INSIA
Bases de données
SIGL 3
Optimisation – 2 : Indexation

Bertrand LIAUDET

SOMMAIRE

Première édition : octobre 2007

Deuxième édition : septembre 2009

INDEXATION

1. Problématique

Le problème de l'indexation est celui de l'accès aux données : comment rechercher rapidement à une donnée dans la BD (travail du SGBD) et dans un disque dur en général (travail du SE).

2. La recherche d'un élément : approche intuitive

On a des éléments rangés dans un tableau. Comment chercher un élément dans ce tableau ?

Il existe plusieurs méthodes associées à une organisation particulière des données

Avec un tableau quelconque

Méthode

On parcourt tous les éléments du tableau jusqu'à ce qu'on trouve celui cherché.

Complexité

O(n/2)

Inconvénients

C'est long et la durée est aléatoire (fonction de la place de l'élément cherché).

Avec un tableau trié

Méthode

On utilise la méthode de la recherche dichotomique.

Complexité

O(log2(n)

Avantages

On trouve l'élément en Log2(n) tests au maximum (par exemple, avec 1 000 000 éléments, soit environ 2 puissance 20 éléments, on trouve l'élément cherché en maximum 20 tests).

Inconvénients

Le maintient du tri en cas d'ajout ou de suppression d'un élément. On est obligé de déplacer tous les éléments en dessous de celui qu'on ajoute ou qu'on supprime. C'est d'autant plus long que les éléments du tableau sont volumineux.

Avec un tableau trié de pointeurs

Méthode

Recherche dichotomique avec un tableau de pointeurs

Complexité

O(log2(n)

Avantages

Ceux de la recherche dichotomique.

Inconvénients

Ceux de la recherche dichotomique.

Toutefois, par rapport à un tableau d'éléments, on a seulement une série de pointeurs à déplacer.

Avec une liste chaînée triée

Méthode

On parcourt tous les éléments de la chaîne jusqu'à ce qu'on trouve celui cherché.

Complexité

O(n/2)

Avantages

On peut insérer facilement un élément

Inconvénients

C'est long et la durée est aléatoire (fonction de la place de l'élément cherché).

Avec un arbre binaire

Méthode

Recherche dichotomique avec un arbre binaire.

Complexité

O(log2(n)

Avantages

On peut insérer facilement un élément

Si l'arbre est équilibré, le nombre de tests est celui d'une recherche dichotomique.

Inconvénients

Le maintien de l'équilibre de l'arbre est complexe et peut être coûteux.

Bilan général de la recherche : séquentielle et dichotomique

On vient de voir qu'il y a deux grandes méthodes de recherche :

- la recherche séquentielle sur des données quelconques.
- la recherche dichotomique sur des données triées.

La recherche dichotomique est la méthode de base pour un accès aux données efficace. L'utilisation d'arbre binaire plutôt que de tableau est la base pour des insertions et suppression de données efficaces.

3. Indexation et BD relationnelle

Présentation

Dans les BD relationnelles, l'indexation est une technique qui va permettre d'accélérer les données en permettant de faire des recherches dichotomiques.

On peut se représenter un index comme étant un tableau à deux attributs : le premier est l'attribut indexé. Le second est l'adresse du tuple correspondant dans la BD. Un index est un tableau trié selon l'attribut indexé.

Exemple

Soit le table des employés :

Adresse	NE	Nom	Job	DateEmb	Salaire	Comm	#ND	*NEchef
Ad1	7839	KING	PRESIDENT	17/11/81	5000	NULL	10	NULL
Ad2	7698	BLAKE	MANAGER	01/05/81	2850	NULL	30	7839
Ad3	7782	CLARK	MANAGER	09/06/81	2450	NULL	10	7839
Ad4	7566	JONES	MANAGER	02/04/81	2975	NULL	20	7839
Ad5	7654	MARTIN	SALESMAN	28/09/81	1250	1400	30	7698
Ad6	7499	ALLEN	SALESMAN	20/02/81	1600	300	30	7698
Ad7	7844	TURNER	SALESMAN	08/09/81	1500	0	30	7698
Ad8	7900	JAMES	CLERK	03/12/81	950	NULL	30	7698
Ad9	7521	WARD	SALESMAN	22/02/81	1250	500	30	7698
Ad10	7902	FORD	ANALYST	03/12/81	3000	NULL	20	7566
Ad11	7369	SMITH	CLERK	17/12/80	800	NULL	20	7902
Ad12	7788	SCOTT	ANALYST	09/12/82	3000	NULL	20	7566
Ad13	7876	ADAMS	CLERK	12/01/83	1100	NULL	20	7788
Ad14	7934	MILLER	CLERK	23/01/82	1300	NULL	10	7782

