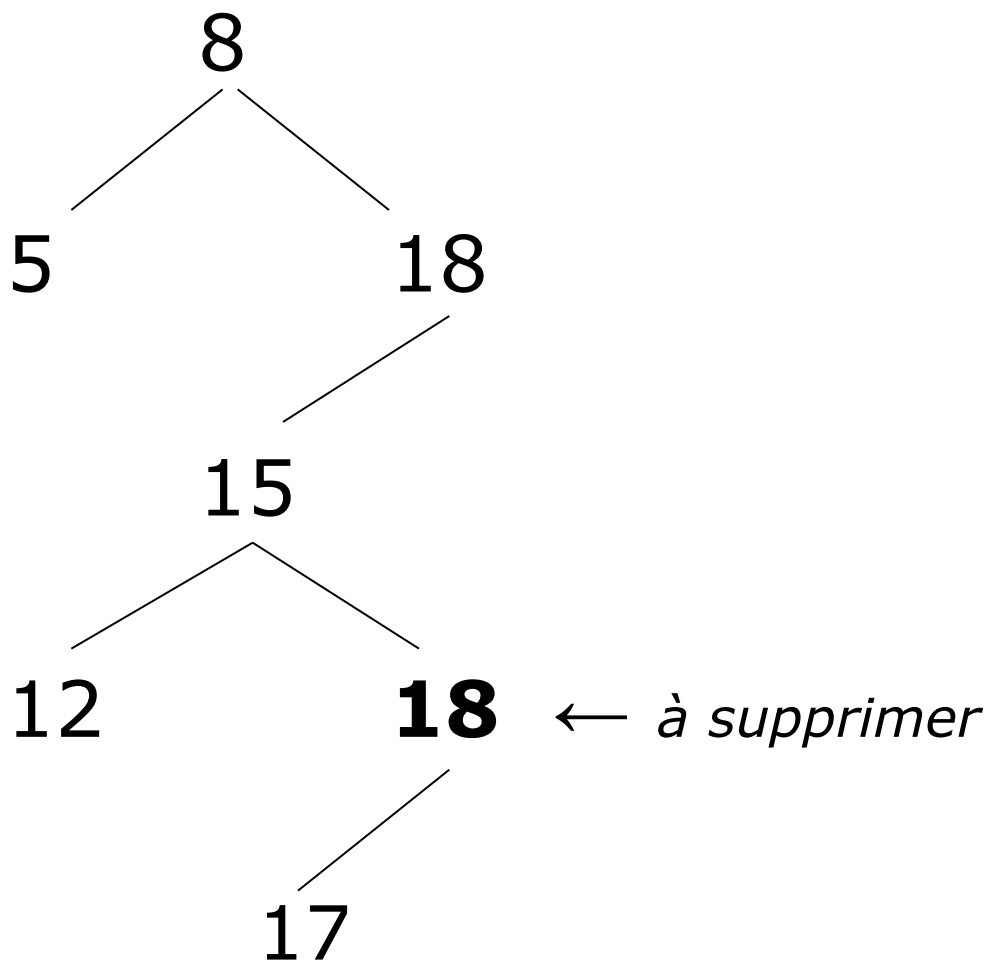
Exemple



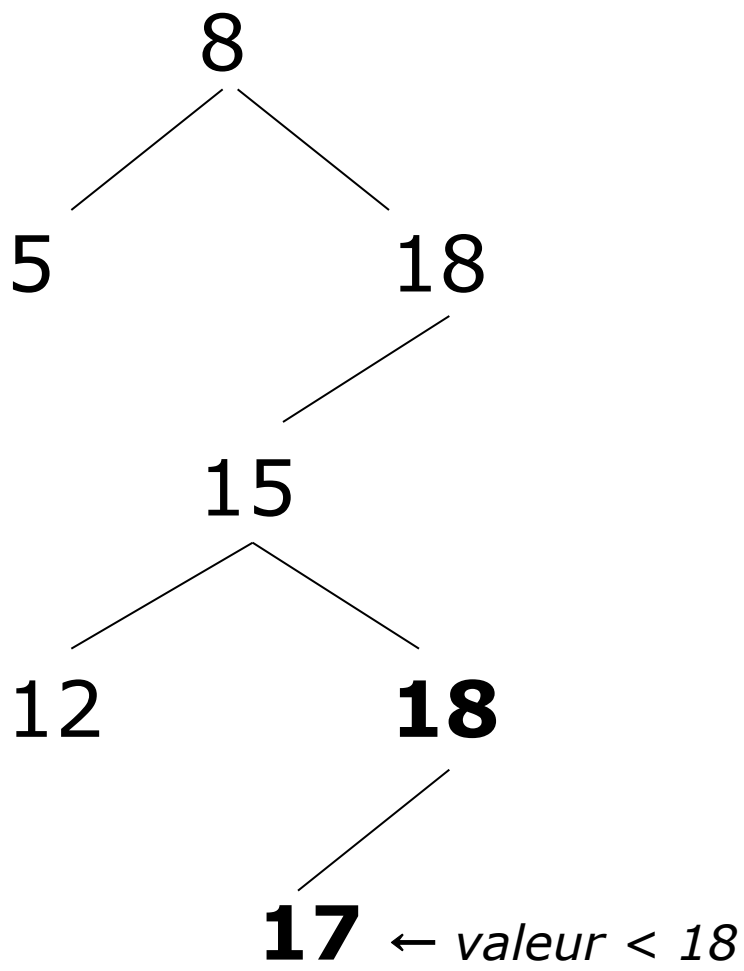
Exemple



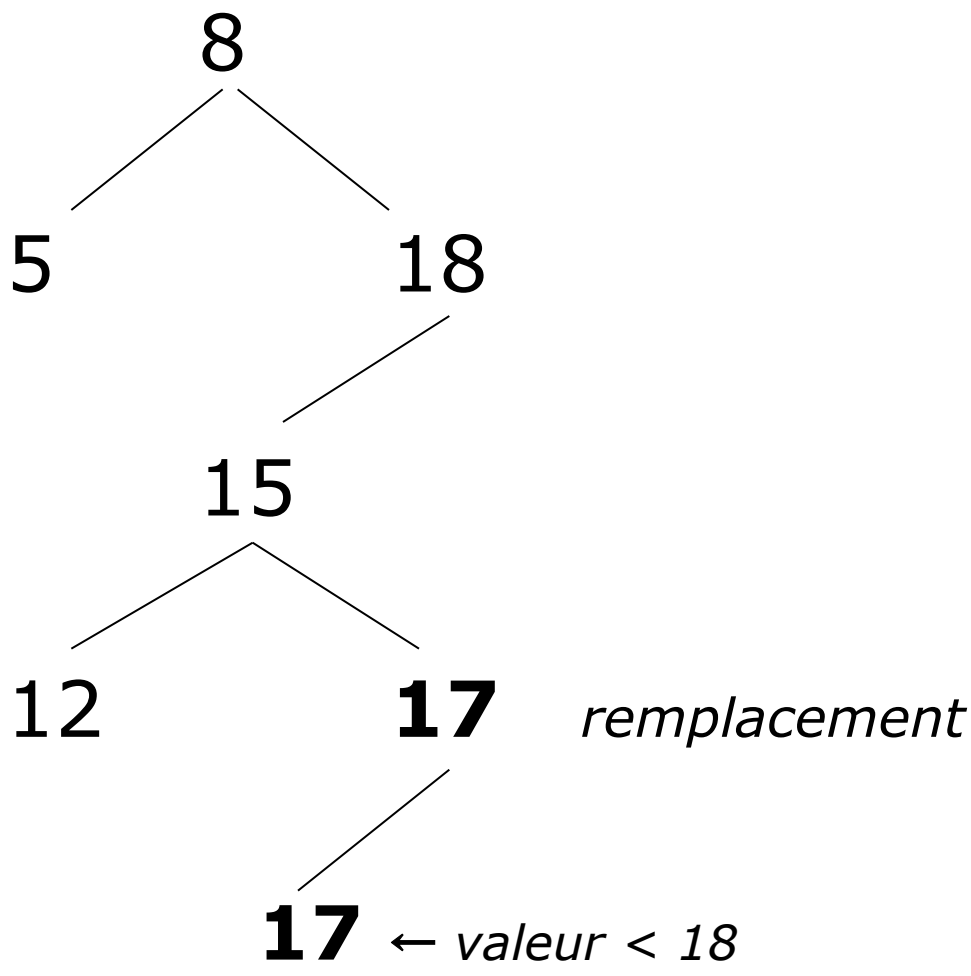
Exemple



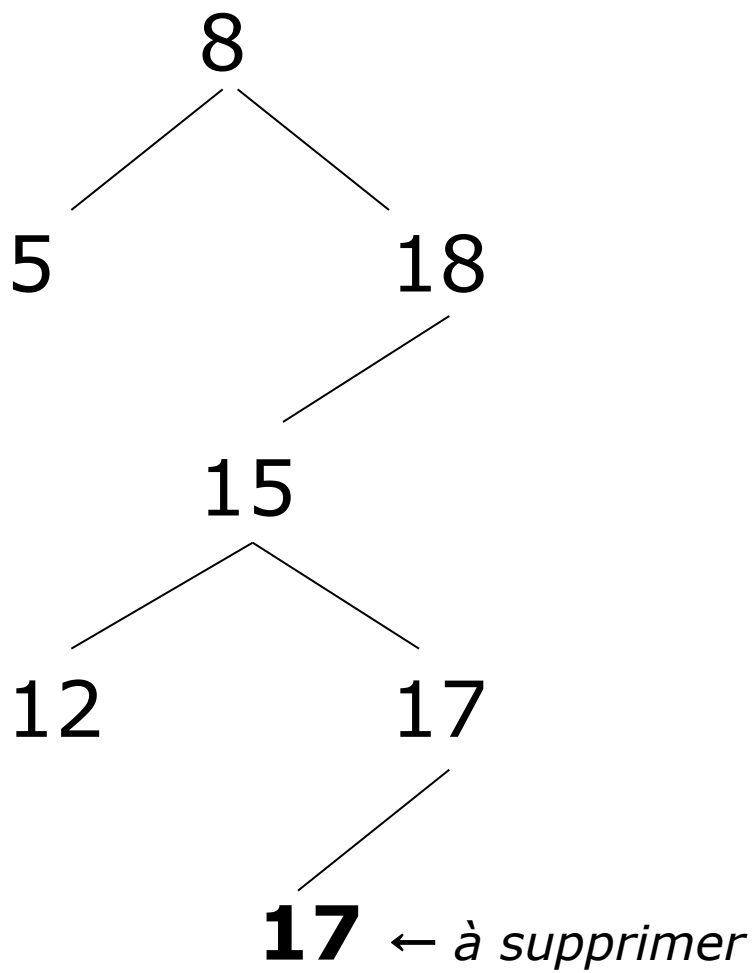
Exemple



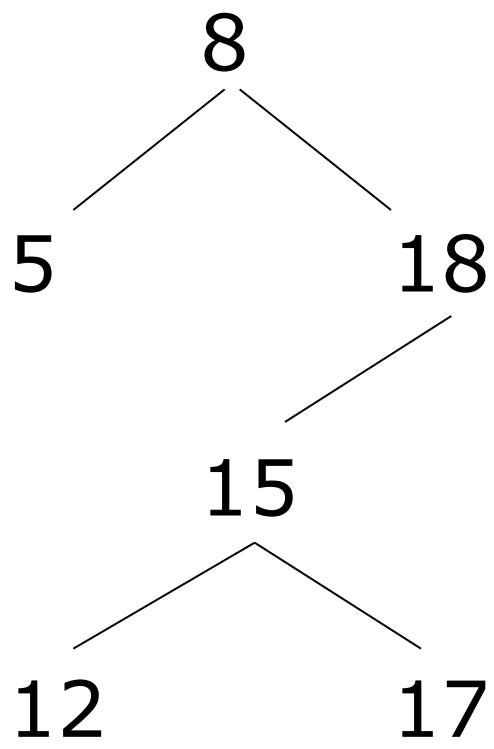
Exemple



Exemple



Exemple : arbre résultat



Chapitre 4 suite

Arbres complets / parfaits

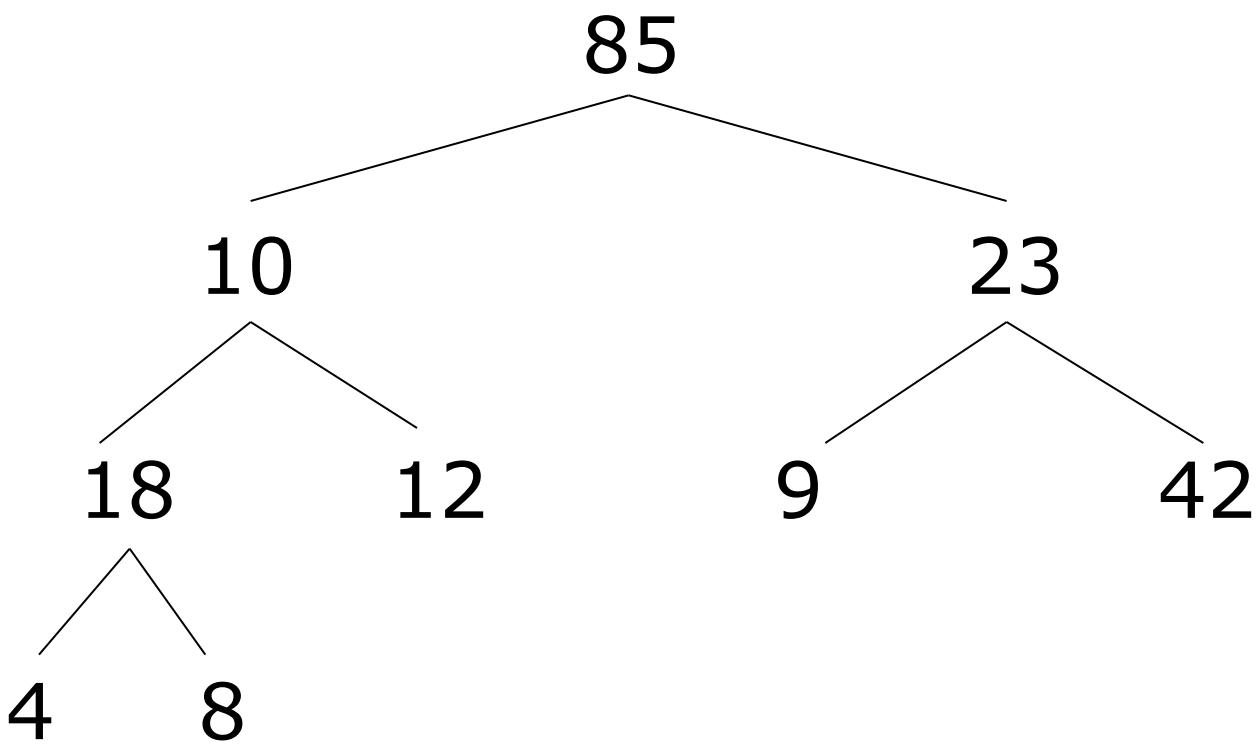
2) Arbres complets et parfaits

Un arbre binaire **complet** est un arbre binaire dont chaque nœud interne a 2 fils.

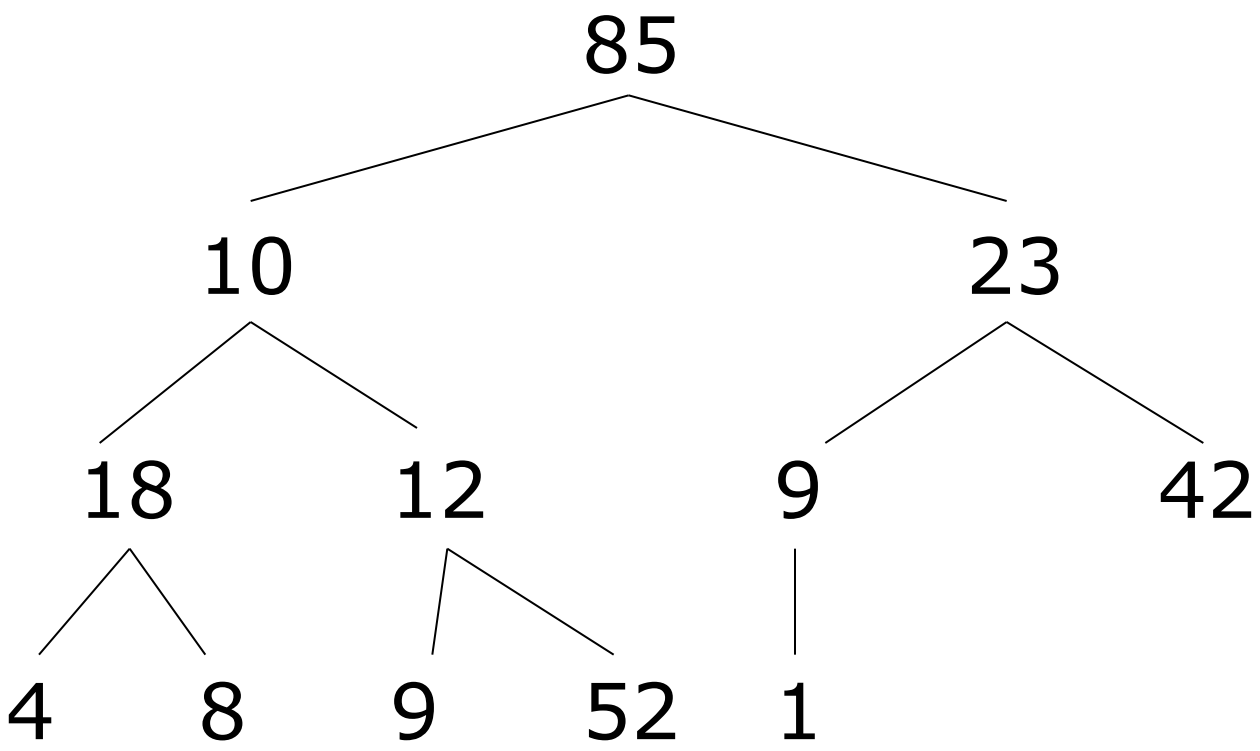
