Programmation fonctionnelle

ensIIE

Semestre 3 – 2026/27

Institut Mines-Télécom
ensIIE école associée
université PARIS-SACLAY

ensIIE

www.ensiie.fr

Recherche par dichotomie

on aimerait avoir des opérations de test d'appartenance, d'insertion et de suppression efficaces en moyenne et dans le pire des cas

tableau *trié*

- recherche en $O(\log n)$ par dichotomie

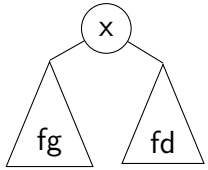
mais

- tri du tableau en $O(n \log n)$
- insertion et suppression coûteuses ($O(n)$, décalages)
- structure non persistante

Arbres binaires de recherche (ABR)

Structure de données pour exploiter la recherche par dichotomie

Arbre dont les nœuds contiennent des éléments, possédant au plus deux fils.



Invariant : pour un nœud

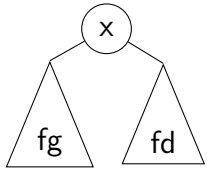
- les valeurs dans le sous-arbre gauche fg sont toutes plus petites que x
- les valeurs dans le sous-arbre droit fd sont toutes plus grandes que x
- fg et fd vérifie l'invariant

Preuve par induction sur les ABR

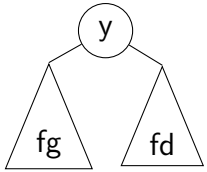
Pour montrer que P est vrai pour tout arbre binaire, il suffit de montrer que

- P est vrai pour l'arbre vide
- pour tous arbres fg et fd, pour toute valeur x,
  si on suppose que P est vraie pour fg et fd (hypothèses d'induction)

alors P est vrai pour



Appartenance

Appartenance de x dans 

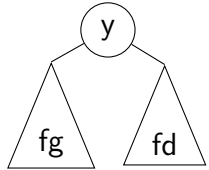
- si x $=$ y, vrai
- si x $<$ y et fg est non-vide, tester appartenance de x dans fg
- si x $>$ y et fd est non-vide, tester appartenance de x dans fd

Complexité : $O(h)$

- à chaque appel récursif, on diminue la hauteur de l'arbre d'au moins 1
- si la bonne clef se trouve au plus bas de l'arbre, il faudra h étapes, où h est la hauteur de l'arbre

Ajout

add x dans l'arbre vide : retourner un nœud singleton x

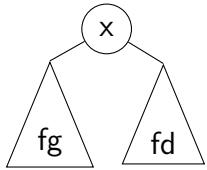
add x dans 

- x $=$ y, retourner l'arbre inchangé
- si x $<$ y, ajouter x dans fg
- si x $>$ y, ajouter x dans fd

Complexité : $O(h)$

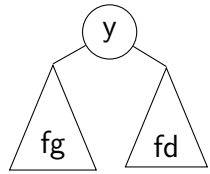
- à chaque appel récursif, on diminue la hauteur de l'arbre d'au moins 1
- on peut avoir à descendre jusqu'au nœud le plus bas

Suppression

Supprimer x dans 

- si fg vide, on retourne fd
- si fd vide, on retourne fg
- sinon,
  - on recherche le nœud avec la plus petite valeur y dans fd (celle la plus à gauche)
  - on la met à la place de la racine
  - on supprime y dans fd

Suppression (suite)

Supprimer x dans 

avec x $\neq$ y :

- si x $<$ y, supprimer x dans fg
- si x $>$ y, supprimer y dans fd

Complexité : $O(h)$

- si la clef à supprimer est dans le nœud le plus à droite, $O(h)$
- donc si la clef à supprimer est à la racine, recherche du plus à droite en $O(h)$ et suppression en $O(h) \Rightarrow O(h)$
- si la clef n'est pas à la racine, on fait un appel récursif sur un arbre de hauteur $h-1$ au maximum $\Rightarrow O(h)$

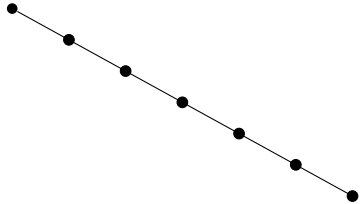
Complexité

h hauteur de l'ABR, n nombre de nœuds

- vide : $\mathcal{O}(1)$
- appartenance : $\mathcal{O}(h)$
- ajout : $\mathcal{O}(h)$
- cardinal : $\mathcal{O}(n)$
- union : $\mathcal{O}(n_1 \times h_2)$
- intersection : $\mathcal{O}(n_1 \times h_2)$
- retrait : parcours $\mathcal{O}(h)$
- agrégation $\mathcal{O}(n)$, map $\mathcal{O}(n^2)$
- égalité $\mathcal{O}(n \times h)$

Complexité dans le pire des cas

La hauteur d'un arbre à n nœuds est n dans le pire des cas :



En particulier, on obtient un arbre de ce type si on insère les éléments par ordre croissant

La complexité dans le pire des cas est $O(n)$ pour l'appartenance, l'ajout et la suppression

Complexité en moyenne

La hauteur moyenne d'un arbre à n nœuds est $\sqrt{n}$

Toutefois, si on considère les arbres obtenus en insérant les éléments de $[0 \cdots n-1]$ suivant toutes les permutations possibles, la hauteur moyenne d'un arbre est $\log n$

La complexité en moyenne est donc $O(\log n)$ pour l'appartenance, l'ajout et la suppression