Indexer la clé primaire et le job consiste à créer les deux tables d'index suivantes :

Adresse	NE
Ad11	7369
Ad6	7499
Ad9	7521
Ad4	7566
Ad5	7654
Ad2	7698
Ad3	7782
Ad12	7788
Ad1	7839
Ad7	7844
Ad13	7876
Ad8	7900
Ad10	7902
Ad14	7934

Adresse	Job
Ad10	ANALYST
Ad12	ANALYST
Ad8	CLERK
Ad11	CLERK
Ad13	CLERK
Ad14	CLERK
Ad2	MANAGER
Ad3	MANAGER
Ad4	MANAGER
Ad1	PRESIDENT
Ad5	SALESMAN
Ad6	SALESMAN
Ad7	SALESMAN
Ad9	SALESMAN

Dans les BD, l'index est toujours trié. Ainsi, on pourra faire une recherche dichotomique à partir de NE ou à partir du job et accéder aux tuples.

Index primaires et index secondaires : unicité ou multiplicité du résultat

Index primaire ou unique

Les index primaires sont ceux qui s'appliquent à la clé primaire ou aux clés secondaires d'une table.

La sélectivité est alors la plus petite possible : 1 / nombre de tuples de la table.

La recherche conduit à un élément et un seul. On veut par exemple l'employé n° 7654. La sélectivité est de 100/14 %

Index secondaire ou non unique

Les index secondaires sont ceux qui s'appliquent à des attributs qui ne sont ni clés primaires ni clés secondaires.

Dans ce cas la sélectivité est supérieure à 1 / nombre de tuples de la table.

La rechercher conduit à plusieurs éléments. On recherche par exemple tous les SALESMAN et on obtient 4 employés. La sélectivité est de 100*4 / 14 % soit 28,5%.

Statisitiquement, cette sélectivité ne devrait pas beaucoup bouger : elle correspond au taux de SALESMAN nécessaire au fonctionnement de l'entreprise.

Utilité d'un index : sélectivité < 30 %

Pour qu'une recherche indexée soit efficace, il faut que la sélectivité reste faible.

En pratique, on considère que pour une sélectivité > 20 ou 30%, l'index est inefficace.

Ceci vient du fait que les accès disques sont l'élément le plus pénalisant en terme de coût. Or, l'accès direct à un grand nombre N d'éléments conduit à au même nombre N d'accès disque, tandis qu'en accès séquentiel, le nombre d'accès disque est limité du fait de la lecture du disque par page et de la présence de plusieurs tuples sur la même page.

Index plaçant et non plaçant

Un index plaçant est un index dont le tri correspond au tri physique sur le fichier.

En général, la clé primaire est un index plaçant. D'où l'intérêt de ne pas gérer soi-même la clé primaire et d'en faire un attribut auto-incrémenté.

Les index plaçant sont en général organisé avec des arbres B+ : l'arbre permet de gérer les index et les tuples de la table.

Index bitmap

L'index bitmap s'applique aux attributs ayant un nombre de valeurs limité.

Il consiste à créer une table contenant comme attributs toutes les valeurs possibles de l'attribut de départ qu'on veut indexer. Dans cet index, les valeurs possibles seront 0 ou 1 selon les attributs de l'index correspondent ou pas à la valeur de l'attribut de départ.

Un index bitmap est donc une matrice creuse (constituée de 0 ou de 1) encore appelée « bitmap ».

Il n'y a pas de tri, mais seulement le remplacement d'une valeur de type chaîne de caractères par une valeur de type booléen.

Le tri n'est pas nécessaire car l'index (0 ou 1) donne directement la valeur sur un bit.

Exemple : index bitmap sur le JOB

Adresse	PRESIDENT	MANAGER	SALESMAN	CLERK	ANALYST
Ad1	1				
Ad2		1			
Ad3		1			
Ad4		1			
Ad5			1		
Ad6			1		
Ad7			1		
Ad8				1	
Ad9			1		
Ad10					1
Ad11				1	
Ad12					1
Ad13				1	
Ad14				1	

Index MySQL : INDEX = KEY

Les 4 types d'index MySQL

Index de clé primaire (unique) : PRIMARY KEY

Index unique (non clé primaire) : UNIQUE

Index non unique : INDEX ou KEY (pour les clés étrangères et les attributs de recherche).

