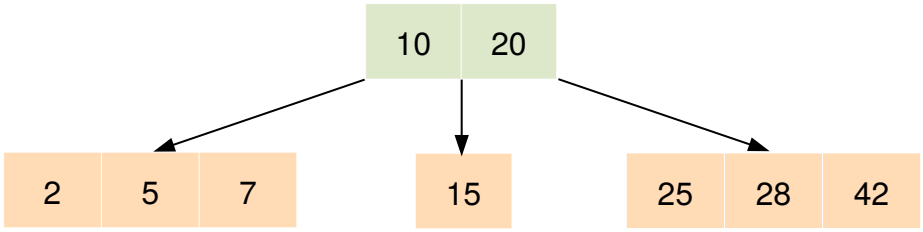
B-Arbres

Référence : 131 Développements pour l'oral, D.Lesesvre, P. Montagnon.

Motivation et exemple. Dans le cas d'une grande quantité d'information, il est parfois nécessaire d'utiliser des supports de stockage dont le temps de réponse pour une lecture est élevé (disque dur, ...). Pour implanter une structure de dictionnaire dans ce cas, on préférera contracter notre ABR afin de faire moins d'appel à la mémoire morte.

Exemple (Un B-arbre d'ordre 2 et de hauteur 2)**.**



Définition.

Définition 1 (B-arbre)**.** *Soit un entier $t \geqslant 2$. On appelle B-arbre d'ordre t un arbre vérifiant les invariants suivants :*

- *Chaque nœud x contient les attribut suivants :*
  1. *le booléen x.feuille (VRAI si X est une feuille, FAUX sinon) ;*
  2. *le nombre $x.n$ de clés dans ce nœud ;*
  3. *le tableau trié x.clé de taille $x.n$ contenant les clés de ce noeud (numéroté de 1 à $x.n$) ;*
  4. *le tableau x.fils de taille $x.n+1$ contenant l'ensemble des enfants du nœud x (numéroté de 0 à $x.n$).*
- *pour tout nœud x, toute clé k_i apparaissant dans le fils $x.fils_i$, on a*

$$k_0 \leqslant x.clé_1 \leqslant k_1 \leqslant x.clé_2 \leqslant ... \leqslant x.clé_n \leqslant k_n$$

- *toutes les feuilles ont la même profondeur h ;*
- *pour tout noeud x non racine, on a $t-1 \leqslant x.n \leqslant 2t-1$;*
- *la racine x_0 vérifie $1 \leqslant x_0.n \leqslant 2t-1$.*

Ici, au vu de son utilisation, on suppose le B-arbres stocké dans une mémoire auxiliaire d'accès lent, on va donc compter notre complexité en fonction du nombre de lecture et d'écriture dans cette mémoire. On considère deux fonctions **LIRE** et **ECRIRE** permettant d'accéder à notre mémoire auxiliaire.

Recherche dans un tel arbre. La fonction de recherche dans un B-arbres ressemble beaucoup à la recherche dans un ABR, sauf que dans chaque nœud, il faut chercher le bon intervalle.

On suppose que notre racine est en mémoire principale, il n'y donc pas besoin de faire de appel à **LIRE**.

On va maintenant étudier la complexité de notre algorithme et comparer à la complexité si on avait utiliser un ABR, c'est-à-dire un B-arbre d'ordre 2.

Algorithme 1 : Recherche(x,c)

Données : un noeud x et une clé c
Résultat : si c est dans un nœud y de l'arbre, on renvoie y et sa place dans le nœud, sinon on renvoie NIL.

```
# Recherche du bon intervalle ;
i ← 1 ; tant que i ⩽ x.n et x > x.clé_i faire
  i ← i + 1 ;
si i ⩽ x.n et c = x.clé_i alors
  retourner x, i
si x.feuille alors
  retourner NIL
f ← LIRE(x.fils_{i−1}) ;
retourner Recherche(f, c)
```

Théorème 1. *Dans un B-arbre T d'ordre $t \geqslant 2$ contenant n clés, sa hauteur h vérifie*

$$h \leqslant \ln_t \left(\frac{n+1}{2} \right)$$

Démonstration. Par définition, T contient au moins une clé à la racine, et donc possède sa racine possède au moins 2 enfants. On a donc :

— au moins 1 nœud à la profondeur 0 ;
— au moins 2 nœuds à la profondeur 1 ;
— au moins $2t$ nœuds à la profondeur 2...

On montre alors par récurrence immédiate qu'on a au moins $2t^{i-1}$ nœud à la profondeur i. Puisque chaque nœud contient au moins $(t-1)$ clés, on a :

$$n \geqslant 1 + (t-1) \sum_{i=1}^{h} 2t^{i-1} = 1 + 2(t-1) \left(\frac{t^h - 1}{t-1} \right) = 2t^h - 1$$

Le résultat en découle alors directement. □

Étant donné un B-arbre T de racine x_0, la fonction Recherche(x,c) fait h appels à **LIRE** dans le pire des cas, que l'on peut borner via le théorème ci-dessus.

Insertion dans un B-arbre. Ici les choses se compliquent puisque l'insertion est moins évidente. Il y a des situations qui peuvent poser problème, par exemple si l'arbre est déjà plein.

Idée de l'algorithme : sur l'entrée c et le nœud x,
— On va à la feuille correspondante en choisissant les chemins via les intervalles ;
— Si la feuille possède strictement moins de $2t-1$ étiquettes, on peut insérer c ;
— Sinon, en notant $c_1, ..., c_{2t}$ les nouvelles étiquettes de cette feuilles (avec $c = c_{2t}$), on sépare cette feuilles en deux feuilles d'étiquettes $c_1, ..., c_{t-1}$ et $c_{t+1}, ..., c_{2t}$, et on remonte c_t comme séparateur dans le parent ;
— Si le parent contient alors trop d'étiquettes, on le coupe en deux de la même manière et ainsi de suite jusqu'à la racine.
— Si la racine doit être coupé en deux, on crée une nouvelle racine qui ne contiendra qu'une étiquette.