

Introduction: Arbres de recherche + Rappel: Arbres binaires de recherche

Rappel : Dictionnaires ordonnés:

- Opérations principales:
 - trouver(k)**: Si le dictionnaire a une entrée de clé k , retourne la valeur associée à cette entrée, sinon retourne NULL.
find(k):
 - findAll(k)**: Retourne un itérateur de toutes les entrées de clé k , ou NULL si aucune entrée de clé k
 - insérer(k,v)**: insérer l'entrée (k,v) dans le dictionnaire
put(k,v):
 - enlever(k)**: Si le dictionnaire a une entrée de clé k , l'enlever et retourner sa valeur associée, sinon retourner NULL
remove(k):
 - removeAll(k)**: Si le dictionnaire a une ou plusieurs entrées de clé k , les enlever et retourner un itérateur des valeurs associées à chacune de ces entrées, sinon retourner NULL

Dictionnaires ordonnés (suite)

- Opérations principales (suite):
 - successesurs(k)**: Retourne un itérateur des entrées dont la clé est plus grande ou égale à k ; en ordre croissant
successors(k):
 - prédécesseurs(k)**: Retourne un itérateur des entrées dont la clé est plus petite ou égale à k ; en ordre décroissant
predecessors(k):
 - closestKeyBefore(k)**: Retourne la clé (ou la valeur) de l'entrée ayant la plus grande clé plus petite ou égale à k
closestValBefore(k):
 - closestKeyAfter(k)**: Retourne la clé (ou la valeur) de l'entrée ayant la plus petite clé plus grande ou égale à k
closestValAfter(k):

Implémentations Dictionnaires ordonnés:

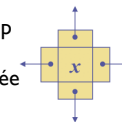
- "Look-up table"
 - Complexité en temps:
 - insérer $O(n)$ enlever $O(n)$ trouver $O(\log n)$: recherche binaire
 - Complexité en espace: $O(n)$
- "Skip List"
 - Une "skip list" est une structure de données pour les dictionnaires qui utilise un algorithme randomisé pour l'insertion d'éléments
 - L'implémentation d'une skip list se fait à l'aide d'une structure chaînée, composée de noeuds quadruples:
 - On a avec une très haute probabilité les complexités suivantes:
 - insérer $O(\log n)$
 - enlever $O(\log n)$
 - trouver $O(\log n)$
 - Complexité en espace: $O(n)$

4 4 9 18 28 28 32 59 59 59 75 78 82 88

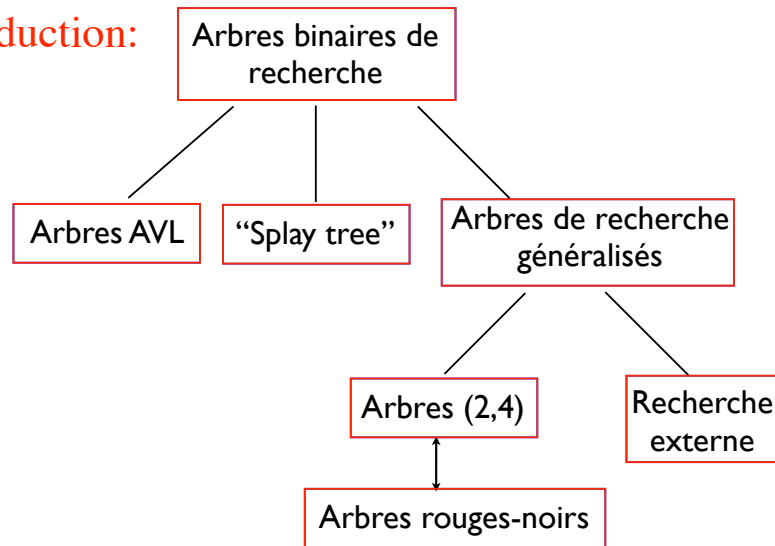
S_2 [-∞] [4] [+∞]

S_1 [-∞] [23] [+∞]

S_0 [-∞] [12] [23] [45] [+∞]



Introduction:



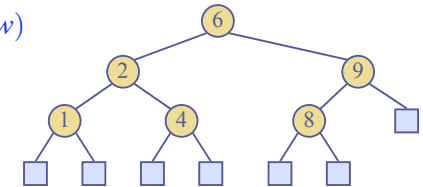
Arbres binaires de recherche

- Un arbre binaire de recherche est un arbre binaire qui garde en mémoire des entrées (clé-valeur) dans ses noeuds internes et qui satisfait la propriété suivante:

- Soient u, v et w trois noeuds tels que u est dans le sous-arbre gauche de v et w dans son sous-arbre droit. Alors, on a

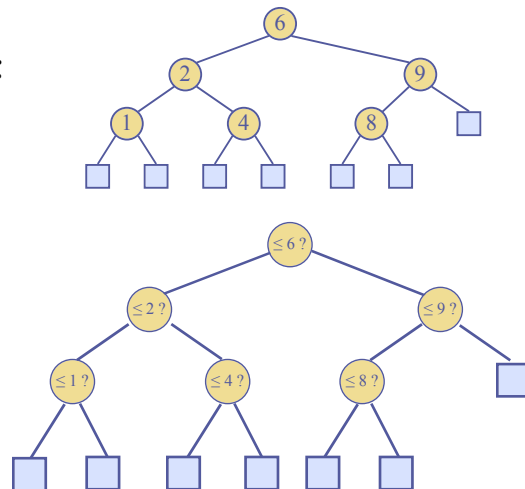
$$clé(u) < clé(v) \leq clé(w)$$

- Les noeuds externes ne gardent en mémoire aucune entrée
- Un parcours symétrique de l'arbre visite les clés en ordre croissant