Un arbre binaire **parfait** est un arbre binaire complet dont toutes les feuilles sont à la même hauteur.

Un arbre binaire **quasi-parfait** est un arbre binaire parfait dont le dernier niveau peut être réduit par la droite.

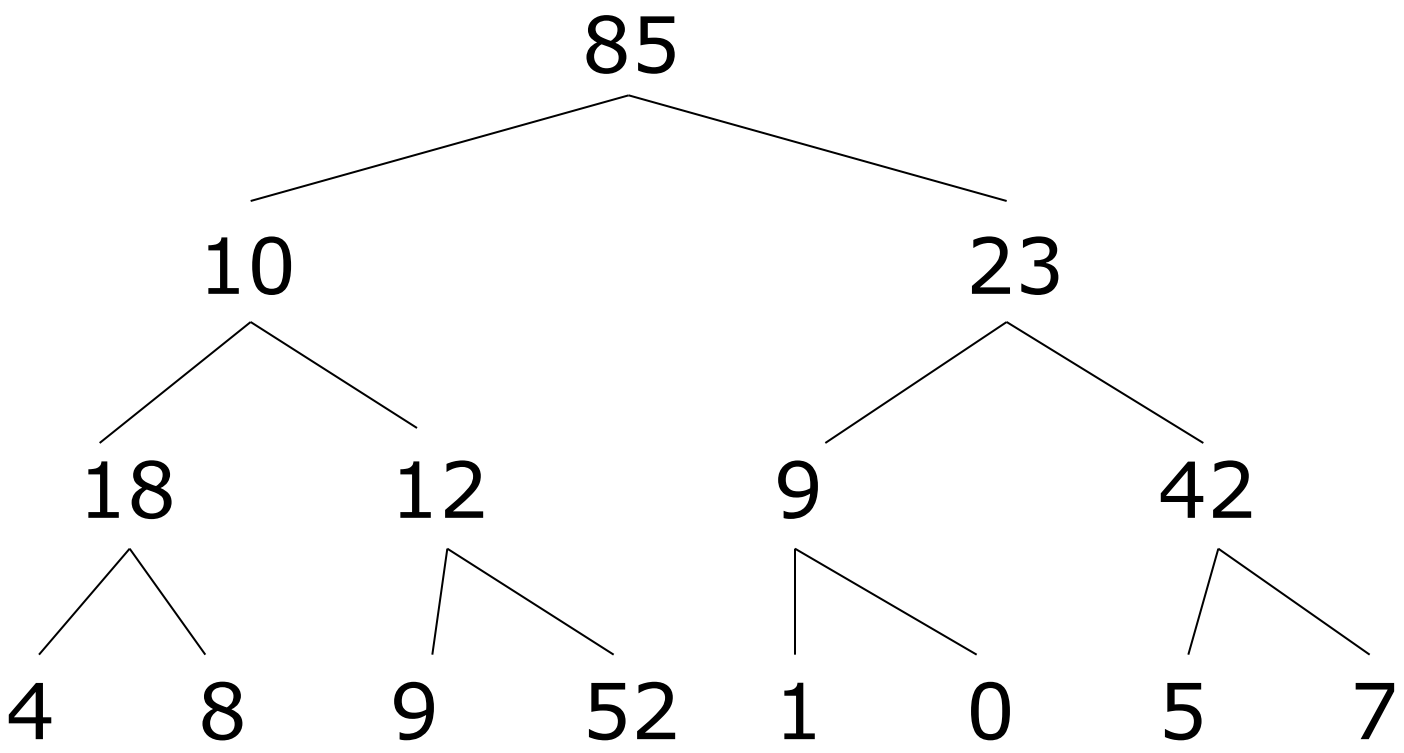
Exemple : arbre complet



Exemple : arbre quasi-parfait



Exemple : arbre parfait



Intérêt des arbres complets

Un arbre complet peut être représenté dans un tableau de manière contiguë sans perte de place

Les accès se font par calcul.

