

Les structures d'arbres en mémoire secondaire

Introduction

Les méthodes par tables d'index sont limitées à certains type de fichiers (petits fichiers ou fichiers statiques). Les méthodes basées sur les structures d'arbres sont mieux adaptées aux fichier volumineux et/ou dynamiques.

Afin de mieux occuper l'espace des blocs, on utilise des arbres m-aires.

Les arbres de recherche m-aires

Un arbre de recherche m-aires est la généralisation d'un arbre de recherche binaire pour les ordres > 2

Un arbre de recherche m-aires d'ordre n est un arbre où chaque noeud peut avoir au maximum n fils (Fils[1..n]) et $n-1$ valeurs (Val[1..n-1]).

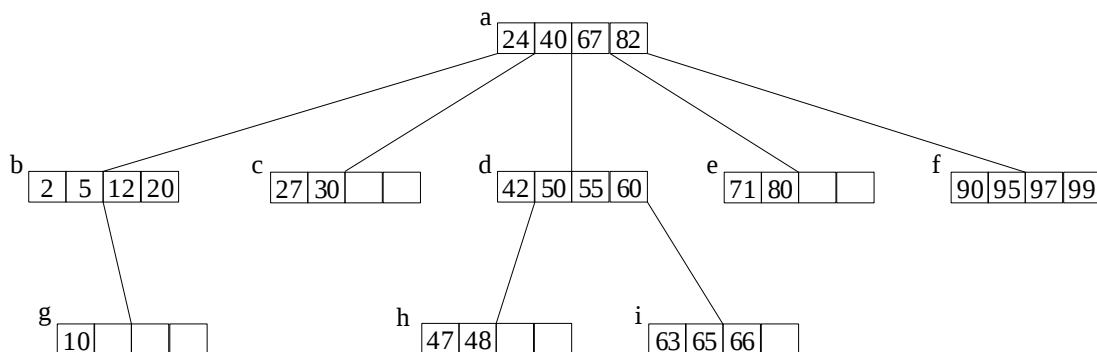
Les valeurs à l'intérieur d'un noeud sont ordonnées en ordre croissant. Les fils sont organisés en fonction des valeurs du noeuds, selon les règles suivantes :

- i) Le Fils[1] pointe un sous-arbre contenant des valeurs $<$ à Val[1]
- ii) Le Fils[i] pointe un sous-arbre contenant des valeurs $>$ Val[i-1] et $<$ Val[i] , pour $i=2..n-1$
- iii) Le Fils[n] pointe un sous-arbre contenant des valeurs $>$ Val[n-1]

Le degré d'un d'un noeud (le nombre de fils) est le nombre de valeurs stockées plus un.

La figure ci-dessous montre un arbre de recherche m-aires d'ordre 5 de racine le noeud a. Le noeud b est le fils 1 de a et f est le fils 5 de a.

Un noeud interne peut avoir certains fils à nil et d'autres non. Par exemple tous les fils de b à part le 3e sont à nil. De même, les fils 1, 3 et 4 du noeud d sont à nil, alors que son fils 2 (pointe h) et son fils 5 (pointe i) sont différents de nil.



La recherche d'une valeur C commence dans le noeud racine P et se poursuit le long d'une branche :

- 1- Si C existe dans P alors stop avec succès
- 2- Si C n'existe pas dans P alors
 - soit k la position dans P où devrait être insérer C (pour que les valeurs restent ordonnées)
 - P := Fils[k]
 - Si P différent de nil alors aller à 1 Sinon stop avec échec.

Par exemple, la recherche de 48 dans l'arbre de la figure précédente se déroule comme suit:

- on commence la recherche dans le noeud a. La valeur 48 n'existe pas et est comprise entre 40 et 67. Donc le prochain noeud à visiter est le Fils[3] (le noeud d)
- la recherche se poursuit dans le noeud d, et le prochain noeud à explorer est le Fils[2], car 48 est comprise entre 42 et 50
- on continue la recherche dans le noeud h. La valeur 48 existe (Val[2]), on s'arrête donc avec succès.

Si on avait recherché la valeur 15, on aurait visité d'abord le noeud a, puis le noeud b (Fils[1] de a, car $15 < 24$). Là on serait arrêté avec un échec, car 15 est comprise entre 12 et 20 et le Fils[4] de b est à nil.