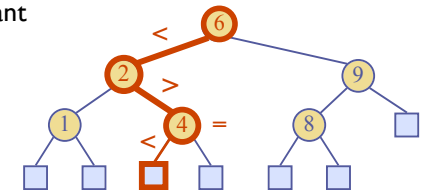
Chercher dans un arbre binaire de recherche

L'idée:



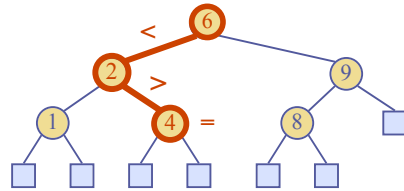
Chercher dans un arbre binaire de recherche

- Pour chercher un élément de clé k dans un arbre binaire de recherche, on va suivre un chemin descendant en commençant la recherche à la racine.
- Exemple 1:** Chercher(4)
- Le prochain noeud visité dépend du résultat de la comparaison de k avec la clé du noeud dans lequel on se trouve.
- Si on trouve un noeud interne de clé k , on retourne la valeur correspondant à cette entrée de clé k
- Sinon, on retourne NULL
 - Exemple 1:** Chercher(3)



Analyse de la recherche dans le pire des cas

- L'algorithme de recherche dans un arbre binaire est récursif et exécute un nombre constant d'opérations élémentaires à chaque appel
- Chaque appel récursif est exécuté sur un enfant du noeud courant et donc, à chaque appel récursif, on descend d'un niveau dans l'arbre
- Dans le pire des cas, on fera $h+1$ appels de la fonction.
- La complexité dans le pire des cas est $O(h)$

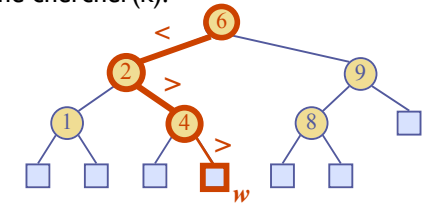


Insérer dans un arbre binaire de recherche

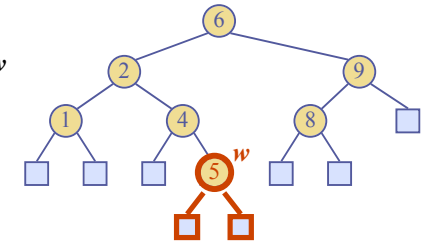
- Pour insérer un élément (k,v) dans un arbre binaire de recherche, on commence par exécuter l'algorithme $\text{chercher}(k)$.

- Exemple 1: Insérer $(5,v)$

- Si k n'est pas dans l'arbre l'algorithme $\text{chercher}(k)$ se terminera dans une feuille w



- On insère k dans w et on change w en un noeud interne

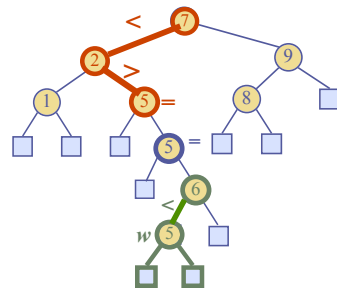


Insérer dans un arbre binaire de recherche

- Si k est dans l'arbre l'algorithme $\text{chercher}(k)$ se terminera sur un noeud interne v . On appelle récursivement l'algorithme sur le filsDroit(v), jusqu'à ce qu'on arrive à un noeud externe w

- Exemple 2: Insérer $(5,v)$

- On insère k dans w et on change w en un noeud interne

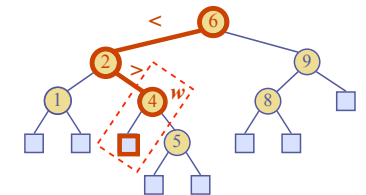


Supprimer dans un arbre binaire de recherche

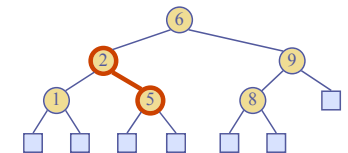
- Pour enlever un élément de clé k dans un arbre binaire de recherche, on commence par exécuter l'algorithme $\text{chercher}(k)$.

- Exemple 1: Enlever (4)

- Si k est dans l'arbre l'algorithme $\text{chercher}(k)$ se terminera dans un noeud interne w



- Si l'un des enfant de w est une feuille, on enlève cette feuille et w



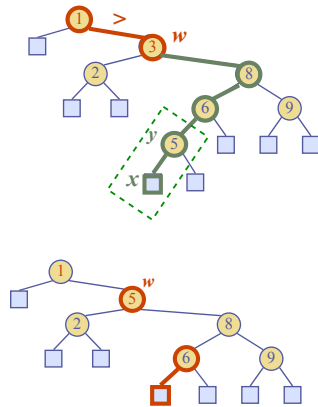
- Sinon...

Supprimer dans un arbre binaire de recherche (suite)

- Si k est dans l'arbre, l'algorithme $\text{chercher}(k)$ se terminera dans un noeud interne w . Si les fils de w sont tous les deux des noeuds internes alors

● Exemple 2: Enlever(3)

- On trouve le noeud interne y qui suit w lors d'un parcours symétrique de l'arbre et son fils gauche x
- On enlève l'entrée dans w et on la remplace par l'entrée dans y
- On enlève les noeuds y et x



Performance:

- Considérons un dictionnaire ordonné avec n entrées implémenté à l'aide d'un arbre ordonné de hauteur h
 - L'espace utilisé est en $O(n)$
 - Les opérations d'insertion, de recherche et de suppressions se font en $O(h)$
- La hauteur h est $O(n)$ dans le pire des cas et $O(\log n)$ dans le meilleur cas

