

Structures de données et algorithmes – TP10

Iulia-Cristina Stanica

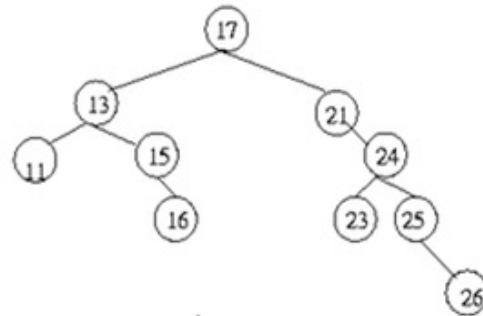
Objectifs pour aujourd'hui

- ▶ Arbres binaires de recherche –
implementation, exercices
- ▶ Rappel: méthodes de parcours

BST – implementation

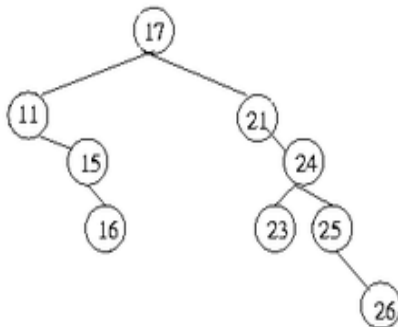
- ▶ Téléchargez l'exemple du Moodle
- ▶ Dessinez le BST de notre exemple
- ▶ Supprimez le nœud «C» et voyez ce qui se passe

Rappel



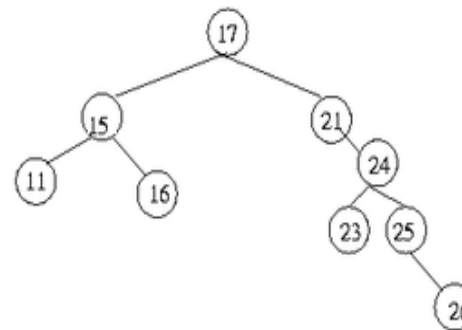
delete 13

/* delete node with both
left and right subtrees */



Method 1.

Find highest valued element
among the descendants of
left child



Method 2

Find lowest valued element
among the descendants of
right child

Exercice 1

- ▶ Écrivez une fonction qui obtient la plus grande valeur d'un BST.
- ▶ **Suggestion:**
 - Traversez l'arbre de la racine vers la droite jusqu'à ce que le pointeur droit soit NULL
 - Le nœud qui a le nœud droit NULL est le maximum
 - `BinarySearchTree <T> * p = this;` – référence à la racine
- ▶ **Que faisons-nous si nous voulons obtenir la valeur minimale?**

Exercice 2

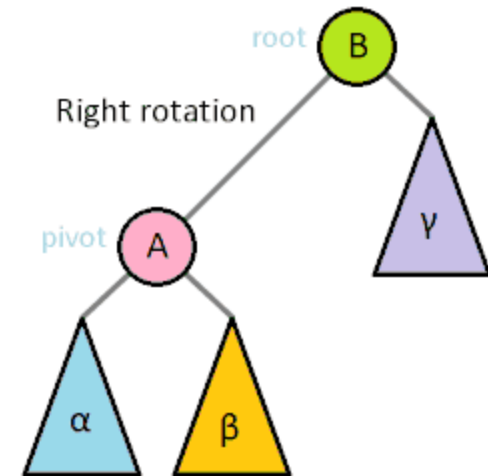
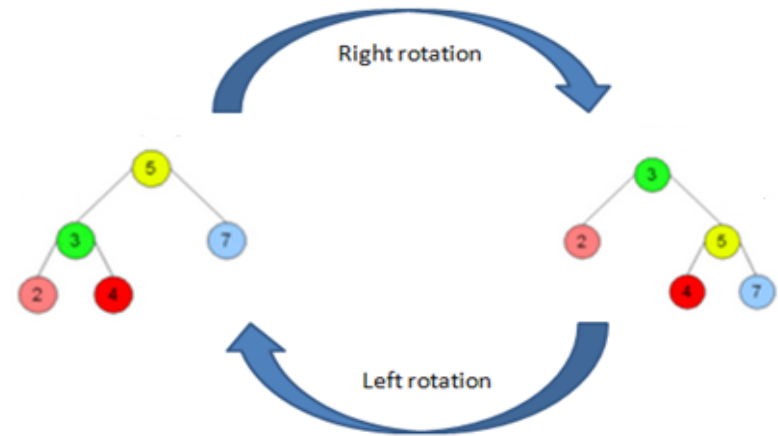
- Écrire une fonction qui affiche les valeurs des nœuds supérieures à k_1 et inférieures à k_2 , où k_1 et k_2 sont des paramètres de la fonction.