Pour insérer une nouvelle valeur V dans un arbre de recherche m-aires, on recherche d'abord la valeur, pour vérifier qu'elle n'existe pas déjà et pour localiser le noeud P où doit être insérée cette valeur (P est le dernier noeud visité dans la recherche). La recherche retourne aussi l'indice k où devrait être insérée V s'il y a avait de l'espace dans P.

Si P n'est pas plein,

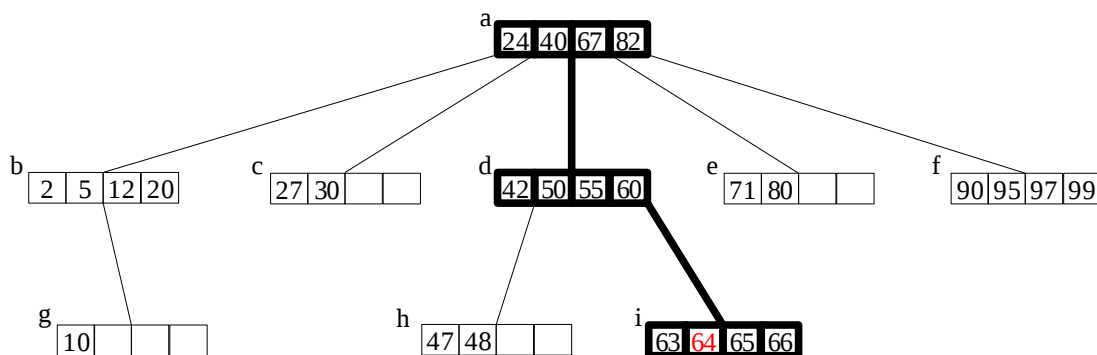
- on insère V dans P, par décalages afin de garder le tableau de valeurs ordonné.

Sinon (P est plein) :

- on alloue un nouveau noeud Q contenant une seule valeur (Val[1] = V) et deux fils à nil (Fils[1] = nil et Fils[2] = nil)
- on fait pointer le nouveau noeud Q par le Fils[k] de P (qui était, avant l'insertion, forcément à nil). L'indice k est celui retourné par la recherche.

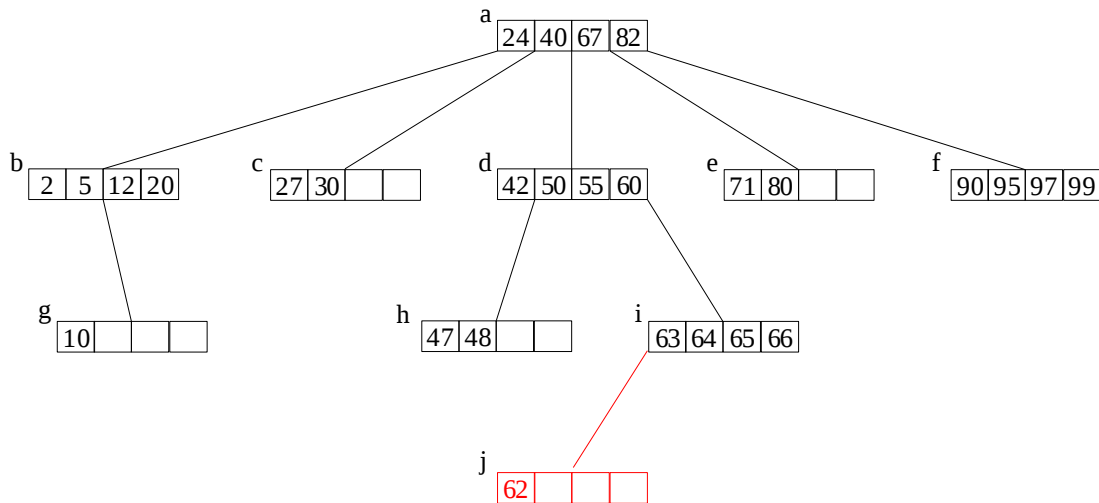
Par exemple, si on insère la valeur 64 dans l'arbre de la figure précédente, on procédera comme suit:

- 1- recherche de 64 → échec (le dernier noeud visité est i et la position où devrait être insérée 64 est $k=2$)
- 2- comme le noeud i n'est pas plein, on peut alors insérer 64 à la position 2 en décalant à droite les valeurs > 64



Si on insère encore la valeur 62, on allouera alors un nouveau noeud (j) qui va contenir 62 et il sera pointé par le Fils[1] du noeud i (i étant le dernier noeud visité dans la recherche de 62 et la position où devrait être insérer 62 dans i, s'il y avait de l'espace, est $k = 1$)

La figure suivante montre le résultat d'une telle insertion:



Un arbre de recherche m-aires est dit « Top-Down » si tous ces noeuds internes sont remplis à 100%.
L'algorithme d'insertion présenté maintient cette propriété.