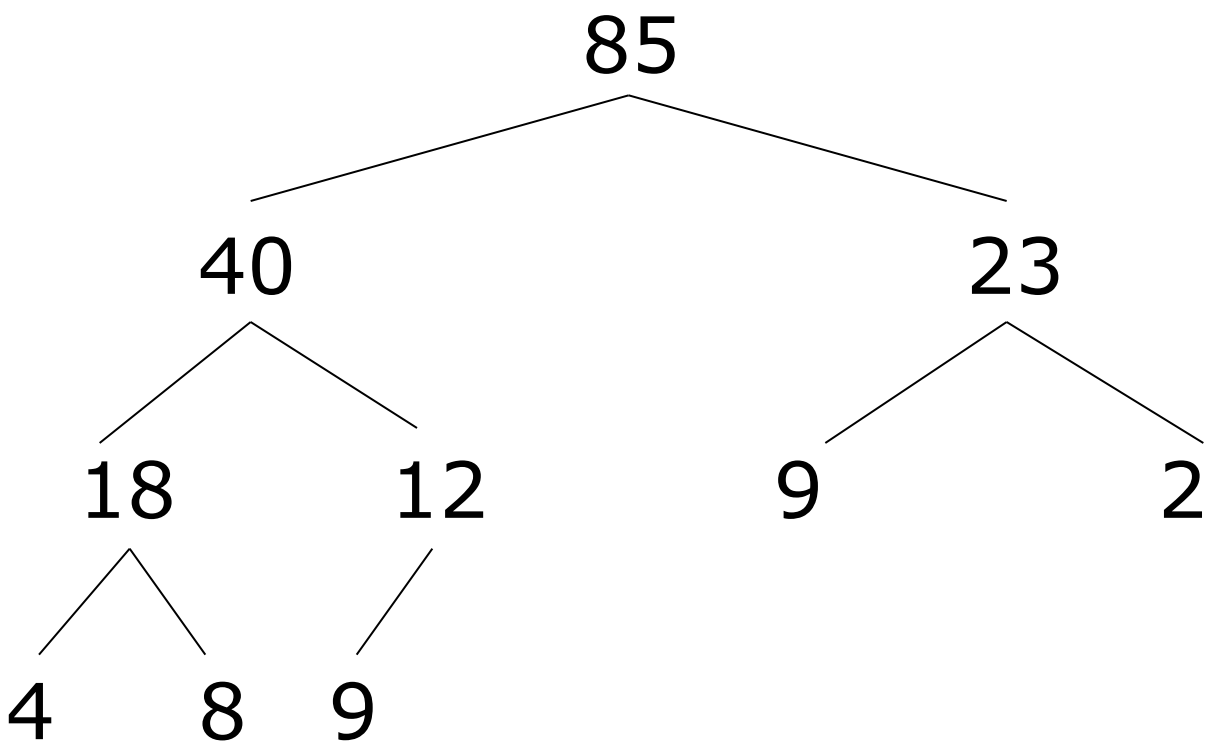
rac	niveau 1		niveau 2				niveau 3		
85	10	23	18	12	9	42	4	8	9

3) Tas min et tas max

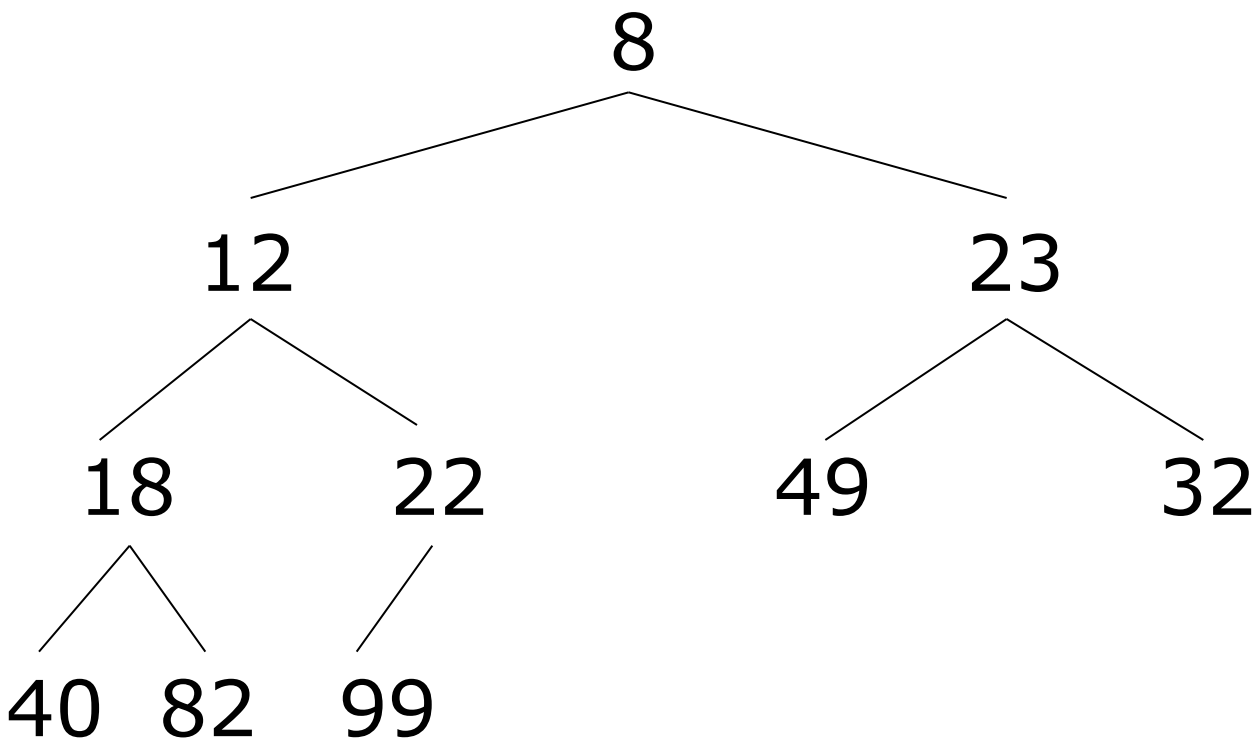
Un **tas max** est un arbre binaire **quasi-parfait** tel que la valeur d'un nœud est ***supérieure*** à celle de tous ses **descendants**

Un **tas min** est un arbre binaire **quasi-parfait** tel que la valeur d'un nœud est ***inférieure*** à celle de tous ses **descendants**

Exemple : un tas max



Exemple : un tas min



Utilisation des tas

Les tas (min ou max) sont utilisés dans le *tri par tas*, en $O(n \log_2 n)$

Deux opérations sont nécessaires :

- ajout d'une valeur quelconque
- retrait de la plus grande (ou plus petite) valeur

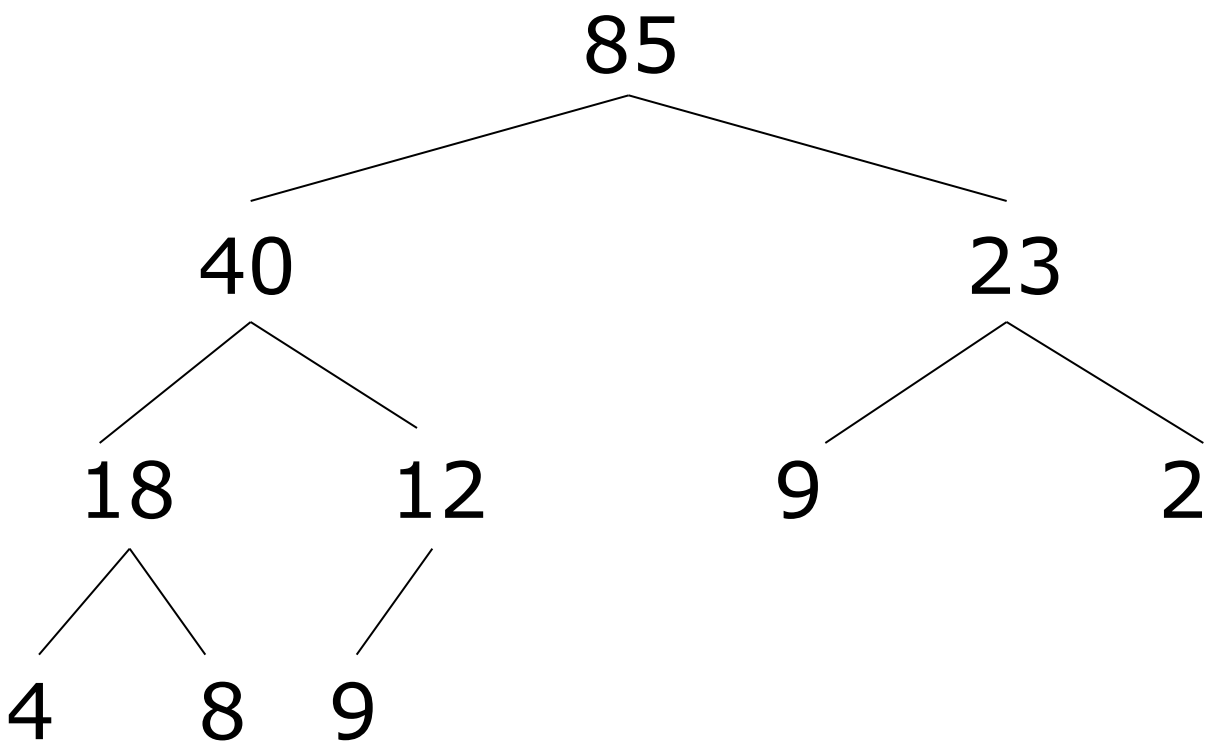
en préservant les propriétés du tas

Ajout dans un tas max

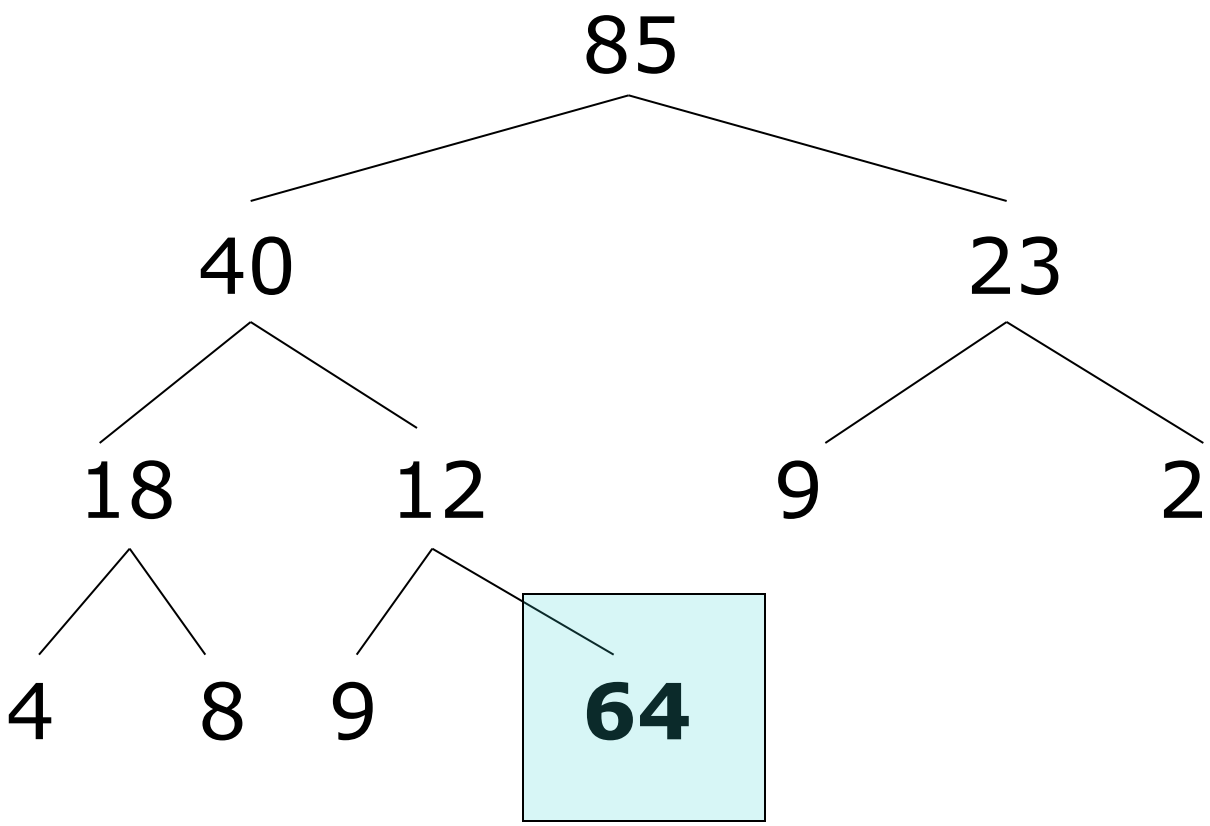
Principe

- La valeur est ajoutée comme feuille la plus à droite du dernier niveau (l'arbre reste complet)
- La valeur est permutée avec la valeur de son père tant qu'elle est supérieure

