Algorithme 2

B-arbres

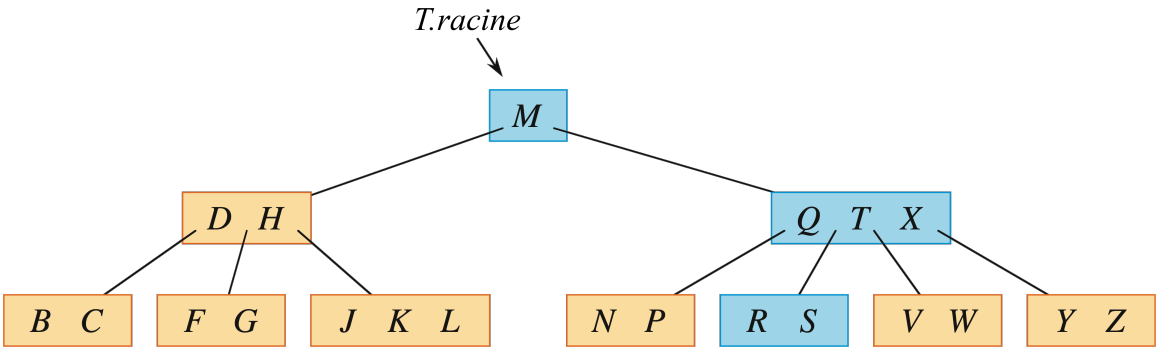
Louis-Claude Canon
louis-claude.canon@univ-fcomte.fr

Licence 2 Informatique – Semestre 4

Plan

Plan

Exemple

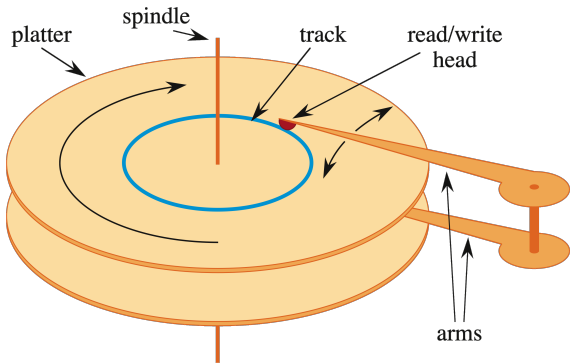


Mémoire centrale et secondaire

- La mémoire centrale (cache et RAM) est rapide, mais de capacité limitée.
- Les très grands volumes de données (To) ne rentrent pas en mémoire centrale.
- Asymétrie prononcée entre les temps de calcul (rapides) et les temps d'accès à la mémoire secondaire (interminables).
- Implication : on cherchera à réduire le nombre d'accès.
  - Chaque accès rapatriera plusieurs élément à la fois : on considérera des blocs de données (quelques ko par exemple).
  - Les algorithmes grouperont les étapes pour éviter de parcourir plusieurs fois les arbres (contrairement aux algorithmes vus pour les arbres rouge-noir).

Notion de blocs

- L'accès à une donnée d'un disque dur nécessite que le plateau qui tourne soit à la bonne position et que le bras avec la tête de lecture se soit positionné correctement.
- Avec des disques durs classiques (7200 tr/mn), une rotation prend 8 ms (des millions de cycles processeurs).
- En moyenne, le temps d'accès est de 3 à 9 ms.
- Lire un octet ou une page entière prend le même temps.
- Les opérations sur les SSDs se basent aussi sur la notion de blocs (avec un impact moins marquée mais qui reste majeur).



Aspect algorithmique

- On suppose que les données ne sont plus directement accessibles (il y en a trop) et qu'il faut donc aller les chercher.
- Notation explicite pour indiquer les accès mémoires (sauf pour le nœud racine qui est supposé être toujours en mémoire).
- L'analyse de la complexité en temps différenciera le temps de calcul du processeur des accès mémoires.

$x \leftarrow$ un pointeur sur un objet
LIRE-DISQUE(x)
`// opérations en lecture/écriture sur x`
ÉCRIRE-DISQUE(x)
`// opérations en lecture sur x`

Application : bases de données

- Les index doivent souvent être mis en place pour optimiser les requêtes SQL.
- Un index, selon son type, permettra de réduire le temps d'accès à une ligne d'une table en évitant de toutes les parcourir.
- Utilisation de B-arbres (ou variantes) : PostgreSQL, MariaDB, SQLite, SQL Server, Oracle Database, etc.

Application : systèmes de fichiers

- Comment trouver l'adresse du i-ième bloc d'un fichier sur le disque ?
- Comme on autorise l'augmentation de la taille d'un fichier, on utilise des structures de données à base de pointeur.
- L'arbre associe une correspondance entre les blocs du fichier et les adresses sur le disque.
- Utilisation de B-arbres (ou variantes) : APFS (Apple), NTFS (Microsoft), ext4 (Linux), etc.