RANGEQUERYSIMPLE(T, k_1, k_2):

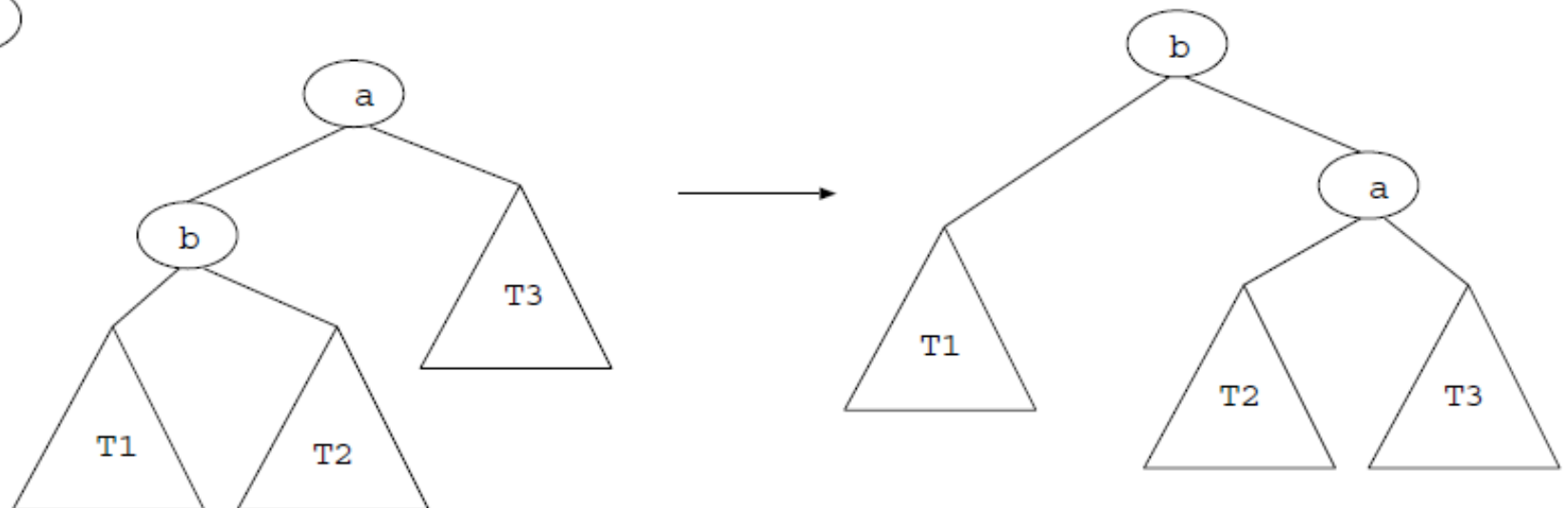
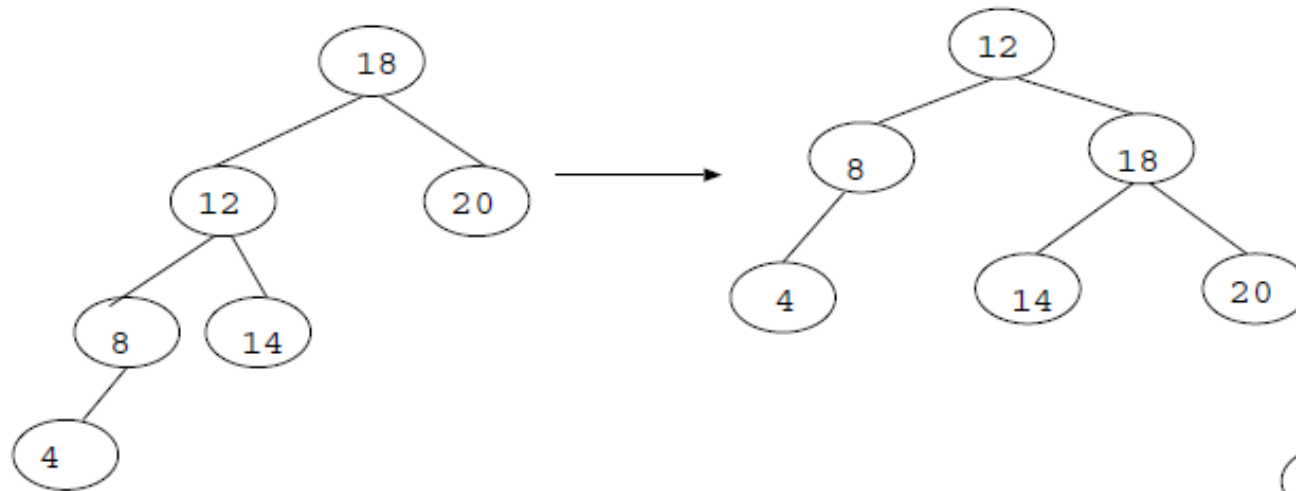
- 1: If T is empty, return
- 2: **if** $\text{key}(T.\text{root}) < k_1$ **then**
- 3: RangeQuerySimple($T.\text{right}, k_1, k_2$)
- 4: **else if** $\text{key}(T.\text{root}) > k_2$ **then**
- 5: RangeQuerySimple($T.\text{left}, k_1, k_2$)
- 6: **else**
- 7: RangeQuerySimple($T.\text{left}, k_1, k_2$)
- 8: report $T.\text{root}$
- 9: RangeQuerySimple($T.\text{right}, k_1, k_2$)