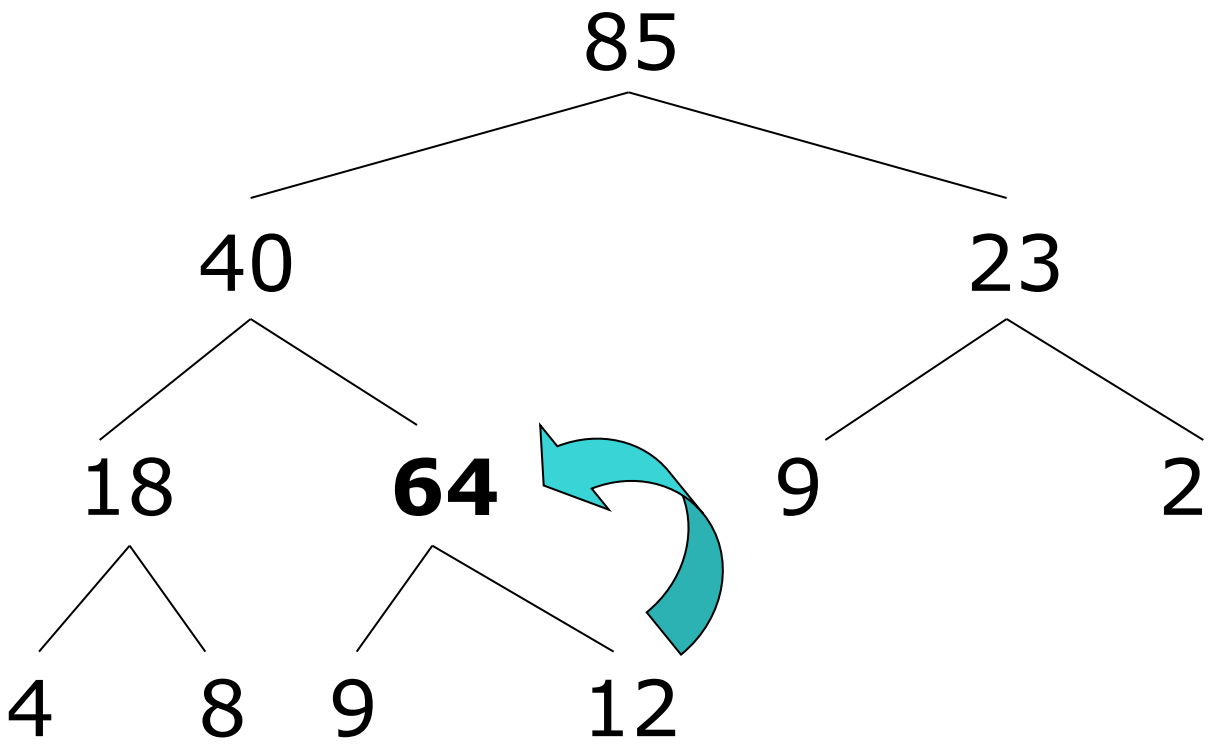
Exemple : ajout de 64



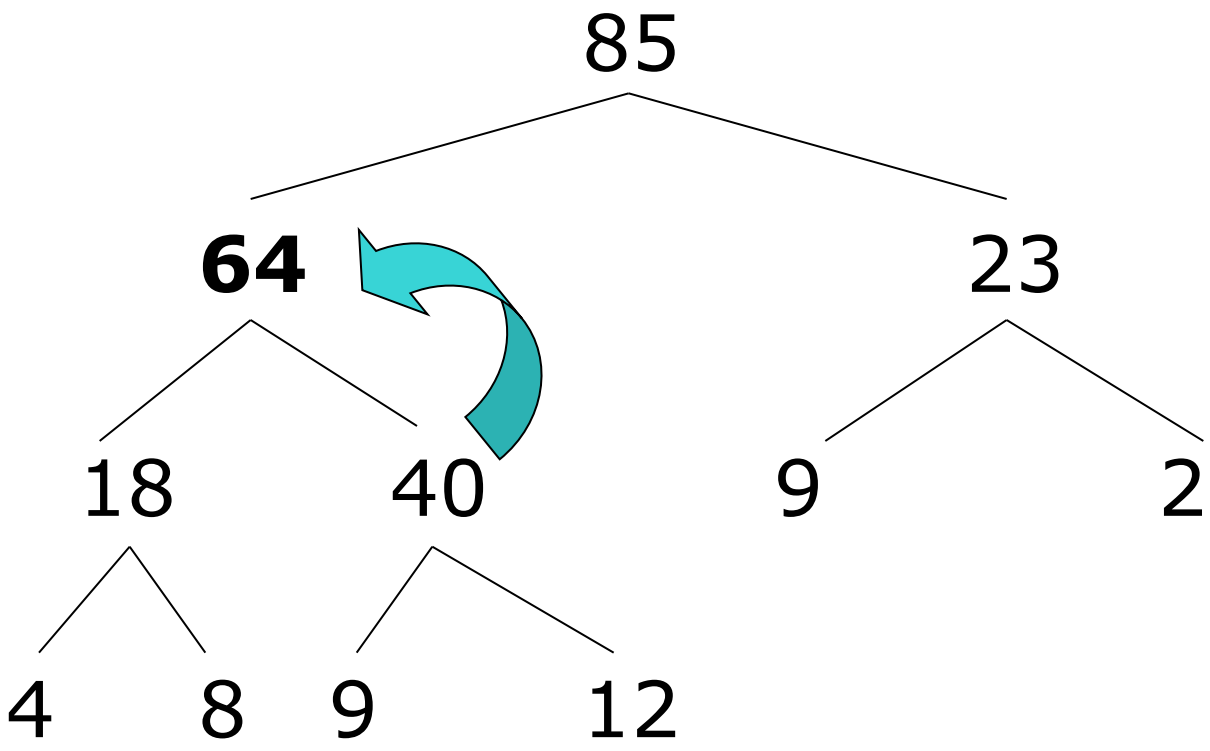
Exemple : ajout de 64



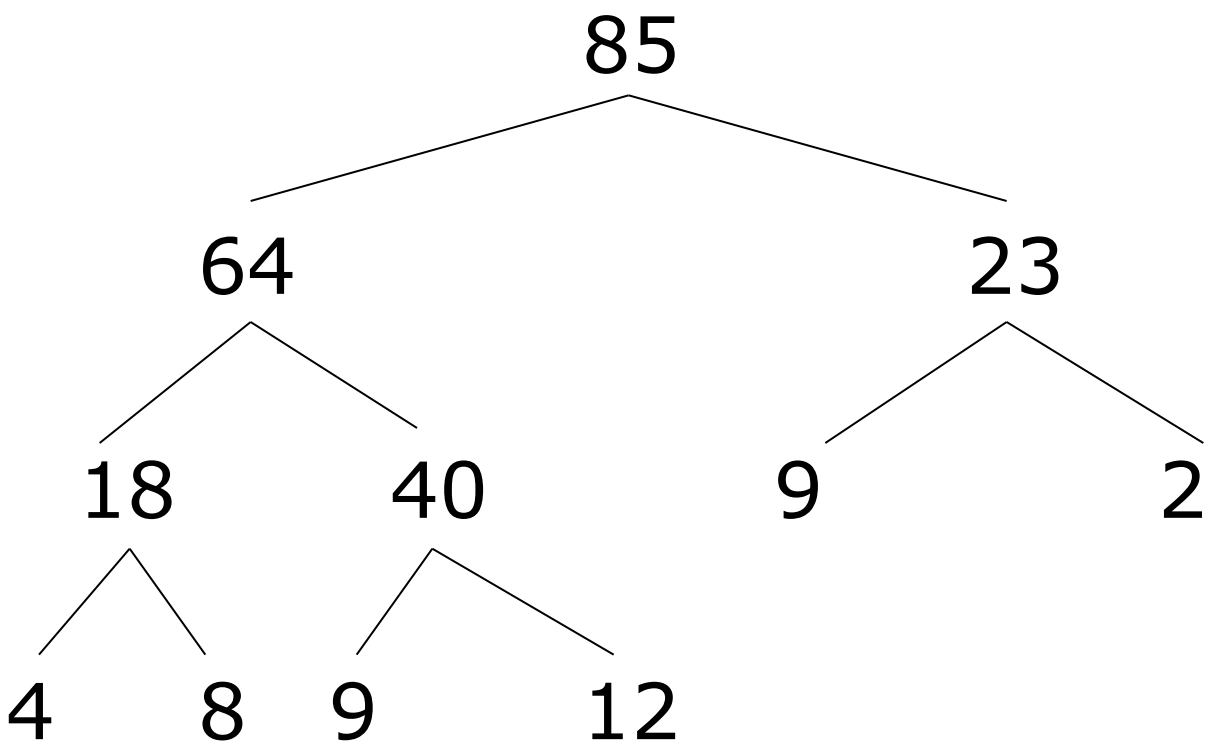
Exemple : ajout de 64



Exemple : ajout de 64



Exemple : ajout de 64

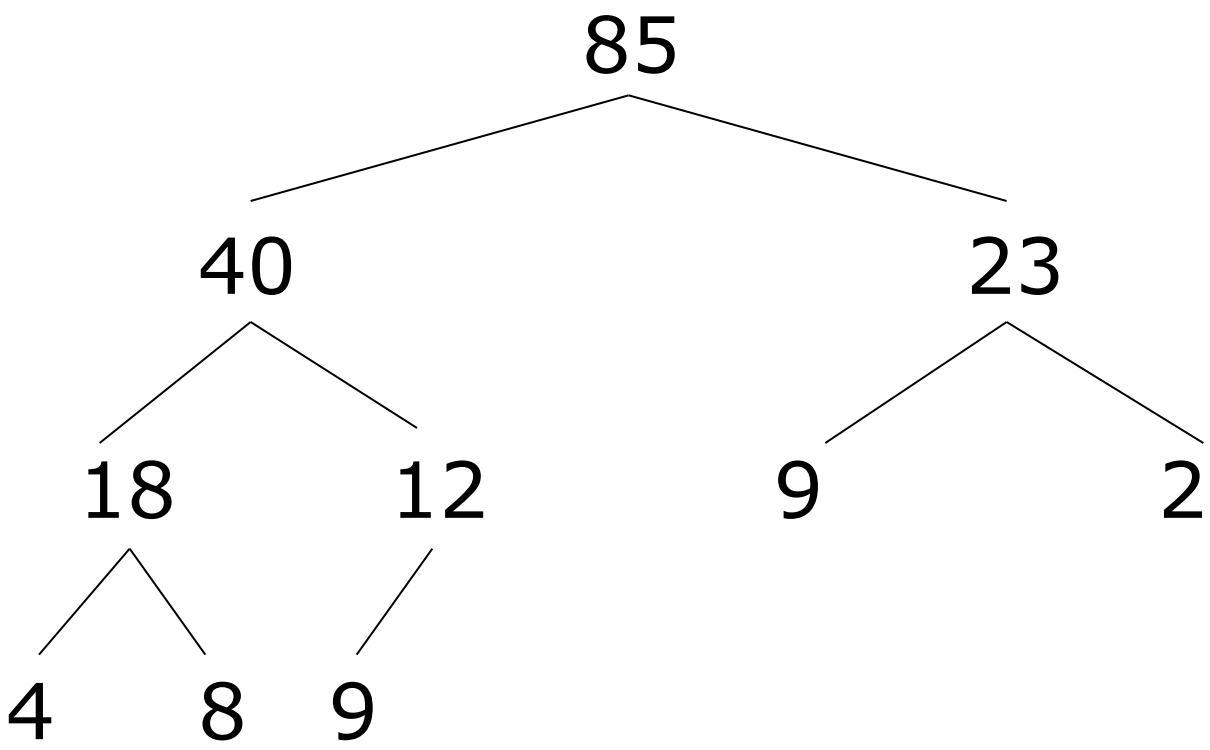


Suppression dans un tas max

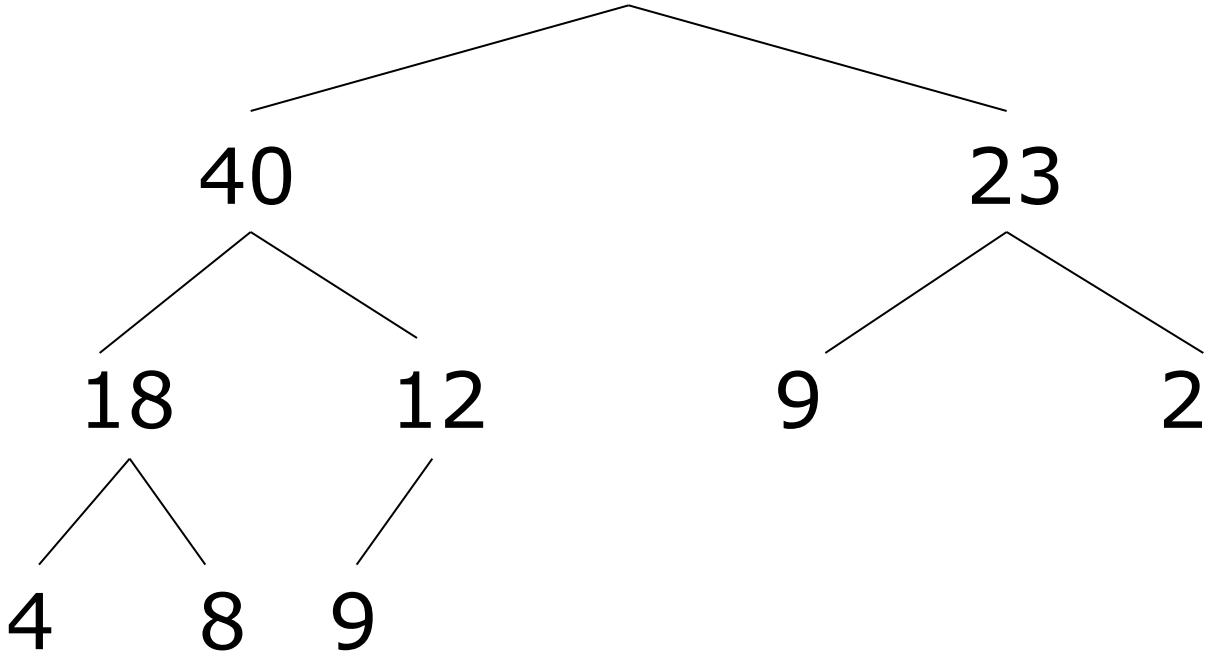
Principe