Plan

Définition

Un *B-arbre* (1970) T est un arbre dont la racine est $T.racine$ avec les propriétés suivantes :

- Chaque nœud contient les champs suivants :
  - $x.n$: le nombre de clés dans le nœud ;
  - $x.clé_1, x.clé_2, \ldots, x.clé_{x.n}$: les clés elles-mêmes ;
  - $x.feuille$: un booléen vrai si le nœud est une feuille ;
  - $x.c_1, x.c_2, \ldots, x.c_{x.n+1}$: les enfants de x si le nœud n'est pas une feuille.
- Les clés respectent la propriété de recherche suivante : si k_i est une clé stockée dans $x.c_i$ ou un de ses descendants, alors $k_1 \leq x.clé_1 \leq k_2 \leq x.clé_2 \leq \ldots \leq x.clé_{x.n} \leq k_{x.n+1}$.
- Toutes les feuilles ont la même profondeur.
- Pour chaque nœud x (hors racine), $t - 1 \leq x.n \leq 2t - 1$ où $t \geq 2$ est le *degré minimal* du B-arbre (le degré étant le nombre d'enfants). Un nœud possède entre t et $2t$ enfants.
- Les parcours préfixe et postfixe sont bien définis, mais pas le parcours infixe.

Définition

Un *B-arbre* (1970) T est un arbre dont la racine est $T.racine$ avec les propriétés suivantes :

- Chaque nœud contient les champs suivants :
  - $x.n$: le nombre de clés dans le nœud ;
  - $x.clé_1, x.clé_2, \ldots, x.clé_{x.n}$: les clés elles-mêmes ;
  - $x.feuille$: un booléen vrai si le nœud est une feuille ;
  - $x.c_1, x.c_2, \ldots, x.c_{x.n+1}$: les enfants de x si le nœud n'est pas une feuille.
- Les clés respectent la propriété de recherche suivante : si k_i est une clé stockée dans $x.c_i$ ou un de ses descendants, alors $k_1 \leq x.clé_1 \leq k_2 \leq x.clé_2 \leq \ldots \leq x.clé_{x.n} \leq k_{x.n+1}$.
- Toutes les feuilles ont la même profondeur.
- Pour chaque nœud x (hors racine), $t - 1 \leq x.n \leq 2t - 1$ où $t \geq 2$ est le *degré minimal* du B-arbre (le degré étant le nombre d'enfants). Un nœud possède entre t et $2t$ enfants.
- Les parcours préfixe et postfixe sont bien définis, mais pas le parcours infixe.

Définition

Un *B-arbre* (1970) T est un arbre dont la racine est $T.racine$ avec les propriétés suivantes :

- Chaque nœud contient les champs suivants :
  - $x.n$: le nombre de clés dans le nœud ;
  - $x.clé_1$, $x.clé_2$, ..., $x.clé_{x.n}$: les clés elles-mêmes ;
  - $x.feuille$: un booléen vrai si le nœud est une feuille ;
  - $x.c_1$, $x.c_2$, ..., $x.c_{x.n+1}$: les enfants de x si le nœud n'est pas une feuille.
- Les clés respectent la propriété de recherche suivante : si k_i est une clé stockée dans $x.c_i$ ou un de ses descendants, alors $k_1 \leq x.clé_1 \leq k_2 \leq x.clé_2 \leq \ldots \leq x.clé_{x.n} \leq k_{x.n+1}$.
- Toutes les feuilles ont la même profondeur.
- Pour chaque nœud x (hors racine), $t - 1 \leq x.n \leq 2t - 1$ où $t \geq 2$ est le *degré minimal* du B-arbre (le degré étant le nombre d'enfants). Un nœud possède entre t et $2t$ enfants.
- Les parcours préfixe et postfixe sont bien définis, mais pas le parcours infixe.

Définition

Un *B-arbre* (1970) T est un arbre dont la racine est $T.racine$ avec les propriétés suivantes :

- Chaque nœud contient les champs suivants :
  - $x.n$: le nombre de clés dans le nœud ;
  - $x.clé_1$, $x.clé_2$, ..., $x.clé_{x.n}$: les clés elles-mêmes ;
  - $x.feuille$: un booléen vrai si le nœud est une feuille ;
  - $x.c_1$, $x.c_2$, ..., $x.c_{x.n+1}$: les enfants de x si le nœud n'est pas une feuille.
- Les clés respectent la propriété de recherche suivante : si k_i est une clé stockée dans $x.c_i$ ou un de ses descendants, alors $k_1 \leq x.clé_1 \leq k_2 \leq x.clé_2 \leq \ldots \leq x.clé_{x.n} \leq k_{x.n+1}$.
- Toutes les feuilles ont la même profondeur.
- Pour chaque nœud x (hors racine), $t - 1 \leq x.n \leq 2t - 1$ où $t \geq 2$ est le *degré minimal* du B-arbre (le degré étant le nombre d'enfants). Un nœud possède entre t et $2t$ enfants.
- Les parcours préfixe et postfixe sont bien définis, mais pas le parcours infixe.

Définition

Un *B-arbre* (1970) T est un arbre dont la racine est $T.racine$ avec les propriétés suivantes :

- Chaque nœud contient les champs suivants :
  - $x.n$: le nombre de clés dans le nœud ;
  - $x.clé_1$, $x.clé_2$, ..., $x.clé_{x.n}$: les clés elles-mêmes ;
  - $x.feuille$: un booléen vrai si le nœud est une feuille ;
  - $x.c_1$, $x.c_2$, ..., $x.c_{x.n+1}$: les enfants de x si le nœud n'est pas une feuille.
- Les clés respectent la propriété de recherche suivante : si k_i est une clé stockée dans $x.c_i$ ou un de ses descendants, alors $k_1 \leq x.clé_1 \leq k_2 \leq x.clé_2 \leq \ldots \leq x.clé_{x.n} \leq k_{x.n+1}$.
- Toutes les feuilles ont la même profondeur.
- Pour chaque nœud x (hors racine), $t - 1 \leq x.n \leq 2t - 1$ où $t \geq 2$ est le *degré minimal* du B-arbre (le degré étant le nombre d'enfants). Un nœud possède entre t et $2t$ enfants.
- Les parcours préfixe et postfixe sont bien définis, mais pas le parcours infixe.