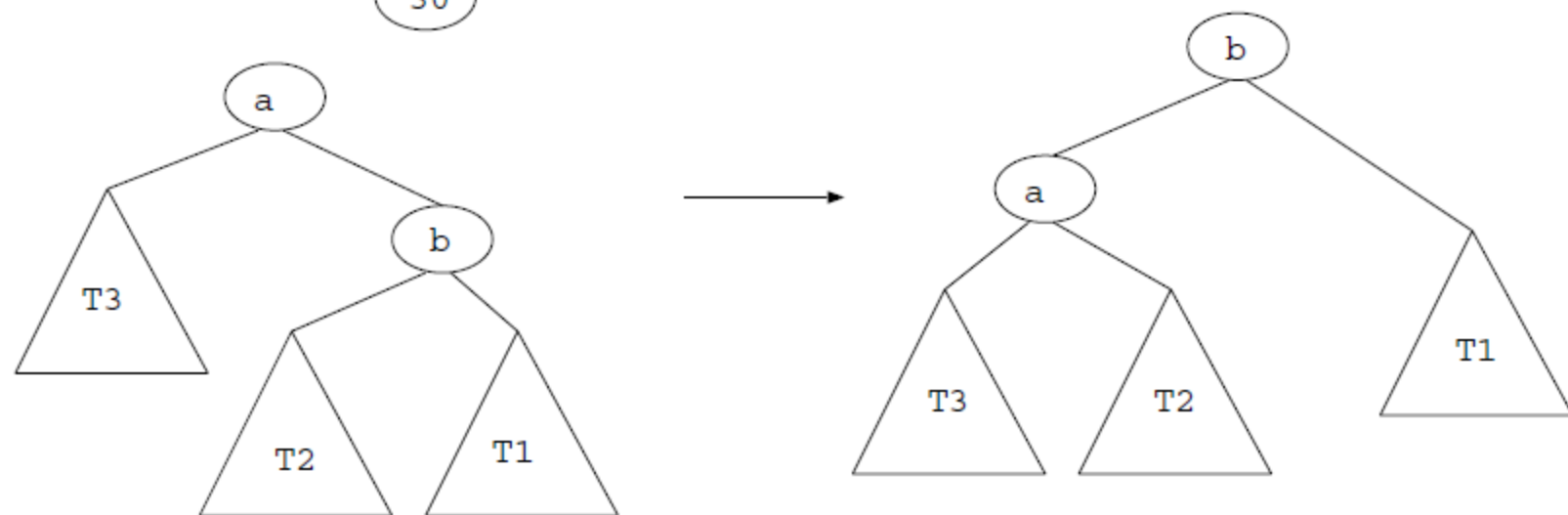
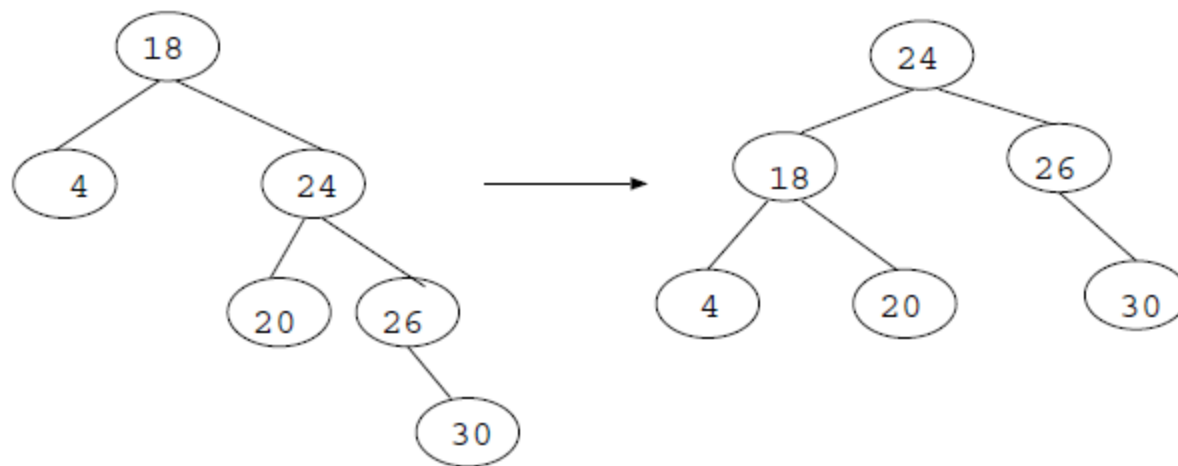
Exercice 3