- La valeur retirée est toujours la racine
- La racine est remplacée par la feuille la plus à droite du dernier niveau (l'arbre reste complet)
- Cette valeur est permutée avec la valeur du plus grand fils tant qu'elle est inférieure à l'un de ses fils

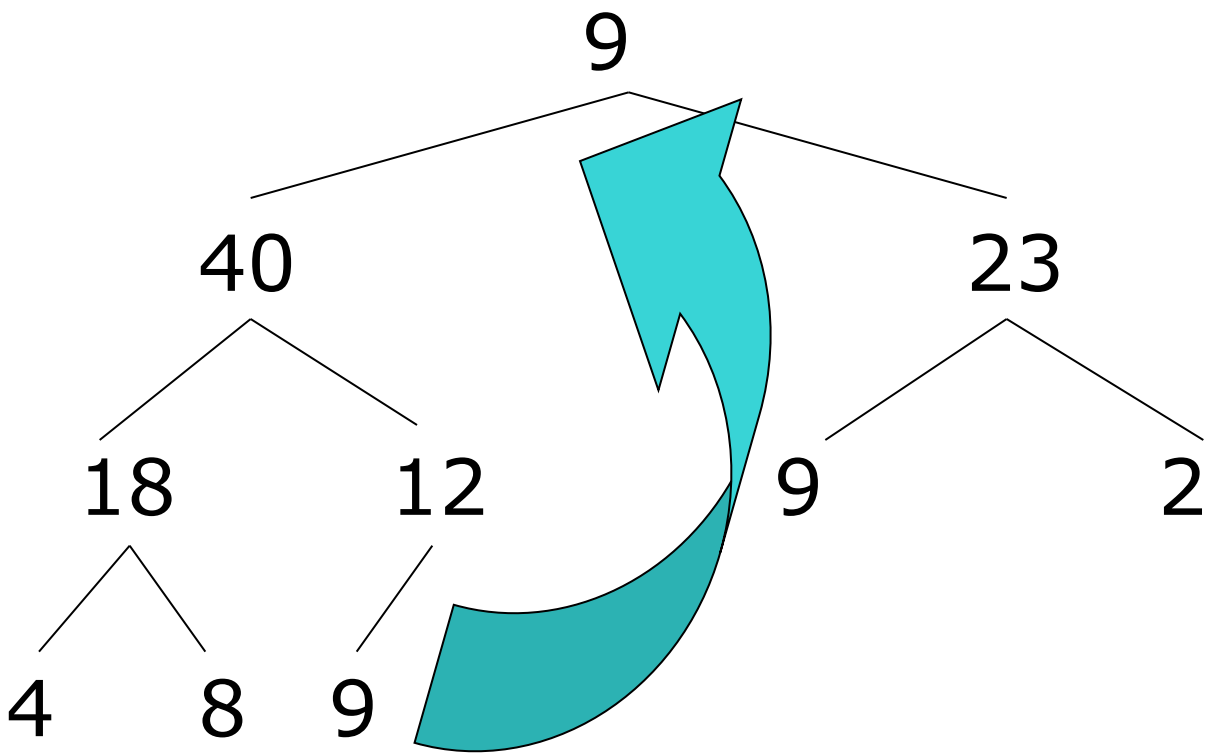
Exemple : suppression



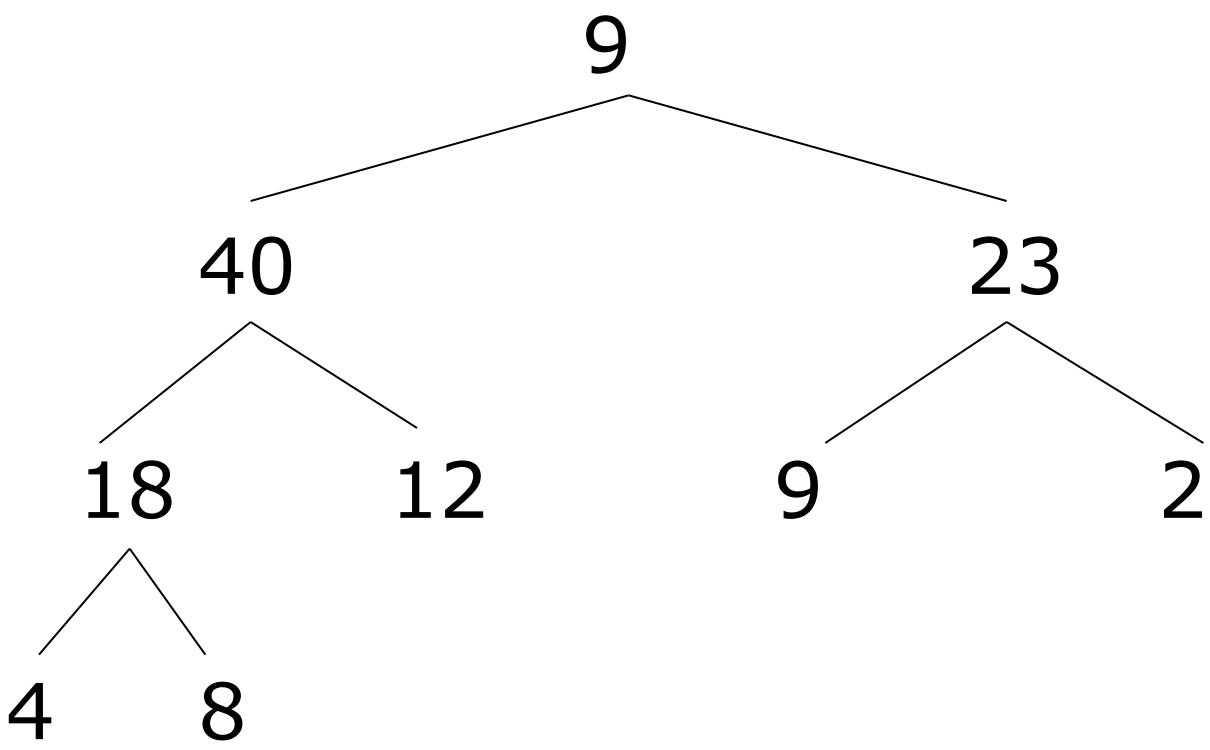
Retrait de la racine (85)