Définition

Un *B-arbre* (1970) T est un arbre dont la racine est $T.racine$ avec les propriétés suivantes :

- Chaque nœud contient les champs suivants :
  - $x.n$: le nombre de clés dans le nœud ;
  - $x.clé_1$, $x.clé_2$, ..., $x.clé_{x.n}$: les clés elles-mêmes ;
  - $x.feuille$: un booléen vrai si le nœud est une feuille ;
  - $x.c_1$, $x.c_2$, ..., $x.c_{x.n+1}$: les enfants de x si le nœud n'est pas une feuille.
- Les clés respectent la propriété de recherche suivante : si k_i est une clé stockée dans $x.c_i$ ou un de ses descendants, alors $k_1 \leq x.clé_1 \leq k_2 \leq x.clé_2 \leq \ldots \leq x.clé_{x.n} \leq k_{x.n+1}$.
- Toutes les feuilles ont la même profondeur.
- Pour chaque nœud x (hors racine), $t - 1 \leq x.n \leq 2t - 1$ où $t \geq 2$ est le *degré minimal* du B-arbre (le degré étant le nombre d'enfants). Un nœud possède entre t et $2t$ enfants.
- Les parcours préfixe et postfixe sont bien définis, mais pas le parcours infixe.

Définition

Un *B-arbre* (1970) T est un arbre dont la racine est $T.racine$ avec les propriétés suivantes :

- Chaque nœud contient les champs suivants :
  - $x.n$: le nombre de clés dans le nœud ;
  - $x.clé_1$, $x.clé_2$, ..., $x.clé_{x.n}$: les clés elles-mêmes ;
  - $x.feuille$: un booléen vrai si le nœud est une feuille ;
  - $x.c_1$, $x.c_2$, ..., $x.c_{x.n+1}$: les enfants de x si le nœud n'est pas une feuille.
- Les clés respectent la propriété de recherche suivante : si k_i est une clé stockée dans $x.c_i$ ou un de ses descendants, alors $k_1 \leq x.clé_1 \leq k_2 \leq x.clé_2 \leq \ldots \leq x.clé_{x.n} \leq k_{x.n+1}$.
- Toutes les feuilles ont la même profondeur.
- Pour chaque nœud x (hors racine), $t - 1 \leq x.n \leq 2t - 1$ où $t \geq 2$ est le *degré minimal* du B-arbre (le degré étant le nombre d'enfants). Un nœud possède entre t et $2t$ enfants.
- Les parcours préfixe et postfixe sont bien définis, mais pas le parcours infixe.

Définition

Un *B-arbre* (1970) T est un arbre dont la racine est $T.racine$ avec les propriétés suivantes :

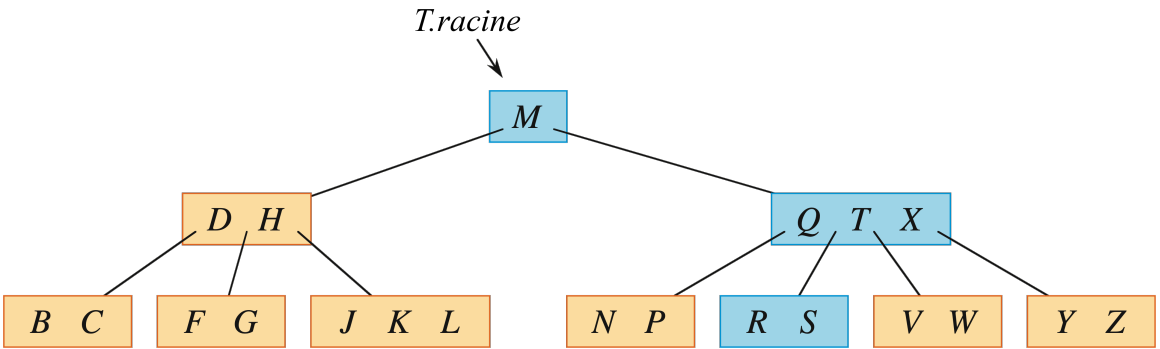
- Chaque nœud contient les champs suivants :
  - $x.n$: le nombre de clés dans le nœud ;
  - $x.clé_1$, $x.clé_2$, ..., $x.clé_{x.n}$: les clés elles-mêmes ;
  - $x.feuille$: un booléen vrai si le nœud est une feuille ;
  - $x.c_1$, $x.c_2$, ..., $x.c_{x.n+1}$: les enfants de x si le nœud n'est pas une feuille.
- Les clés respectent la propriété de recherche suivante : si k_i est une clé stockée dans $x.c_i$ ou un de ses descendants, alors $k_1 \leq x.clé_1 \leq k_2 \leq x.clé_2 \leq \ldots \leq x.clé_{x.n} \leq k_{x.n+1}$.
- Toutes les feuilles ont la même profondeur.
- Pour chaque nœud x (hors racine), $t - 1 \leq x.n \leq 2t - 1$ où $t \geq 2$ est le *degré minimal* du B-arbre (le degré étant le nombre d'enfants). Un nœud possède entre t et $2t$ enfants.
- Les parcours préfixe et postfixe sont bien définis, mais pas le parcours infixe.