- ▶ Implémentez une fonction de rotation (pas une méthode d'instance!) qui fait tourner un arbre binaire de recherche envoyé comme argument à la fonction.
- ▶ Testez sur l'arbre du diapo suivant.
- ▶ Une rotation de l'arbre est une opération sur un arbre binaire de recherche qui modifie la structure sans interférer avec le parcours en ordre des éléments.
- ▶ La rotation se déroule comme suit:
 - rotation de l'arbre vers la droite, si la hauteur de l'enfant droit est inférieure à la hauteur de l'enfant gauche;
 - rotation de l'arbre vers la gauche, si la hauteur de l'enfant gauche est inférieure à la hauteur de l'enfant droit
 - rien si l'arbre est bien équilibré.



N'oubliez pas! Il faut respecter les propriétés BST => même ordre des feuilles

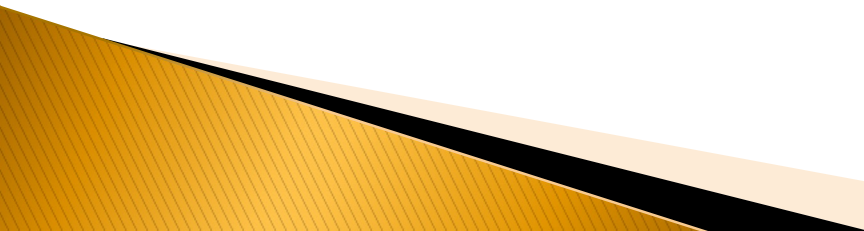




Hint – rotation a droit

- ▶ Algorithme pour faire la rotation d'un nœud vers la droite
 - La racine va être remplacée par l'enfant gauche (l'enfant gauche est maintenant « l'arbre principal »)
 - L'enfant gauche de l'ancienne racine prend la valeur de l'enfant droit de la nouvelle racine
 - L'enfant droit de la nouvelle racine prend la valeur de la racine précédente (de l'ancien arbre)

Hint: On va utiliser “ancienne racine” et “nouvelle racine” dans l'algorithme, donc c'est indiqué de faire au debut une copie de l'arbre.



Hint – rotation a gauche

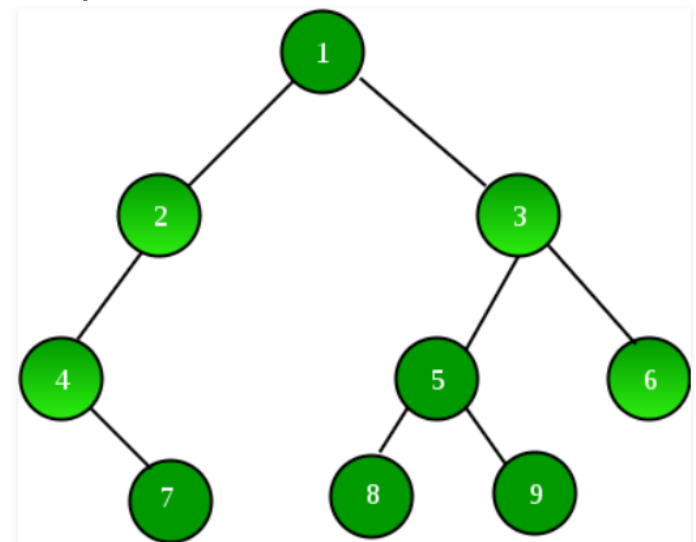
- ▶ Algorithme pour faire la rotation d'un nœud vers la gauche
 - La racine va être remplacée par l'enfant droit
 - L'enfant gauche de la nouvelle racine est affecté à l'enfant droit de l'ancienne racine
 - La racine précédente devient l'enfant gauche de la nouvelle racine