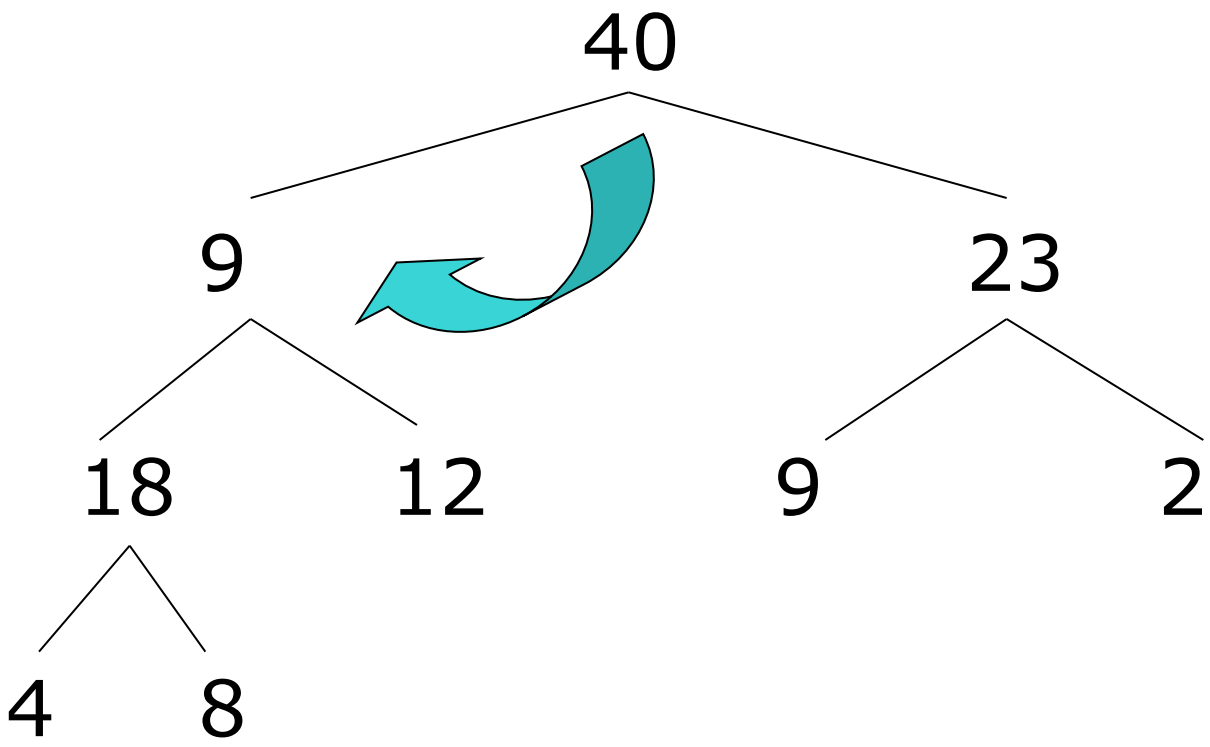
Remplacement par la dernière feuille



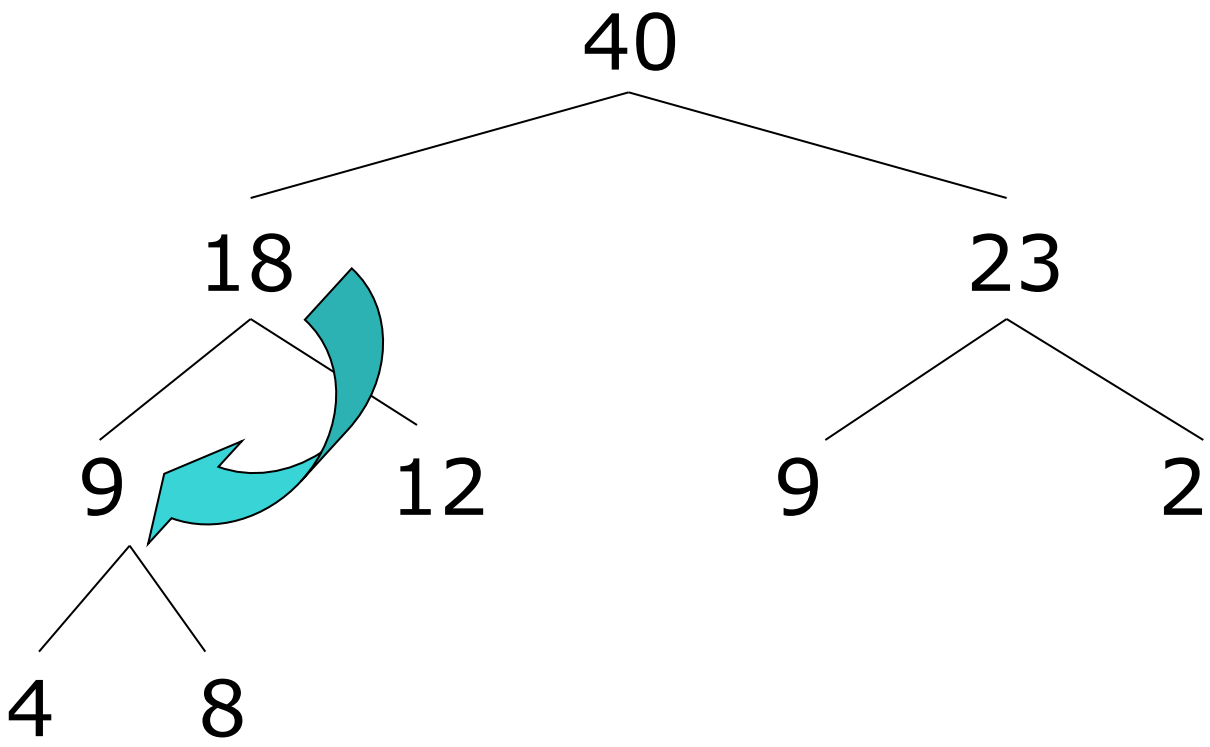
Remplacement par la dernière feuille



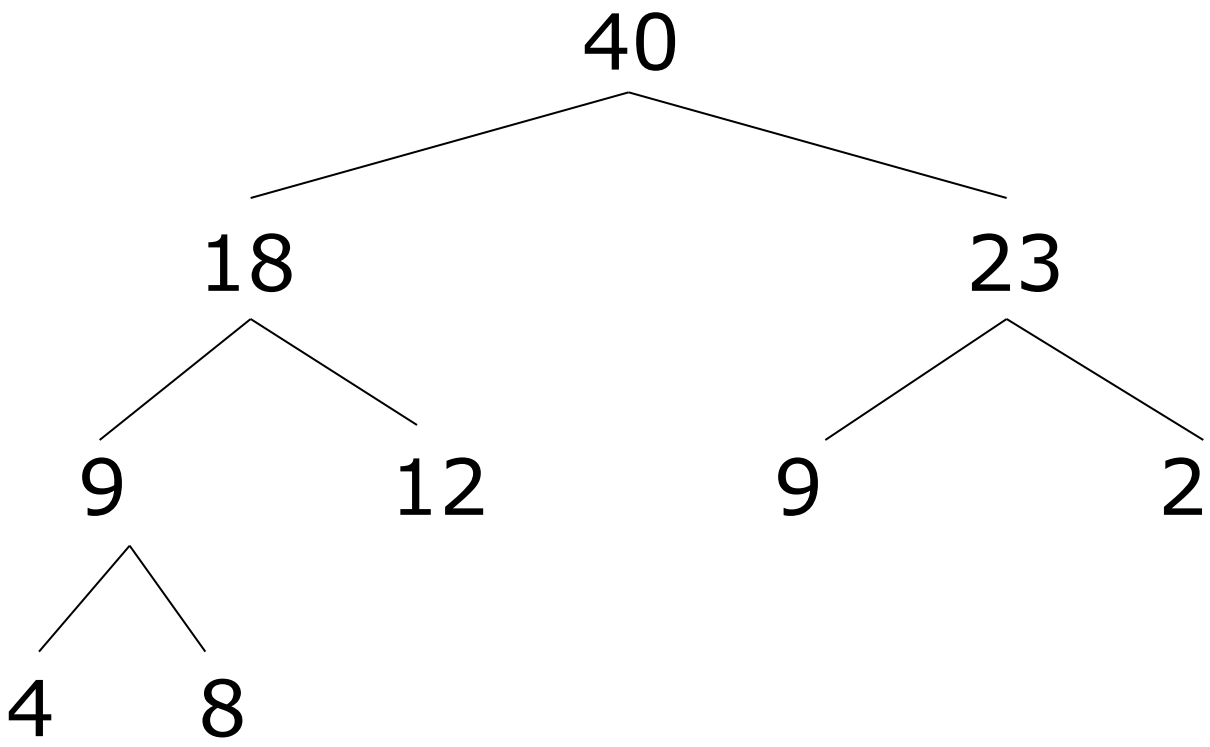
Permutations avec un fils



Permutations avec un fils



Exemple : suppression



Tri par tas

Principe

- ajouter une par une les valeurs à trier dans le tas
- retirer une par une les valeurs du tas : on obtient les valeurs dans l'ordre croissant (pour un tas min) ou décroissant (pour un tas max)

Tri par tas

Lorsque la séquence à trier est stockée dans un tableau, ce tableau représente directement le tas

☞ il n'est pas nécessaire d'introduire une seconde structure de données

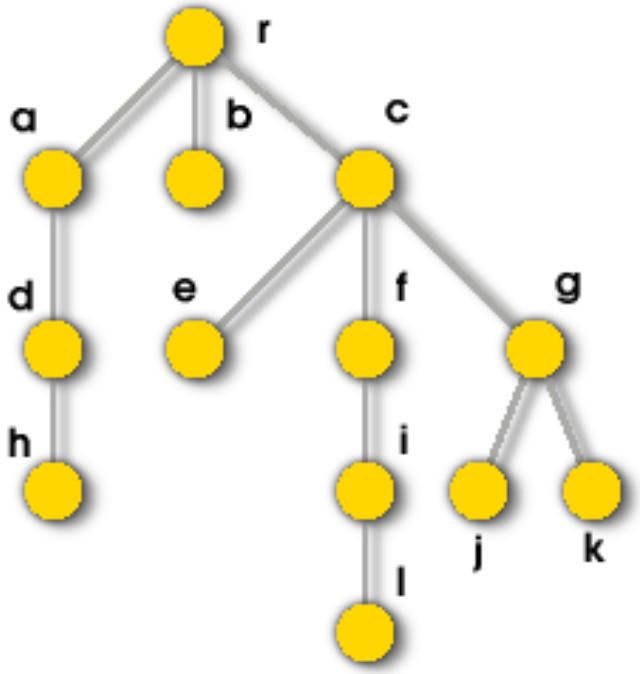
Chapitre 5

Arbres n-aires

Arbres (n-aires)

Dans un arbre quelconque, le nombre de fils d'un nœud n'est pas limité (0, 1, 2, … n)

Les fils ne sont pas ordonnés (pas de fils gauche ni droit)