Définition

Un *B-arbre* (1970) T est un arbre dont la racine est $T.racine$ avec les propriétés suivantes :

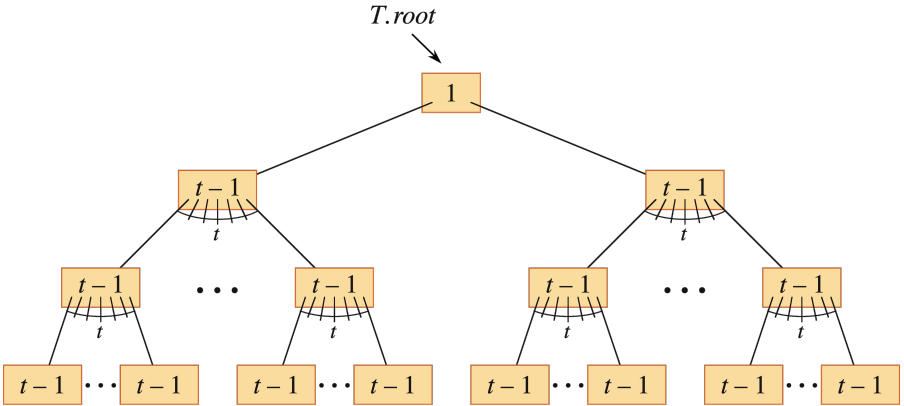
- Chaque nœud contient les champs suivants :
  - $x.n$: le nombre de clés dans le nœud ;
  - $x.clé_1$, $x.clé_2$, ..., $x.clé_{x.n}$: les clés elles-mêmes ;
  - $x.feuille$: un booléen vrai si le nœud est une feuille ;
  - $x.c_1$, $x.c_2$, ..., $x.c_{x.n+1}$: les enfants de x si le nœud n'est pas une feuille.
- Les clés respectent la propriété de recherche suivante : si k_i est une clé stockée dans $x.c_i$ ou un de ses descendants, alors $k_1 \leq x.clé_1 \leq k_2 \leq x.clé_2 \leq \ldots \leq x.clé_{x.n} \leq k_{x.n+1}$.
- Toutes les feuilles ont la même profondeur.
- Pour chaque nœud x (hors racine), $t - 1 \leq x.n \leq 2t - 1$ où $t \geq 2$ est le *degré minimal* du B-arbre (le degré étant le nombre d'enfants). Un nœud possède entre t et $2t$ enfants.
- Les parcours préfixe et postfixe sont bien définis, mais pas le parcours infixe.

Exemple



Hauteur d'un B-arbre

On cherche la hauteur maximale d'un B-arbre de degré minimal t contenant n clés (ou, par équivalence, le nombre minimum de clés que l'on peut mettre dans un arbre de hauteur h). On considère le pire cas : chaque nœud possède le nombre minimal d'enfant.



depth	number of nodes
0	1
1	2
2	$2t$
3	$2t^2$

Analyse mathématique

Dans un B-arbre de hauteur h et de degré minimal t, le nombre de clés n est au moins :

$$n \geq 1 + (t-1)\sum_{i=1}^{h} 2t^{i-1}$$

Notion mathématique

Rappel, pour un réel $x \neq 1$:

$$\sum_{k=0}^{n} x^k = \frac{x^{n+1}-1}{x-1}$$

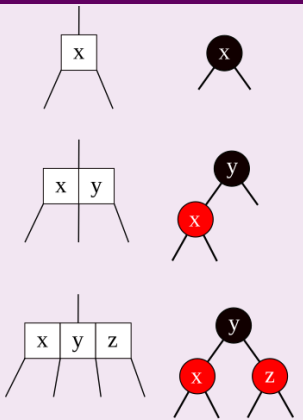
$$n \geq 1 + 2(t-1)\frac{t^h - 1}{t-1} = 2t^h - 1$$

Donc, $h \leq \log_t \frac{n+1}{2}$.

B-arbres et arbres rouge-noir

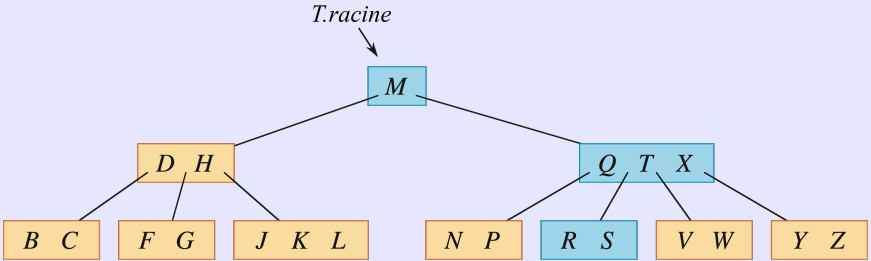
Curiosité

Si chaque nœud noir d'un arbre rouge-noir absorbe ses enfants rouges, en incorporant leurs enfants aux siens, on obtient un B-arbre de degré minimal $t = 2$, que l'on appelle aussi un *arbre 2-3-4*.