La suppression d'une valeur V peut être logique (ce qui favorise le maintien de la propriété top-down) ou bien physique en généralisant la suppression des arbres de recherche binaires (mais le maintien de la propriété top-down est plus difficile).

- En mémoire secondaire

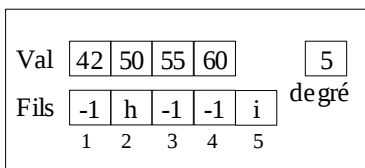
Chaque noeud de l'arbre est représenté par un bloc d'E/S.
La structure générale d'un bloc est alors comme suit :

type

```
Tbloc = structure
    Val : tableau[N-1] de typeqlq;      // enregistrements ou (clés-adr)
    Fils : tableau[N] d'entier;          // numéros de blocs
    degré : entier;                     // nb d'elt dans Fils
Fin;
```

Parmi les caractéristiques d'un fichier vu comme arbre, il y a la racine. C'est le numéro du bloc contenant la racine de l'arbre.

Par exemple, si l'arbre de la figure précédente était un fichier, le contenu du bloc d serait comme suit:



Val	<div style="border: 1px solid black; padding: 2px;">10</div>	<div style="border: 1px solid black; width: 40px; height: 20px;"></div>	<div style="border: 1px solid black; width: 40px; height: 20px;"></div>	<div style="border: 1px solid black; width: 40px; height: 20px;"></div>		<div style="border: 1px solid black; padding: 2px;">2</div>
Fils	<div style="border: 1px solid black; padding: 2px;">-1</div>	<div style="border: 1px solid black; padding: 2px;">-1</div>	<div style="border: 1px solid black; width: 40px; height: 20px;"></div>	<div style="border: 1px solid black; width: 40px; height: 20px;"></div>	<div style="border: 1px solid black; width: 40px; height: 20px;"></div>	dégré
	1	2	3	4	5	

```
Rech(c:Typeqlq; nomf:chaine; var trouv:boolean; var i,j,prec:entier)
/*
```

Debut

Fin

```
Ins( e:Tenreg; nomf:chaîne )
```


Debut

4

```

    Aff_entete( F, 1, i );      // la nouvelle racine
    buf.degre := 1;
    buf.Val[1] := e;
    buf.Fils[1] := -1;
    buf.Fils[2] := -1;
    EcrireDir( F, i, buf );
Sinon
    Rech( e.cle, " , trouv, i, j, prec );
    Si ( Non trouv )
        // si le dernier bloc visité n'est pas plein, on y insère l'enreg e ...
        Si ( buf.degre < N )
            // décalages interne à partir de j ...
            Pour k := buf.degre, j+1, -1 // boucle arrière avec pas = -1
                buf.Val[k] := buf.Val[k-1];
                buf.Fils[k+1] := buf.Fils[k];
            FinPour
            // insérer e à la pos j
            buf.Val[j] := e;
            // et son « fils droit » à nil
            buf.Fils[j+1] := -1;
            // incrémenter le degré
            buf.degre := buf.degre + 1;

            EcrireDir( F, prec, buf );

        Sinon // si le bloc est plein, il faut allouer un nouveau ...
            i := AllocBloc( F );
            // et le chaîner avec le précédent
            buf.Fils[j] := i
            EcrireDir( F, prec, buf );
            // insérer e à la pos 1 du nouveau bloc ...
            buf.degre := 
            buf.Val[1] := e;
            buf.Fils[1] := -1;
            buf.Fils[2] := -1;
            EcrireDir( F, i, buf )
        Fsi
    Fsi // Non trouv
Fsi // Entete(F,1) = -1
Fermer(F)
Fin

```

Les B-Arbres

Ceux sont des arbres de recherche m-aires qui restent toujours équilibrés et sont donc très utilisés pour gérer des fichiers volumineux et dynamiques.

