INSERTION DANS UN B-ARBRE.

Leçons 901, 921, 932

Référence Cormen chap. 18

Théorème

L'insertion dans un B-arbre de hauteur h se fait en $O(h)$ accès disque.
(La suppression aussi)
+ la hauteur h d'un B-arbre avec $t-1$ à $2t-1$ clés par noeud est $h = O(\log_t(n))$ où n est le nombre total de clés.

Résumé

0 - Petit mot sur B-arbre vs. arbre binaire de recherche et sur le coût des accès disques et de la recherche dans un B-arbre.
I - Etude de l'insertion dans un B-arbre.
II - Borne sur la hauteur h.
Bonus - Un mot sur la suppression.

0 · Un B-arbre c'est une arborescence T telle que :

1) Chaque noeud x contient les champs
   - $n(x)$: le nombre de clés
   - $clé_1(x) \leq \dots \leq clé_{n(x)}(x)$: les valeurs stockées dans le noeud.
   - $feuille(x)$: booléen qui dit si l'arbre est une feuille.
2) Chaque noeud x contient $c_1(x), \dots, c_{n+1}(x)$ pointeurs vers ses enfants
3) Les clés déterminent les valeurs stockées dans les noeuds enfants.
4) Toutes les feuilles ont pour profondeur h.
5) Chaque noeud a minimum $t-1$ clés (sauf la racine) et maximum $2t-1$.

- L'idée d'un B-arbre est de généraliser les ABR de façon à minimiser le nombre d'accès disque, qui sont de loin les opérations les plus coûteuses en temps, mais qui sont nécessaires pour stocker de gros volumes de données.

- Pour un B-arbre, la recherche se fait en $O(t \log_t(n))$ contre $O(\log_2(n))$ dans un ABR.
  t est effectivement une constante mais en pratique on prend $t \simeq 1000$ ce qui permet de stocker beaucoup de clés dans une petite hauteur.

Remarque : accéder à 1000 clés stockées consécutivement sur un disque dur n'est pas si long car le bras du disque bouge peu.

I] • On va avoir besoin d'une procédure qui permet de découper un noeud plein en deux noeuds.

Définissons ainsi

PARTAGER

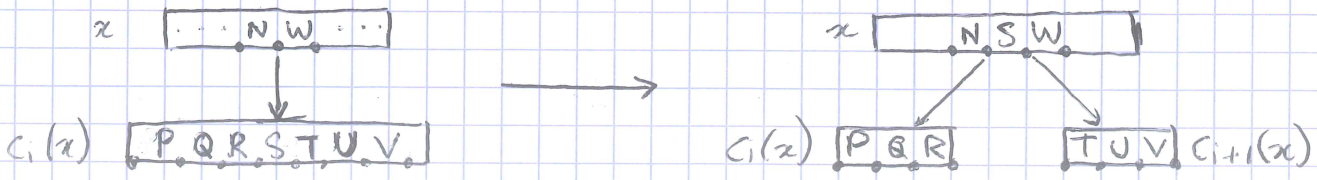
Entrées : x un noeud non plein
i un indice tel que $c_i(x)$ est plein.

1) Découper $c_i(x)$ autour de la médiane et rajouter la médiane au noeud x.

2) Ecrire sur le disque dur les mises à jour du noeud parent et des deux noeuds enfants

3 ECRITURES

Exemple avec $t = 4$.



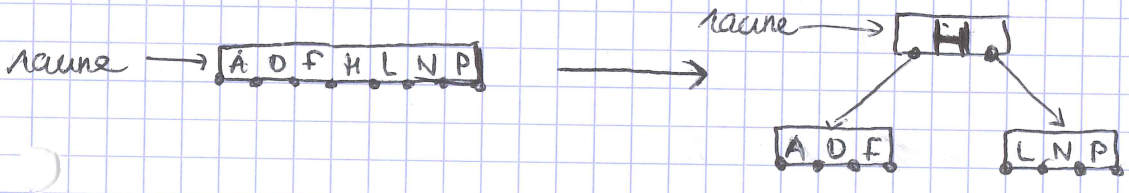
• On est maintenant armés pour insérer une nouvelle clé dans un arbre : on effectue l'insertion par le haut !

On a vu qu'on peut toujours s'en sortir si on vient d'un noeud qui n'est pas plein pour vider celui d'en dessous qui nous intéresse.

Avant de décrire notre algorithme d'insertion, on va s'assurer que la racine de T n'est pas pleine. LECTURE

→ si elle est pleine, on rajoute une racine vide au dessus, notée s, puis on appelle PARTAGER (s, 1)

Exemple avec $t=4$



• Maintenant on peut définir la procédure d'insertion.

INSERER

Entrées : x un noeud supposé non plein
k la clé à insérer.

1) Si x est une feuille, on insère simplement la clé dedans. ECRITURE

2) Sinon

• Parcourir les clés jusqu'à trouver l'enfant de x dans lequel on est susceptibles de pouvoir rajouter la clé k.

• Si ce noeud enfant est plein LECTURE

- le couper en deux en appelant PARTAGER ECRITURE
- choisir l'enfant du côté qui correspond à k.

• INSÉRER récursivement sur le noeud non plein choisi.

- Notons (ECRITURE/LECTURE) à chaque endroit où l'on a accédé au disque dur. Il y en a $O(1)$ pour chaque appel de INSERER.
→ on en effectue donc en tout $O(h)$.

II] - Tout l'intérêt des B-arbres c'est qu'on va pouvoir avoir une complexité agréable sur l'insertion : estimons h !

- Comptons le nombre minimal de [illegible] dans un B-arbre de hauteur h :

Profondeur	0	1	2	...	h
Nb min de clés	1	2	$2t$		$2t^{h-1}$

D'où la relation sur le nombre de clés :

$$n \geq 1 + \underbrace{(2 + 2t + \dots + 2t^{h-1})}_{\text{nombre minimum de noeuds}}\underbrace{(t-1)}_{\text{nombre minimum de clés par noeud.}} = 1 + 2(t-1)\frac{t^h - 1}{t-1} = 2t^h - 1$$

racine

On en déduit donc $h \leq \log_t\left(\frac{n+1}{2}\right)$.

À titre d'exemple, avec $t = 1000$ et $n = 10^9$, on a $h \leq 3$.
ce qui explique que SGL s'en serve pour faire des index :
- on accède très peu de fois au disque dur
- on peut rajouter facilement des clés