Question

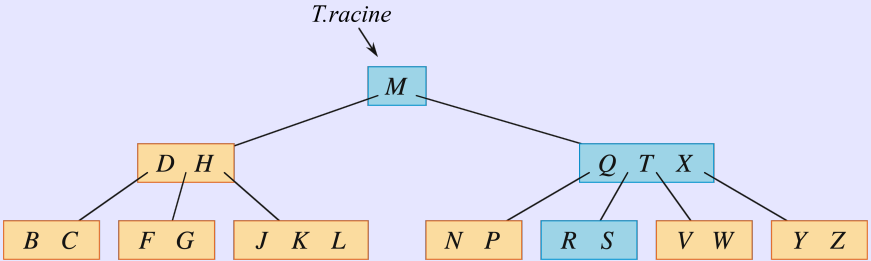
Pour quelles valeurs de degré minimal t l'arbre suivant est-il un B-arbre valide ?



1. $t = 1$
2. $t = 2$
3. $t = 3$
4. $t = 4$

Question

Pour quelles valeurs de degré minimal t l'arbre suivant est-il un B-arbre valide ?



1. $t = 1$
2. $t = 2$ ✓
3. $t = 3$ ✓
4. $t = 4$

Plan

Rechercher-B-Arbre

RECHERCHER-B-ARBRE(x, k)

$i \leftarrow 1$
tant que $i \leq x.n$ **et** $k > x.\text{clé}_i$ **faire**
  $i \leftarrow i + 1$
si $i \leq x.n$ **et** $k = x.\text{clé}_i$ **alors**
  retourner (x, i)
si $x.\textit{feuille}$ **alors**
  retourner *NIL*
sinon
  LIRE-DISQUE$(x.c_i)$
  retourner RECHERCHER-B-ARBRE$(x.c_i, k)$

- Généralisation directe de RECHERCHER-ARBRE.
- On l'appelle avec RECHERCHE-B-ARBRE $(T.\textit{racine}, k)$.
- Nombre d'accès disque : $h - 1$ lectures donc $O(\log_t n)$.
- Temps CPU total : $O(t)$ comparaisons par nœud, donc $O(th) = O(t \log_t n)$ pour l'ensemble des nœuds sur le chemin de recherche.

Créer-B-Arbre

```
Créer-B-Arbre(T)
x ← Allouer-Nœud()
x.feuille ← Vrai
x.n ← 0
Écrire-Disque(x)
T.racine ← x
```

- Initialisation d'un arbre.
- Prend $\Theta(1)$ opérations.

Principe de l'insertion

Contraintes :

- Respecter les propriétés des B-arbres.
- Réduire les accès aux nœuds, donc un unique parcours de la racine à une feuille (il faut éviter la stratégie des arbres rouge-noir où l'on rééquilibre en remontant jusqu'à la racine).

Mécanisme :

- On parcours les nœuds pour trouver la feuille où insérer la nouvelle clé.
- À chaque fois qu'un nœud sur le chemin est complet (de degré maximum $2t$), on le partage en deux.
- Cela garantit qu'il y a toujours une place pour la clé.
- Pour éviter de remonter l'arbre (pas de pointeur vers le parent), ce n'est pas le nœud courant que l'on partage, mais l'un des enfants (sauf pour la racine).

Principe de l'insertion

Contraintes :

- Respecter les propriétés des B-arbres.
- Réduire les accès aux nœuds, donc un unique parcours de la racine à une feuille (il faut éviter la stratégie des arbres rouge-noir où l'on rééquilibre en remontant jusqu'à la racine).

Mécanisme :

- On parcours les nœuds pour trouver la feuille où insérer la nouvelle clé.
- À chaque fois qu'un nœud sur le chemin est complet (de degré maximum $2t$), on le partage en deux.
- Cela garantit qu'il y a toujours une place pour la clé.
- Pour éviter de remonter l'arbre (pas de pointeur vers le parent), ce n'est pas le nœud courant que l'on partage, mais l'un des enfants (sauf pour la racine).

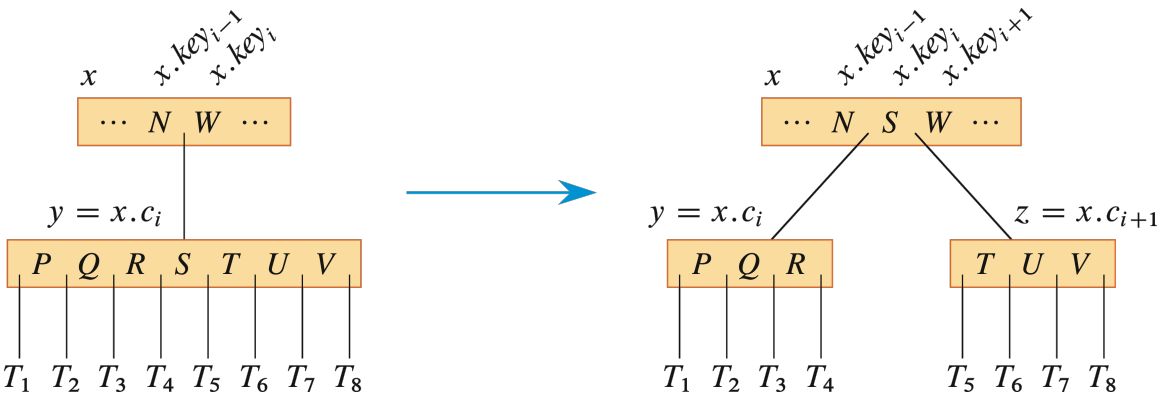
Description de Partager-Enfant-B-Arbre

- C'est la procédure qui permet de diviser un nœud en deux et qui garantit qu'il y aura de la place pour la clé à insérer.
- Elle s'applique sur un nœud pour diviser son i-ième enfant qui est de degré $2t$.
- Elle prélève la clé médiane (d'indice t) de l'enfant, l'insère dans le nœud courant et crée deux nœuds enfants (y et z) avec les $2t - 2$ clés restantes.