Exercice extra

- Trouvez le miroir d'un nœud donné dans un BST. Regardez les indications a ce lien:

<https://www.geeksforgeeks.org/find-mirror-given-node-binary-tree/>

Examples:

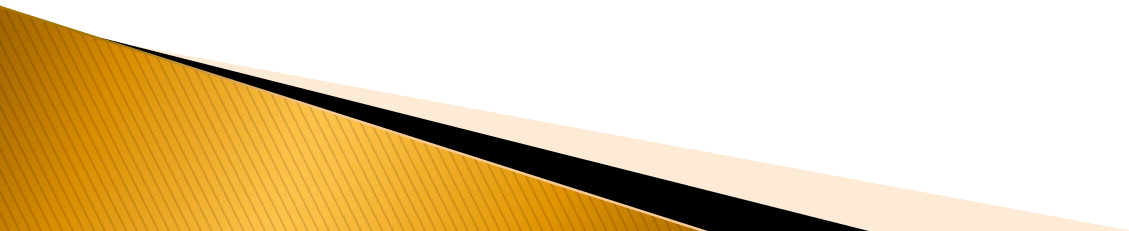


In above tree-

Node 2 and 3 are mirror nodes

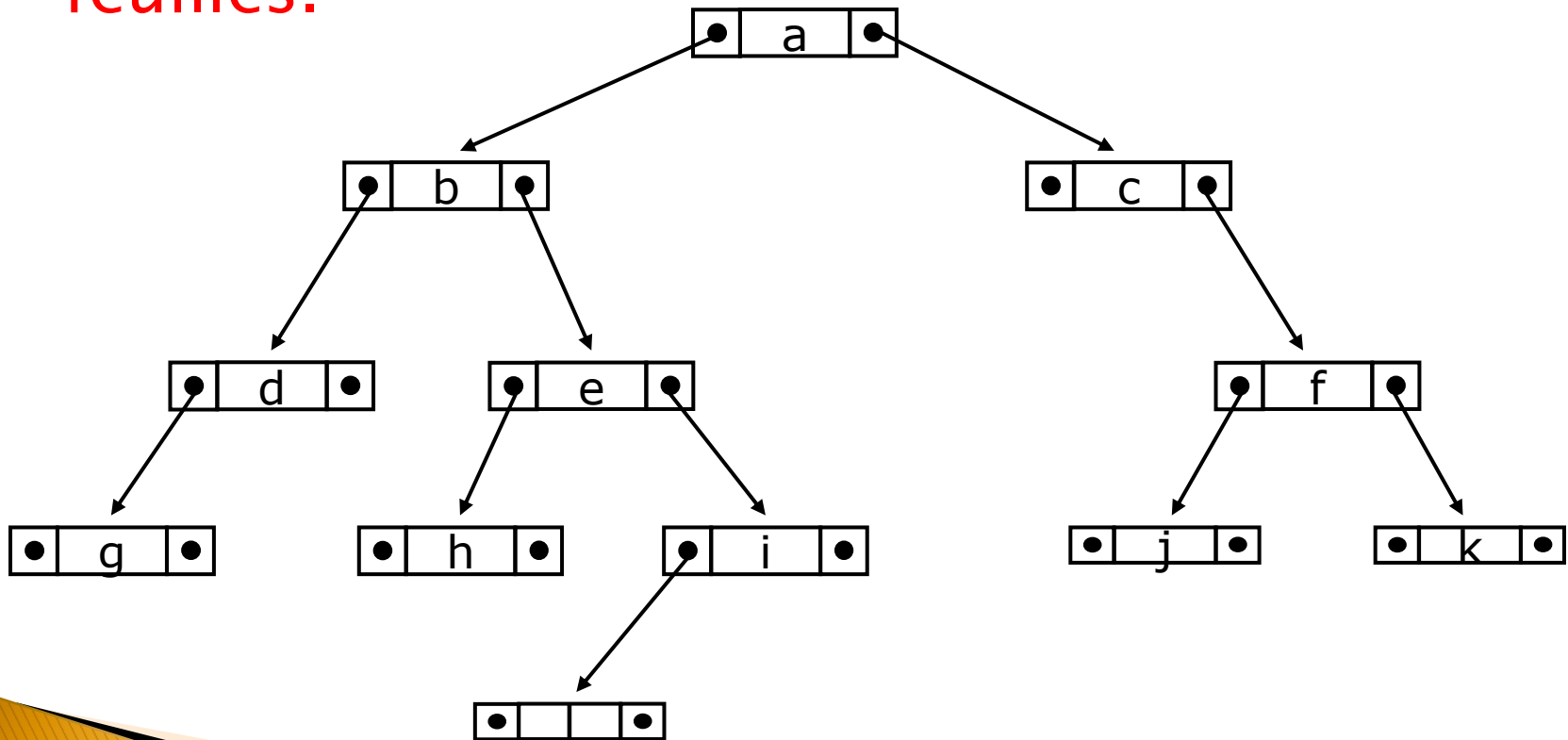
Node 4 and 6 are mirror nodes.