Inventés par R. Bayer et E. Mc Creight en 1972, les B-Arbres sont devenus un standard de facto à cause de leurs performances remarquables et ce, jusqu'à nos jours.

Pour simplifier la présentation, on choisira un ordre impair ($N = 2d + 1$)

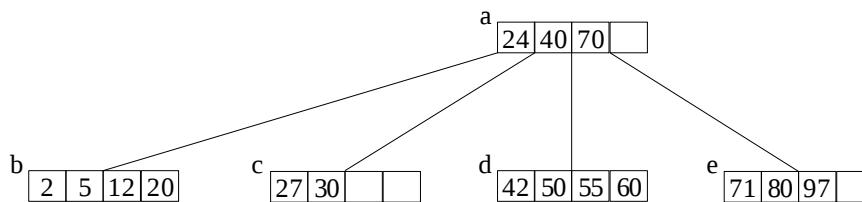
Dans un B-Arbre d'ordre N :

tous les noeuds à part la racine, sont remplis au minimum à 50% (soit d valeurs)

la racine peut contenir au minimum 1 valeur

toutes les branches ont la même longueur (arbre complètement équilibré)

La figure ci-dessous montre un B-Arbre d'ordre 5 ($d=2$)



La recherche dans un B-Arbre est similaire à la recherche dans un arbre de recherche m-aires. La différence se situe dans l'algorithme d'insertion et l'algorithme de suppression.

Pour insérer une valeur V dans un B-Arbre de racine R, on procède comme suit :

On insère V et son fils droit fd (initialement $fd = -1$), c-à-d si V sera insérée à une position j dans un noeud, alors il faudra insérer aussi « son fils droit » fd à la position j+1 (dans le tableau des fils)

1- rechercher V, pour vérifier qu'elle n'existe pas et trouver le dernier noeud feuille visité (P)

2- si P n'est pas plein, on insère (V,fd) dans P par décalages internes et on s'arrête.

3- si P est déjà plein, alors il va « éclater » en deux (P et Q, un nouveau noeud alloué) :

a) former la « séquence ordonnée » des valeurs de P + la valeur V

b) affecter les d premières valeurs avec les d+1 premiers fils au noeud P

c) affecter les d dernières valeurs avec les d+1 derniers fils au nouveau noeud Q

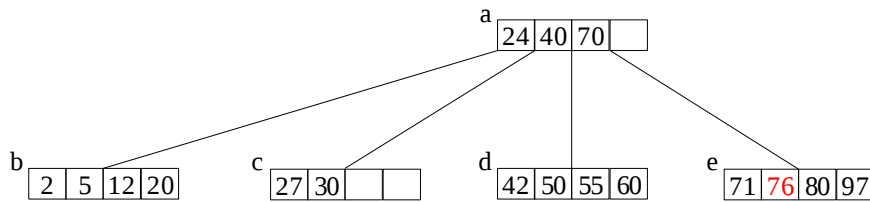
d) insérer la valeur du milieu (celle qui se trouve à la position d+1 dans la séquence ordonnée) dans le noeud parent de P. Le fils droit de cette valeur sera le noeud Q – Aller à 2

Par exemple, l'insertion de 76 dans le B-Arbre de la figure précédente, se déroule comme suit:

1- recherche de 76 → trouv = FAUX, le dernier visité = e (c'est une feuille) et la position où doit être insérée 76 est 2

2- comme le noeud e n'est pas plein, on insère alors 76 avec son « fils droit » -1 respectivement aux positions 2 dans le tableau Val et 3 dans le tableau Fils du noeud e

On obtient alors le B-Arbre ci-dessous :



Si on insère maintenant la valeur 57, c'est le noeud d qui sera visité en dernier par la recherche.