Pseudo-code de Partager-Enfant-B-Arbre

```
PARTAGER-ENFANT-B-ARBRE(x, i)
y ← x.c_i
z ← ALLOUER-NŒUD()
z.feuille ← y.feuille
z.n ← t − 1
pour j ← 1 à t − 1 faire
    z.clé_j ← y.clé_{j+t}
si non y.feuille alors
    pour j ← 1 à t faire
        z.c_j ← y.c_{j+t}
y.n ← t − 1
pour j ← x.n + 1 à i + 1 faire
    x.c_{j+1} ← x.c_j
x.c_{i+1} ← z
pour j ← x.n à i faire
    x.clé_{j+1} ← x.clé_j
x.clé_i ← y.clé_t
x.n ← x.n + 1
ÉCRIRE-DISQUE(y)
ÉCRIRE-DISQUE(z)
ÉCRIRE-DISQUE(x)
```

Illustration de Partager-Enfant-B-Arbre

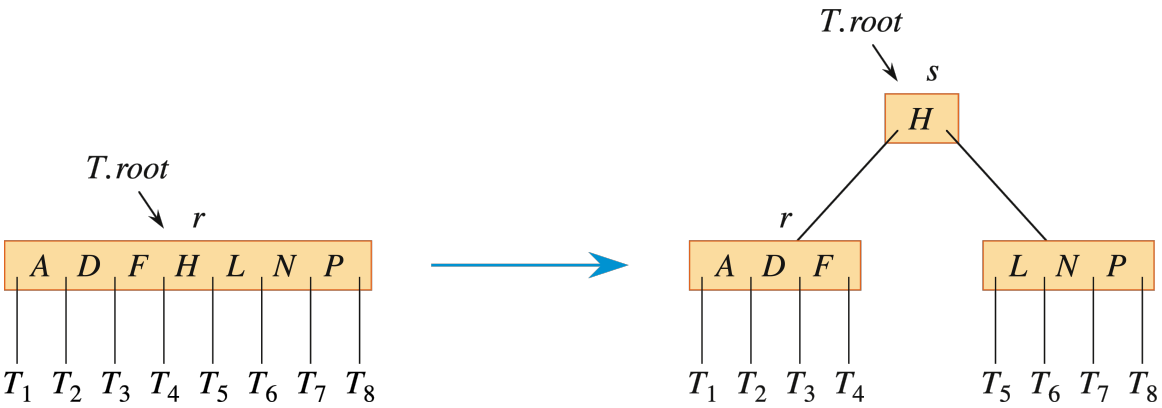


Pseudo-code d'Insérer-B-Arbre

```
Insérer-B-Arbre(T, k)
r ← T.racine
si r.n = 2t − 1 alors
    s ← Allouer-Nœud()
    T.racine ← s
    s.feuille ← Faux
    s.n ← 0
    s.c_1 ← r
    Partager-Enfant-B-Arbre(s, 1)
    Insérer-B-Arbre-Incomplet(s, k)
sinon
    Insérer-B-Arbre-Incomplet(r, k)
```

- Cas particulier du partage de la racine si elle est complète.
- Dans ce cas, on crée un nouveau nœud avec une seule clé.
- C'est le seul cas où la hauteur de l'arbre change.
- La recherche de la feuille où insérer la clé continue avec Insérer-B-Arbre-Incomplet.

Illustration d'Insérer-B-Arbre



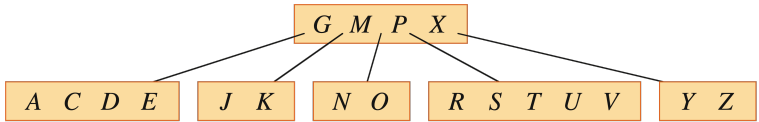
Description d'Insérer-B-Arbre-Incomplet

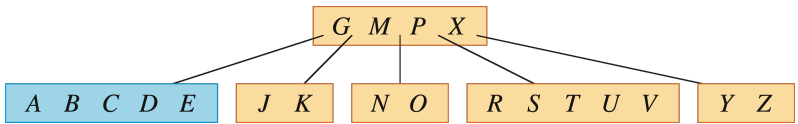
- Si le nœud est une feuille, on y insère la clé.
- Sinon le nœud courant n'est pas une feuille et on cherche l'enfant adapté pour insérer la clé (en respectant la propriété de recherche).
- Si cet enfant est complet, on le partage.