Parcours standards d'arbres

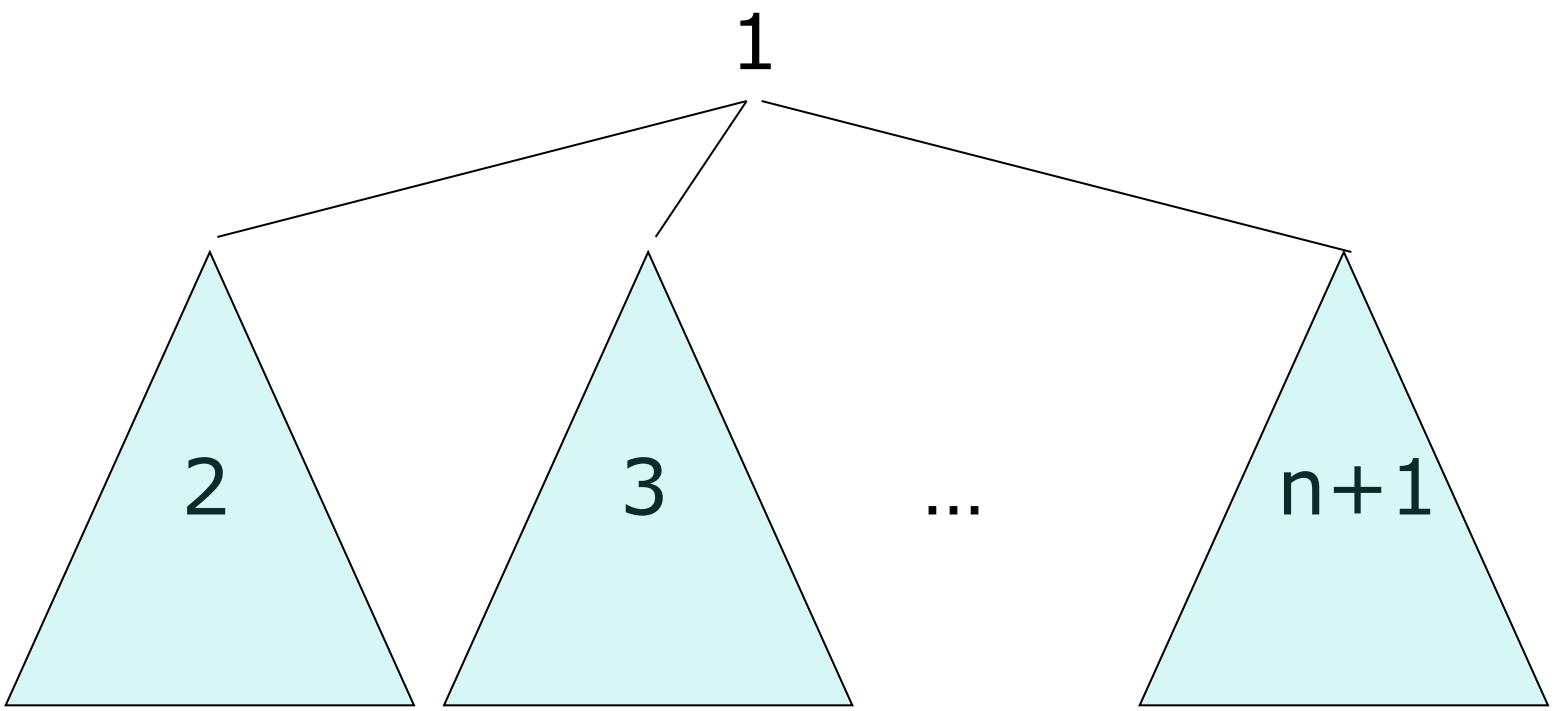
On retrouve (presque) les mêmes parcours standard :

- Parcours en profondeur
  - Préfixe ou DGD (descendant gauche droite)
  - Postfixe ou AGD (ascendant gauche droite)
- Parcours en largeur

En revanche le parcours infixe (ou symétrique) n'est possible que pour les arbres binaires

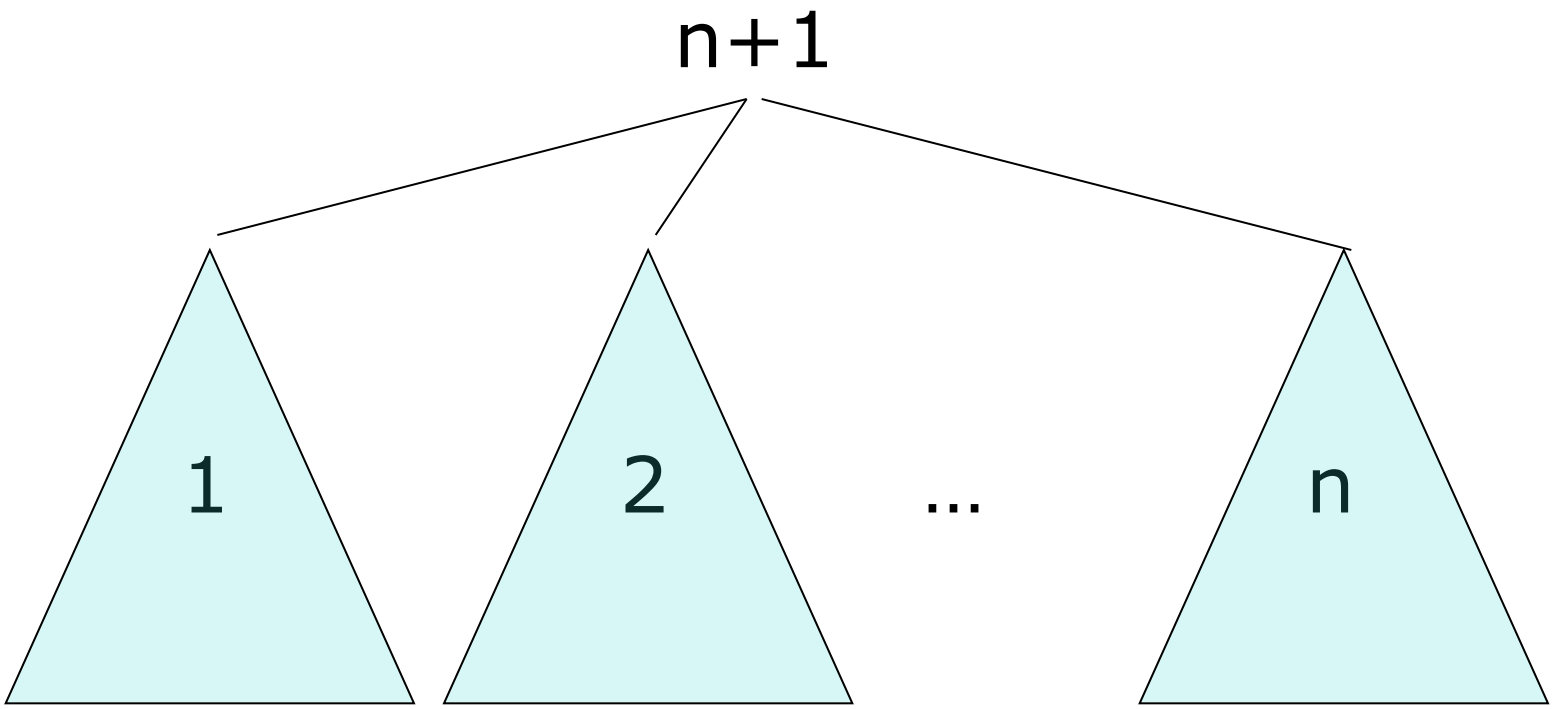
Parcours préfixe ou DGD

On étudie le nœud courant puis ses sous-arbres

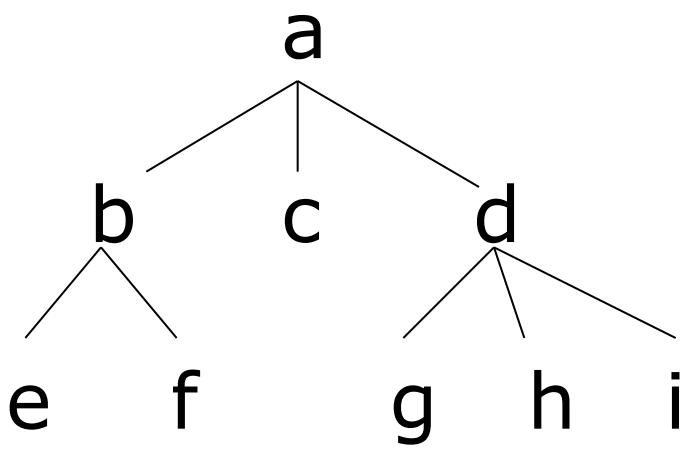


Parcours postfixe ou AGD

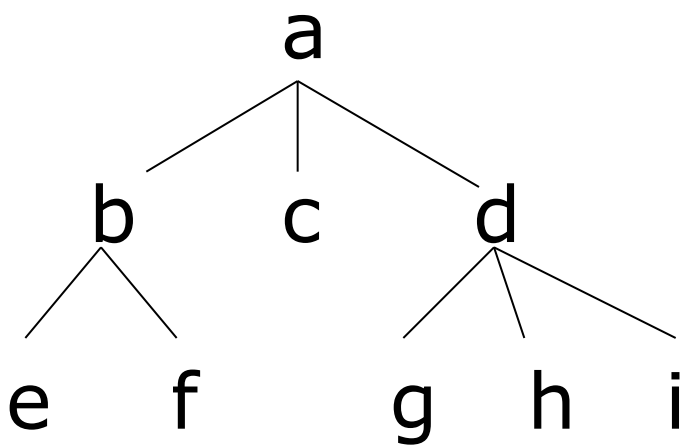
On étudie les sous-arbres puis le nœud courant



Exemple

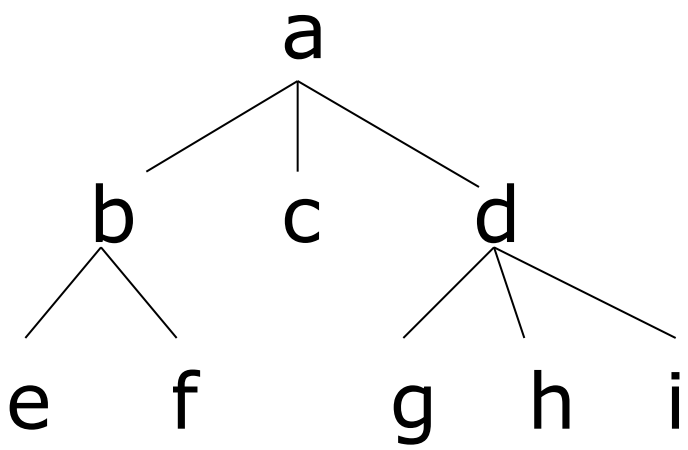


Exemple



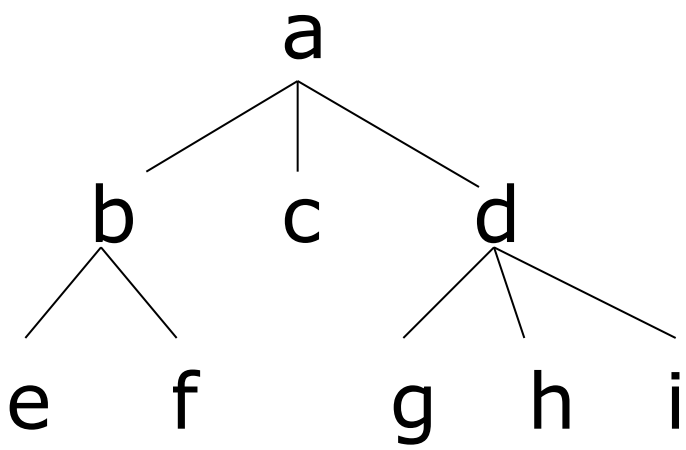
Parcours préfixe : a b e f c d g h i

Exemple



Parcours préfixe : a b e f c d g h i
Parcours postfixe : e f b c g h i d a

Exemple



Parcours préfixe : a b e f c d g h i
Parcours postfixe : e f b c g h i d a
Parcours en largeur : a b c d e f g h i

Spécifications d'un arbre n-aire

On retrouve les mêmes groupes de primitives que pour les arbres binaires :

- Primitives d'initialisation et de test
- Primitives de déplacement du nœud courant
- Primitives de consultation et de modification du nœud courant
- Primitives d'ajout et de retrait d'une feuille

Spécifications d'un arbre n-aire

Problème : comment gérer le nombre variable de fils d'un nœud ?