Index de texte : FULLTEXT (on utilise ensuite MATCH et AGAINTS pour les recherche).

Création des index

CREATE TABLE

ALTER TABLE

CREATE INDEX

La commande SHOW INDEX

```
mysql> show index from livres\G
*************************** 1. row ***************************
       Table: livres
  Non_unique: 0
    Key_name: PRIMARY
Seq_in_index: 1
 Column_name: NL
   Collation: A
 Cardinality: 30
    Sub_part: NULL
      Packed: NULL
        Null:
  Index_type: BTREE
     Comment:
*************************** 2. row ***************************
       Table: livres
  Non_unique: 1
    Key_name: NO
Seq_in_index: 1
 Column_name: NO
   Collation: A
 Cardinality: 30
    Sub_part: NULL
      Packed: NULL
        Null:
  Index_type: BTREE
     Comment:
2 rows in set (0.01 sec)
```

Non unique : precise si l'index est unique ou pas.

Key name : précise le nom de l'index

Seq in index : précise la position de l'attribut dans l'index dans le cas d'index concaténé.

Cardinality : précise le nombre de tuples de la tables.

4. Indexation et gestion de fichiers

Définition générale

En informatique, la notion d'index concerne d'abord les fichiers et les enregistrements (fiches enregistrées) dans les fichiers.

Chaque enregistrement possède une clé primaire.

Un index est un tableau qui associe à chaque fiche du fichier son adresse relative dans le fichier.

Plus précisément, l'index associe la valeur de la clé de chaque fiche à l'adresse relative de la fiche dans le fichier.

L'index d'un fichier F peut être rangé dans un nouveau fichier ou dans le fichier F, à la fin par exemple (c'est le cas le plus courant).

Exemple de fichier avec un index à la fin :

	F7			F4		F1				F3	F7	0	F4	3	F1	5	F3	9
Adresse :	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17

Le fichier contient 4 fiches de tailles variables. Les clés de ces fiches sont : F7, F4, F1, F3.

La clé de chaque fiche est en général placée au début de la fiche.

A noter qu'en principe l'index n'est pas forcément trié.

Caractéristiques générales de l'indexation

Index trié

Un index peut être trié ou pas. En général, on utilise plutôt les index triés.

Fichier trié

Un index peut s'appliquer à un fichier trié ou pas.

Index dense et non dense

La densité d'un index c'est le rapport entre le nombre d'éléments indexés et le nombre total d'éléments (autrement dit, dans le cas d'une BDR, entre le nombre de lignes de la table d'index et le nombre de lignes de la table d'origine).

La densité d'un index varie entre 0 et 1.

Quand la densité vaut 1, tous les éléments du fichier (ou de la table) sont indexés. On parle d'index dense.

Les 8 différents cas possibles

Cas	Tri du fichier	Densité	Tri de l'index	Applications
1	Non trié	Non dense	Non trié	Impossible
2	Non trié	Non dense	Trié	Impossible
3	Non trié	Dense	Non trié	∅ : évolue en 4
4	**Non trié**	**Dense**	**Trié**	**IS3**
5	Trié	Non dense	Non trié	∅ : évolue en 6
6	**Trié**	**Non dense**	**Trié**	**ISAM, VSAM, UFAS**
7	Trié	Dense	Non trié	∅ : évolue en 6
8	Trié	Dense	Trié	∅ : lourd : évolue en 6

ISAM : Indexed Sequential Acces Method. Utilisé par DOS et MVS-IBM.
VSAM : Virtual Sequential Acces Method. C'est un ISAM amélioré qui utilise des arbres B+.
UFAS : inspire de VSAM, c'est une méthode de BULL.
IS3 : Indexed Serie 3. Utilisé sur les AS400 d'IBM. C'est un ISAM dérivé avec arbres B+.
Ce sont des méthodes de gestion de fichier de certains SE.

5. Organisation des fichiers : arbre, arbre B, arbre B+

Problématique

Les techniques ISAM, VSAM, UFAS utilisent des fichiers triés.

Toutes les techniques utilisent des index triés.

Le problème d'une série triée, c'est l'ajout d'un élément dans le fichier : il faut insérer l'élément au bon endroit. Si la gestion est séquentielle : il faudra déplacer tous ceux du dessous, ce qui est très coûteux.

Le problème se pose de la même façon pour les index si on a des index très grand.

