

# Les structures de données arborescentes avancées

## 1 Objectifs

Construire des structures de données adaptées à des opérations privilégiées. En particulier, des structures de données qui supportent plusieurs opérations dynamiques telles que celles-ci :

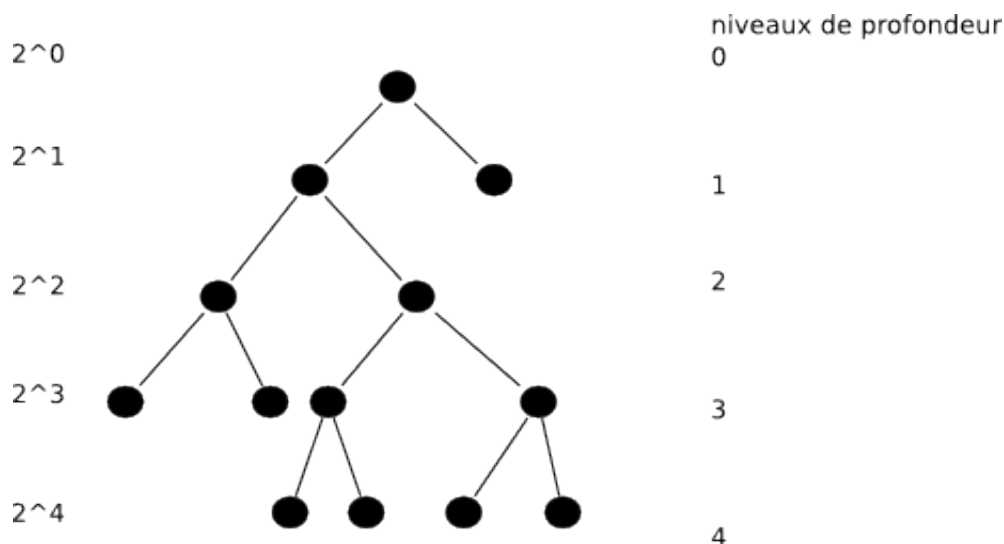
- Recherche, insertion, suppression, modification, successeur, prédécesseur
- Minimum, maximum, extraire-minimum, extraire-maximum
- Union, find

En informatique, les données sont stockées sous forme d'enregistrements, dotés de clés. Nous considérons les structures suivantes :

- Structures linéaires (tableaux, listes)
- Structures arborescentes (ABR, Arbre Rouge et Noir ou ARN), B-arbres, arbres binaires, tas binaires, tas de fibonacci).
- Structures avec hachage

### 1.1 Retour sur les arbres

Un arbre est un graphe acyclique connexe. Un noeud particulier sont les racines et les feuilles. La hauteur de l'arbre est la longueur maximale qui sépare la racine de la feuille la plus éloignée. La hauteur d'un sommet est la longueur maximale par rapport à la feuille la plus éloignée.



Le nombre de noeuds d'un arbre complet est de la forme :

$$2^h = 2^{h+1} - 1$$

, avec  $h$  la hauteur.

Si  $n$  est le nombre de noeuds (ou sommets) d'un arbre binaire complet, alors

$$2^h \leq n \leq 2^{h+1}$$

$$h = \lfloor \log_2(n) \rfloor$$

Soit  $k$  le niveau d'un sommet  $x$ . Le sous-arbre maximal de racine  $x$  contient  $n(x)$  sommets, alors

$$\frac{n}{2^k + 1} \leq n(x) \leq \frac{n}{2^k}$$

La hauteur de  $x$  est  $\lfloor \log_2(n) \rfloor - h$  ou  $\lfloor \log_2(n) \rfloor - (k + 1)$ .

## 2 Stockage des données dans un SGDB

Dans une base de données Oracle par exemple, les données sont sous forme d'enregistrement (records), eux-mêmes stockés dans des blocks des fichiers, eux-mêmes sur des disques.

La requête SQL si elle doit accéder à chaque donnée pour vérifier sa validité permet de prendre beaucoup de temps, il est nécessaire d'accélérer les opérations de recherche (via des indexes).

### 2.1 Indexation linéaire

L'accès aux données est séquentiel ; les enregistrements sont rangés dans l'ordre croissant des clés.

### 2.2 Remarques

L'indexation linéaire présente des limitations que l'indexation arborescente via des B-Arbres résoud. Nous allons donc étudier cette structure dans la partie suivante.