Pseudo-code d'Insérer-B-Arbre-Incomplet

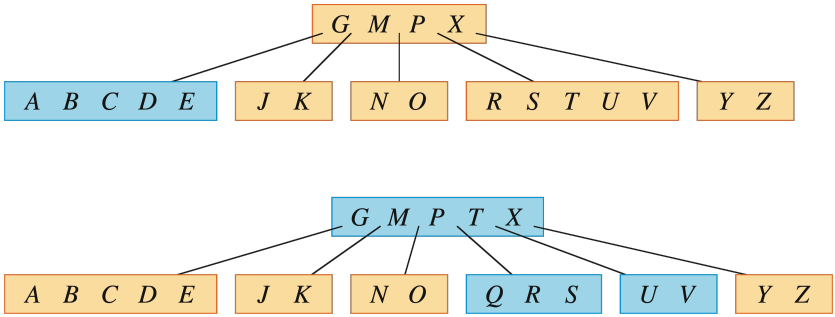
```
Insérer-B-Arbre-Incomplet(x, k)
i ← x.n
si x.feuille alors
    tant que i ≥ 1 et k < x.clé_i faire
        x.clé_{i+1} ← x.clé_i
        i ← i − 1
    x.clé_{i+1} ← k
    x.n ← x.n + 1
    Écrire-Disque(x)
sinon
    tant que i ≥ 1 et k < x.clé_i faire
        i ← i − 1
    i ← i + 1
    Lire-Disque(x.c_i)
    si x.c_i.n = 2t − 1 alors
        Partager-Enfant-B-Arbre(x, i)
        si k > x.clé_i alors
            i ← i + 1
    Insérer-B-Arbre-Incomplet(x.c_i, k)
```

Exemple d'insertions : **B**, Q, L et F ($t = 3$)

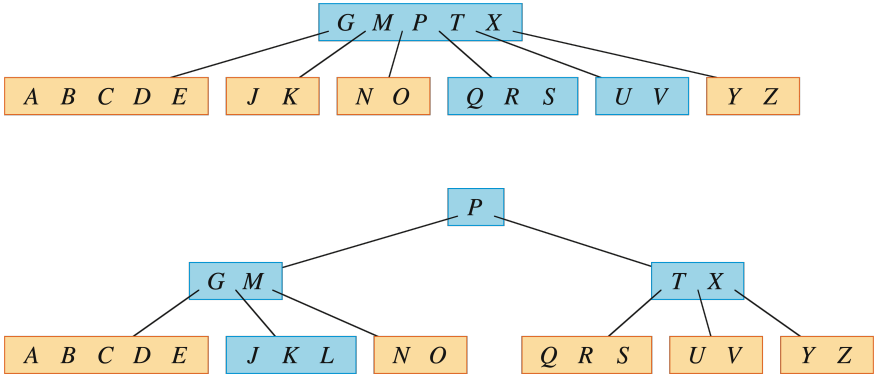




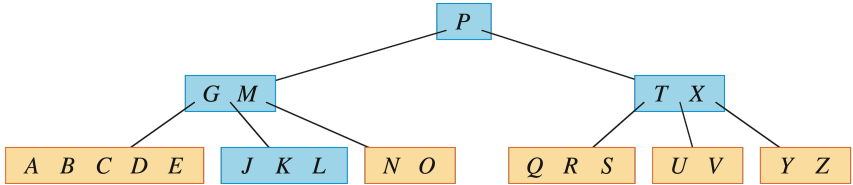
Exemple d'insertions : B, **Q**, L et F ($t = 3$)

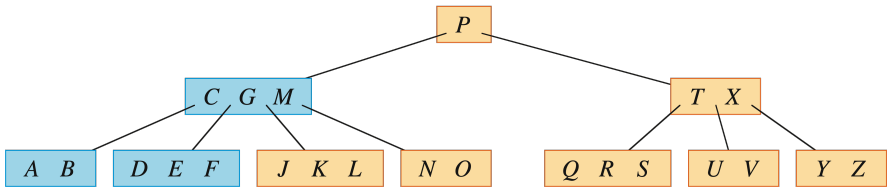


Exemple d'insertions : B, Q, **L** et F ($t = 3$)



Exemple d'insertions : B, Q, L et **F** ($t = 3$)





Analyse de complexité

- Nombre d'accès disque :
  - $h - 1$ lectures (on ne lit pas la racine).
  - Au plus $3h + 1$ écritures (on partage tous les nœuds sur le chemin de recherche).
  - Au total : $O(h) = O(\log_t n)$.
- Temps CPU total :
  - $O(t)$ itérations pour chaque partage.
  - $O(t)$ itérations la recherche sur chaque nœud.
  - Au total : $O(th) = O(t \log_t n)$.

Preuve de validité

Invariant de boucle Au début de chaque appel à la fonction `Insérer-B-Arbre-Incomplet`, le nœud parcouru x a un degré au plus $2t - 1$.

Initialisation Le premier appel se fait soit sur s qui est issu d'un partage et donc de degré t, ou sur la racine r si elle est de degré au plus $2t - 1$.

Conservation Le prochain nœud parcouru, $x.c_i$, est partagé s'il est de degré $2t$. Au prochain appel de la fonction, le nouveau nœud x sera donc soit de degré t (s'il a été partagé), soit de degré au plus $2t - 1$.

Terminaison Comme la clé est insérée sur une feuille et que celle-ci est de degré au plus $2t - 1$, à l'issue de l'insertion, l'ensemble des nœuds modifiés, feuille comprise, respectent bien les propriétés de degré du B-arbre.

Preuve de validité

Invariant de boucle Au début de chaque appel à la fonction `Insérer-B-Arbre-Incomplet`, le nœud parcouru x a un degré au plus $2t - 1$.

Initialisation Le premier appel se fait soit sur s qui est issu d'un partage et donc de degré t, ou sur la racine r si elle est de degré au plus $2t - 1$.