La position de 57 dans d devrait alors être 4 (et son fd à la position 5)

Comme le noeud d est déjà plein, il y aura alors un « éclatement » (cela veut dire, allocation d'un nouveau noeud, f par exemple)

a) formation de la « séquence ordonnée »:

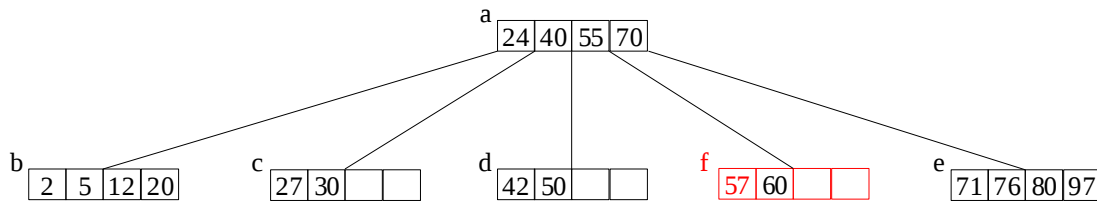
Val : 42, 50, 55, 57, 60

Fils: -1, -1, -1, -1, -1, -1

b) les d premières valeurs (42, 50) et les d+1 premiers fils (-1, -1, -1) seront affectés au noeud e
les d dernières valeurs (57, 60) avec les d+1 derniers fils (-1, -1, -1) seront affectés au noeud f
la valeur du milieu (55) avec comme fils-droit (f) seront insérés dans le noeud parent (a)

comme le noeud a n'est pas plein, l'insertion de (55, f) peut se faire par décalages internes

On obtient alors le B-Arbre suivant:



Si on continue l'insertion de 7, le noeud b va « éclater » (donc allocation d'un nouveau noeud g)

la séquence ordonnée des valeurs de b + la valeur 7 est formée:

Val : 2 , 5 , 7 , 12 , 20

Fils: -1, -1, -1, -1 , -1 , -1

Les valeurs (2 et 5) avec les 3 premiers fils restent dans b. Les valeurs (12 et 20) avec les 3 derniers fils seront affectés au nouveau noeud g. La valeur du milieu (7) avec comme fils-droit le noeud g seront insérés dans le pere (a).

Comme le noeud a est lui aussi plein, il va alors « éclater » à son tour. Un nouveau noeud h est alors alloué et la séquence ordonnée est formée :

Val : 7 , 24 , 40, 55, 70

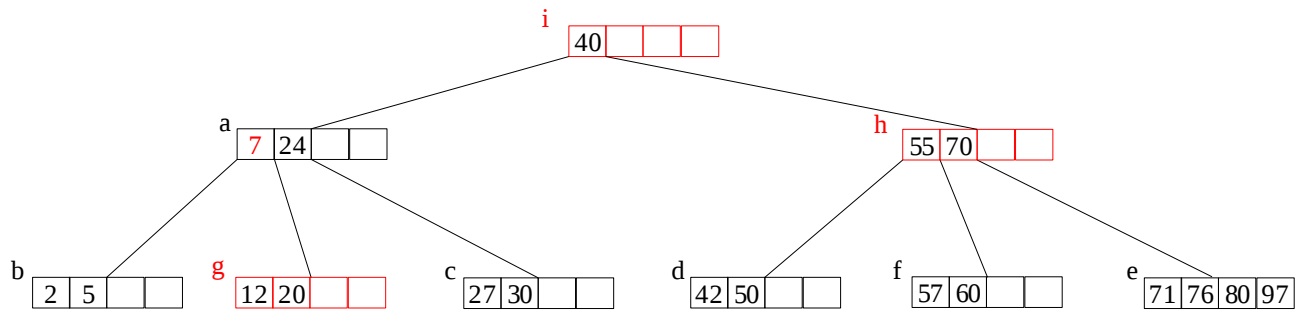
Fils: b , g , c , d , f , e

Les d premières valeurs (7 et 24) avec les 3 premiers fils (b, g et c) restent dans a. Les d dernières valeurs (55 et 70) avec les 3 derniers fils (d, f et e) seront affectés au nouveau noeud h. La valeur du milieu (40) avec comme fils-droit le noeud h seront insérés dans le père du noeud a.

Comme le noeud a était la racine de l'arbre (il n'a donc pas de père), une nouvelle racine (i) est alors allouée contenant uniquement la valeur 40 avec comme fils-gauche (Fils[1]), l'ancienne racine (a) et

comme fils-droit (Fils[2]) le noeud h.

La figure suivante, montre l'arbre obtenu :



Quand la racine d'un B-Arbre éclate (à cause d'une insertion), la profondeur de l'arbre augmente d'un niveau.