Plusieurs solutions :

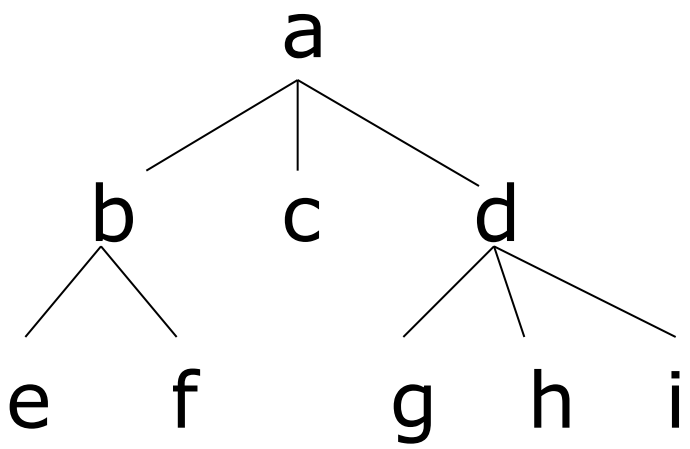
1. Soyons frères !
2. Le patriarche !

Soyons frères !

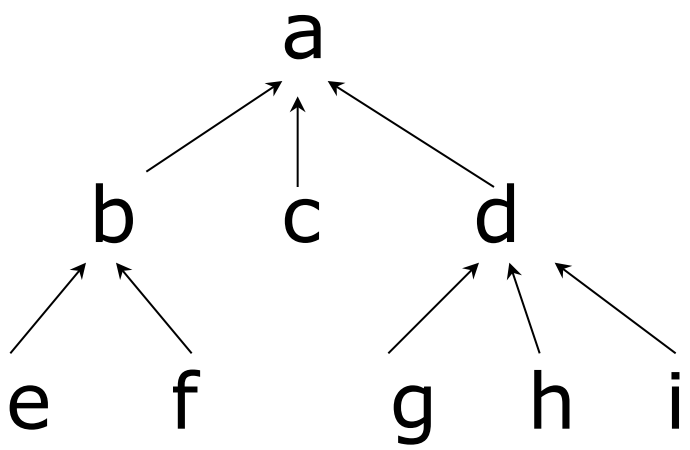
- Chaque nœud est relié à son premier fils et à son frère droit

- Nécessite
  - Modification de la structure **t_noeud**
    - Ajout de **struct noeud * frere**
  - **frere** renvoie vrai si le noeud a un frère
  - **ajout_frere** ajoute un frere

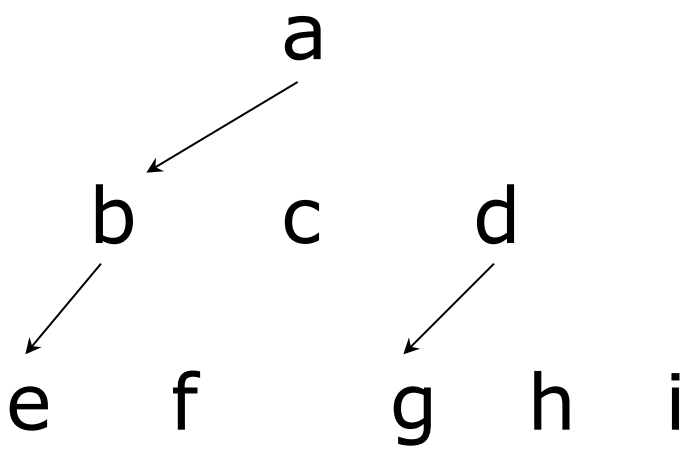
Exemple



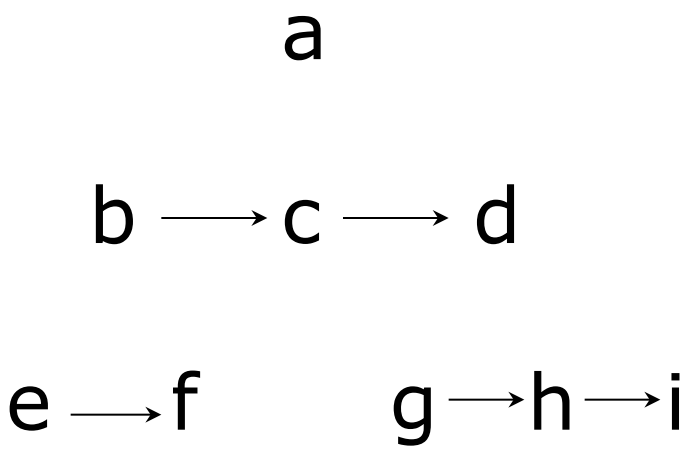
Exemple : liens vers le père



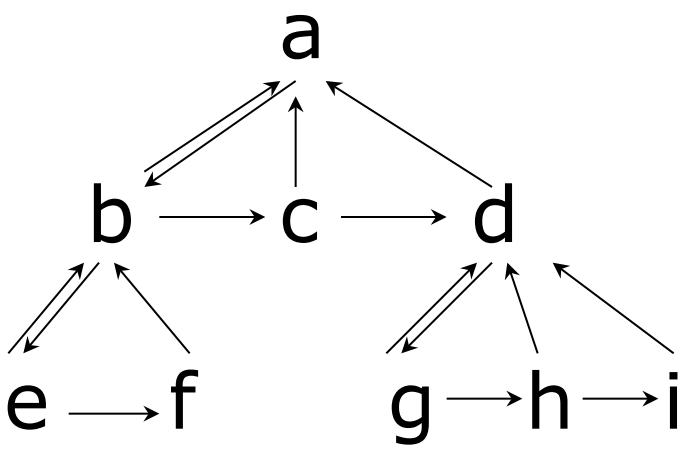
Exemple : liens vers le premier fils



Exemple : liens vers le frère droit



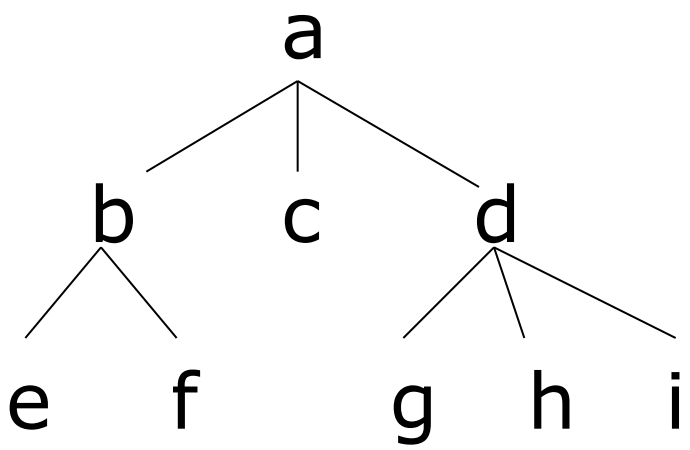
Exemple : tous les liens



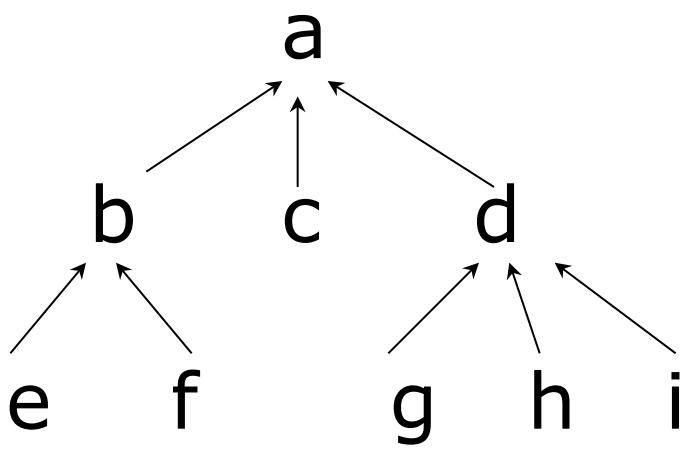
Le patriarche !

- Chaque nœud possède un lien direct vers chacun de ses fils
  - **posit_fils(n)** positionne sur le n-ième fils
- Nécessite
  - Modification de la structure **t_noeud**
    - La variables **fils** est désormais une « collection » de **struct noeud ***
    - Ajout d'une variable permettant de connaître le **nombre de fils** (taille de la collection **fils**)
    - Gestion mémoire statique ou dynamique ?

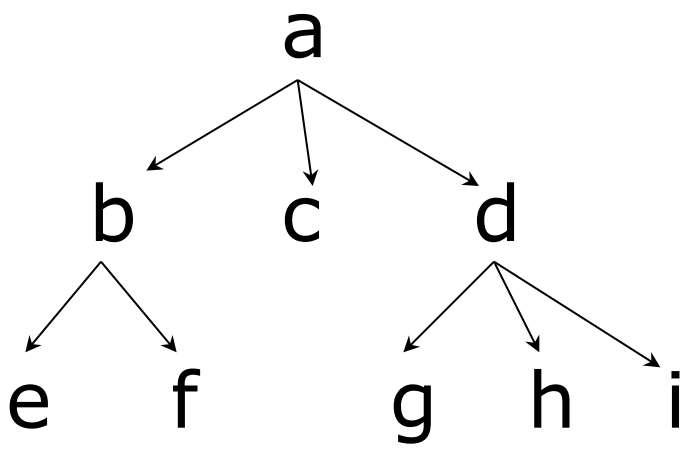
Exemple



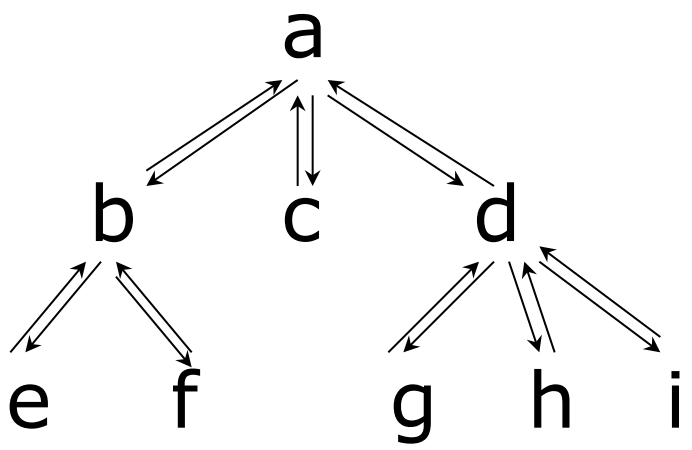
Exemple : liens vers le père



Exemple : liens vers les fils



Exemple : tous les liens



Mise en œuvre des arbres n-aires

- Mise en œuvre par pointeurs