Conservation Le prochain nœud parcouru, $x.c_i$, est partagé s'il est de degré $2t$. Au prochain appel de la fonction, le nouveau nœud x sera donc soit de degré t (s'il a été partagé), soit de degré au plus $2t - 1$.

Terminaison Comme la clé est insérée sur une feuille et que celle-ci est de degré au plus $2t - 1$, à l'issue de l'insertion, l'ensemble des nœuds modifiés, feuille comprise, respectent bien les propriétés de degré du B-arbre.

Preuve de validité

Invariant de boucle Au début de chaque appel à la fonction `Insérer-B-Arbre-Incomplet`, le nœud parcouru x a un degré au plus $2t - 1$.

Initialisation Le premier appel se fait soit sur s qui est issu d'un partage et donc de degré t, ou sur la racine r si elle est de degré au plus $2t - 1$.

Conservation Le prochain nœud parcouru, $x.c_i$, est partagé s'il est de degré $2t$. Au prochain appel de la fonction, le nouveau nœud x sera donc soit de degré t (s'il a été partagé), soit de degré au plus $2t - 1$.

Terminaison Comme la clé est insérée sur une feuille et que celle-ci est de degré au plus $2t - 1$, à l'issue de l'insertion, l'ensemble des nœuds modifiés, feuille comprise, respectent bien les propriétés de degré du B-arbre.

Preuve de validité

Invariant de boucle Au début de chaque appel à la fonction `Insérer-B-Arbre-Incomplet`, le nœud parcouru x a un degré au plus $2t - 1$.

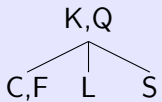
Initialisation Le premier appel se fait soit sur s qui est issu d'un partage et donc de degré t, ou sur la racine r si elle est de degré au plus $2t - 1$.

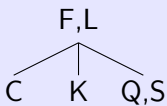
Conservation Le prochain nœud parcouru, $x.c_i$, est partagé s'il est de degré $2t$. Au prochain appel de la fonction, le nouveau nœud x sera donc soit de degré t (s'il a été partagé), soit de degré au plus $2t - 1$.

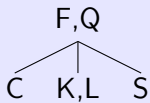
Terminaison Comme la clé est insérée sur une feuille et que celle-ci est de degré au plus $2t - 1$, à l'issue de l'insertion, l'ensemble des nœuds modifiés, feuille comprise, respectent bien les propriétés de degré du B-arbre.

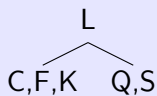
Question

Montrer le résultat de l'insertion successive des clés F, S, Q, K, C et L dans un B-arbre vide de degré minimal $t = 2$.

1. 

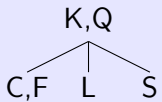
2. 

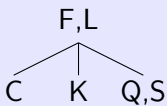
3. 

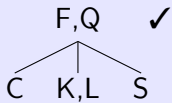
4. 

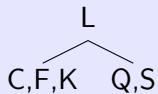
Question

Montrer le résultat de l'insertion successive des clés F, S, Q, K, C et L dans un B-arbre vide de degré minimal $t = 2$.

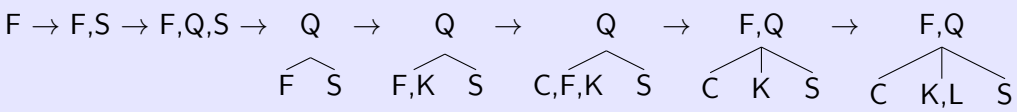
1. 

2. 

3. ✓ 

4. 

Correction



Plan

Principe

- Similaire à l'insertion : pendant que l'on cherche l'élément à supprimer, on fusionne les nœuds qui n'ont pas assez d'enfants.
- L'algorithme garantit que chaque nœud parcouru possède au moins t clés (une de plus que le minimum requis).
- La clé à supprimer n'est pas forcément sur une feuille, il y a donc plus de cas à considérer.
- L'algorithme va parcourir l'arbre jusqu'à l'élément à supprimer. Chaque nœud parcouru est soit une feuille, soit un nœud interne qui contient l'élément, soit un nœud interne qui ne le contient pas.

Plan

Résumé

Contenu

- Les B-arbres sont des structures de données adaptés pour rechercher des éléments et pour les parcourir dans l'ordre (comme les arbres binaires de recherche).
- Ces arbres sont auto-équilibrés et optimisés pour réduire le nombre d'accès à la mémoire secondaire et exploiter leur organisation en blocs ($t = 1024$ par exemple).
- Les opérations classiques restent simples, tandis que l'insertion et la suppression nécessitent de s'assurer que le degré de chaque nœud respecte la propriété des B-arbres.

Prochaines échéances

- QCM à la prochaine séance de CM.
- Mini-projet : à rendre la semaine du 24/2.
- DS 1 : semaine du 24/3.

Résumé

Contenu

- Les B-arbres sont des structures de données adaptés pour rechercher des éléments et pour les parcourir dans l'ordre (comme les arbres binaires de recherche).
- Ces arbres sont auto-équilibrés et optimisés pour réduire le nombre d'accès à la mémoire secondaire et exploiter leur organisation en blocs ($t = 1024$ par exemple).
- Les opérations classiques restent simples, tandis que l'insertion et la suppression nécessitent de s'assurer que le degré de chaque nœud respecte la propriété des B-arbres.

Prochaines échéances

- QCM à la prochaine séance de CM.
- Mini-projet : à rendre la semaine du 24/2.
- DS 1 : semaine du 24/3.