## 3 Les B-Arbres

Les B-Arbres forment les arbres équilibrés, c'est à dire que toutes les feuilles sont au même niveau.

Enonçons les paradigmes :

- Avoir plus de clés par noeuds
- Avoir un facteur de branchement plus large (centaine, milliers)

### 3.1 Définition

#### 3.1.1 Structure des B-Arbres

Le noeud racine d'un B-Arbre non vide doit avoir au moins une clé et possède donc au moins 2 fils.

Dans un B-Arbre de degré minimum  $t$  :

- Tout noeud autre que la racine doit contenir au moins  $t - 1$  clés et tous noeuds au plus  $2t - 1$  clés.
- Tout noeud interne autre que la racine possède au moins  $t$  fils et au plus  $2t - 1$  fils.
- Un noeud est dit plein s'il contient exactement  $2t - 1$  clés.
- Le B-Arbre le plus simple est l'arbre 2.3.4. : ie B-Arbre de degré minimum 2.
- Si toutes les feuilles sont à la même profondeur (ie : hauteur), l'arbre est équilibré.

Remarque : Les informations satellites sont stockées avec chaque clé ou bien comprises dans les feuilles. Dans ce cas, les noeuds internes ne contiennent que les clés ce qui permet d'ailleurs d'augmenter le facteur de branchement.

#### 3.1.2 Hauteur des B-arbres

Pour un B-arbre à  $n$  clés,  $n \geq 1$ , de hauteur  $h$  et de degré minimum  $t \geq 2$ , on a :

$$h \leq \log_t\left(\frac{n+1}{2}\right)$$

Remarque : Le nombre d'accès disque est proportionnel à la hauteur du B-arbre. On peut répondre à une requête SQL en au plus les entrées / sorties.

### 3.2 Les opérations sur les B-Arbres

Remarque : La racine du B-arbre est toujours présente en mémoire RAM.

#### 3.2.1 Les opérations de recherche

C'est une généralisation de l'opération de recherche d'un ABR.

Soient :

- $n(x)$  : le nombre de noeuds compris dans un noeud  $x$
- $key_i(x)$  :  $i$ -ième clé stockée dans le noeud  $x$
- $leaf(x)$  : booléen indiquant si  $x$  est une feuille ou non
- $C_i(x)$  : un pointeur sur le  $i$ -ième fils de  $x$

```

BTreeSearch(i,k)
  i <- 1
  TantQue i <= n(x) et k > key_i(x) Faire
    i <- i + 1
  FinTantQue
  Si i <= n(x) et k = key_i(x) Alors
    Retourner (x,i)
  Finsi
  Si leaf(x) Alors
    Retourner Nil
  Sinon
    DisqueRead(C_i(x))
  Finsi
  Retourner (C_i(x),k)
Fin

```

**Coût de l'opération :** La hauteur d'un arbre est  $O(h) = O(\log_t(n))$ ,  $n$  étant le nombre de clés de l'arbre. Puisque  $n(x) \leq 2 \cdot t - 1$ , la boucle "TantQue" coûte  $O(t)$ . Par conséquent, le coût total est :

$$O(t \cdot \log_t(n))$$

### 3.2.2 Opérations d'insertion

Elle est basée sur une opération fondamentale d'éclatement de noeuds pleins ayant  $2 \cdot t - 1$  clés autour de la valeur médiane  $key_i(y)$  en deux noeuds ayant chacun  $t - 1$  clés. La clé médiane remonte dans le noeud père de  $y$ , qui doit être non plein. S'il n'y a pas de père, la hauteur de l'arbre croît de 1.

```

BTreeSplit(x;i;y)
  y <- newnode()
  leaf(z) <- leaf(y)
  n[z] <- t - 1
  Pour j allant de 1 à t - 1 Faire
    key_i[z] <- key_(j+t)[y]
  Fin
  Si leaf(y) == faux Alors
    Pour j allant de 1 à t Faire
      C_i[j] <- C_(j+t)[y]
    FinPour
  Finsi
  C_(i+1)[x] <- z
  Pour j allant de n[x] à i Faire

```

```
    key_(j-1)[x] <- key_i[x]
  FinPour
  key_i[x] <- key_t[z]
  n[x] <- n(x) + 1
  DisqueWrite(y)
  DisqueWrite(z)
  DisqueWrite(x)
Fin
```

Le coût est  $O(1)$ .