Support theorique



Arbres binaires (rappel)

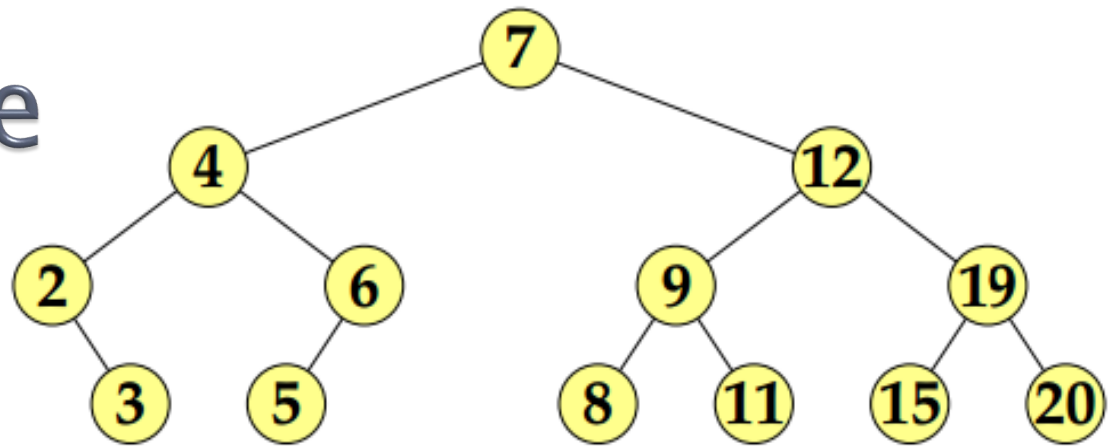
- ▶ S'il n'est pas vide, l'arbre binaire a un noeud **racine**.
- ▶ Les nœuds qui n'ont pas d'enfants s'appellent **feuilles**.



Arbres binaires de recherche (Binary search tree – BST)

- ▶ Arbres binaires aux propriétés supplémentaires:
 - Les éléments (infos) enregistrés dans les nœuds sont comparables
 - Les éléments du sous-arbre gauche du nœud p sont \leq l'élément enregistré p
 - Les éléments du sous-arbre droit du nœud p sont $>$ l'élément enregistré dans p
 - Recherche rapide
 - Suppression compliquée (nous devons conserver les propriétés du BST)

Exemple



Pre-order (Racine Gauche Droit) 7, 4, 2, 3, 6, 5, 12, 9, 8, 11, 19, 15, 20.

In order (Gauche Racine Droit) 2, 3, 4, 5, 6, 7, 8, 9, 11, 12, 15, 19, 20.

Post-order (Gauche Droit Racine) 3, 2, 5, 6, 4, 8, 11, 9, 15, 20, 19, 12, 7

Dans le parcours en ordre, la traversée d'un BST donne une séquence triée de valeurs

Complexité

- ▶ **La recherche** a la pire complexité de $O(n)$. En général, la complexité temporelle est $O(h)$ où h est la hauteur de BST.
- ▶ **L'insertion** présente la pire complexité de $O(n)$. En général, la complexité temporelle est $O(h)$.
- ▶ **La suppression** présente la complexité la plus défavorable de $O(n)$. En général, la complexité temporelle est $O(h)$.