La solution consiste à utiliser une structure d'arbre.

Arbre binaire

Définition

Principes généraux

- Un arbre binaire est une organisation composée de nœuds reliés entre eux par des branches.
- Un nœud peut être parent et/ou enfant.
- Les branches sont orientées : elles vont du nœud parent au nœud enfant.
- Un nœud a au maximum un parent.
- L'unique nœud de l'arbre qui n'a pas de parent est appelé « racine ».
- Les nœuds de l'arbre qui n'ont pas d'enfants sont appelés « feuille ».

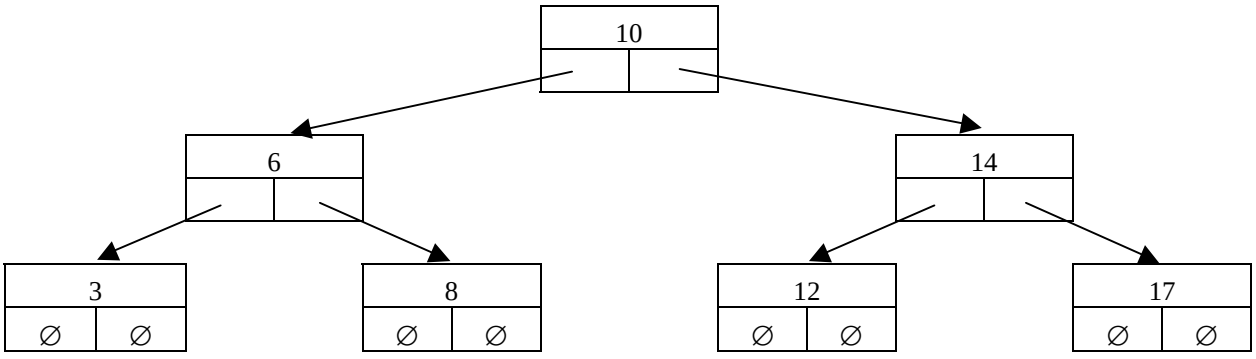
Principes spécifiques

- Un nœud a au maximum deux enfants.
- Chaque nœud porte l'information d'un enregistrement et un seul (une fiche ou un tuple par exemple).

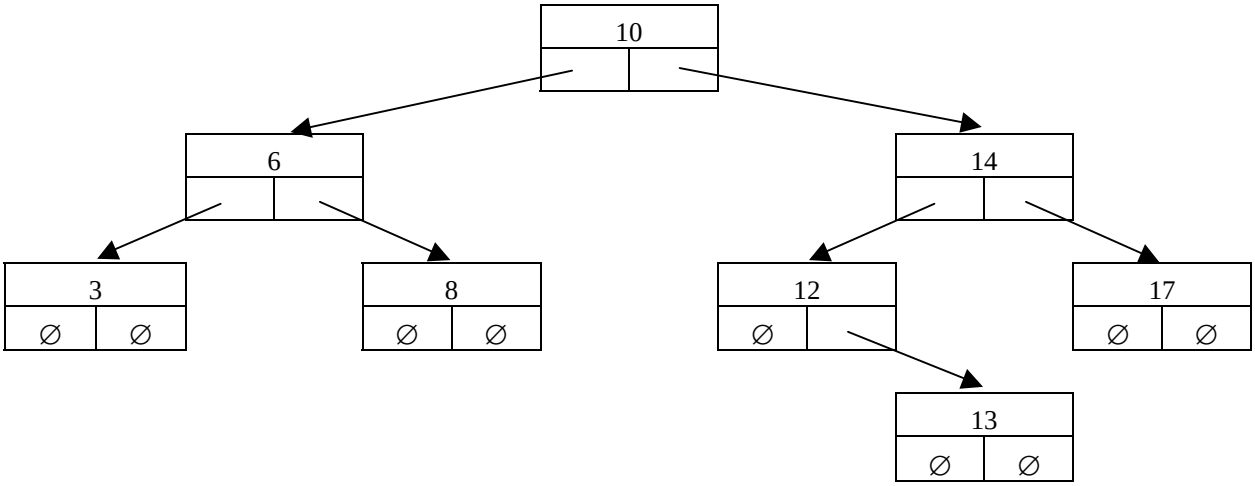
Usage

Un arbre binaire permet de faire des recherches dichotomiques et d'insérer assez facilement de nouvelles données.

Exemple d'arbre binaire



Si on veut ajouter 13 dans cet arbre, on le place sous le 12, à droite :

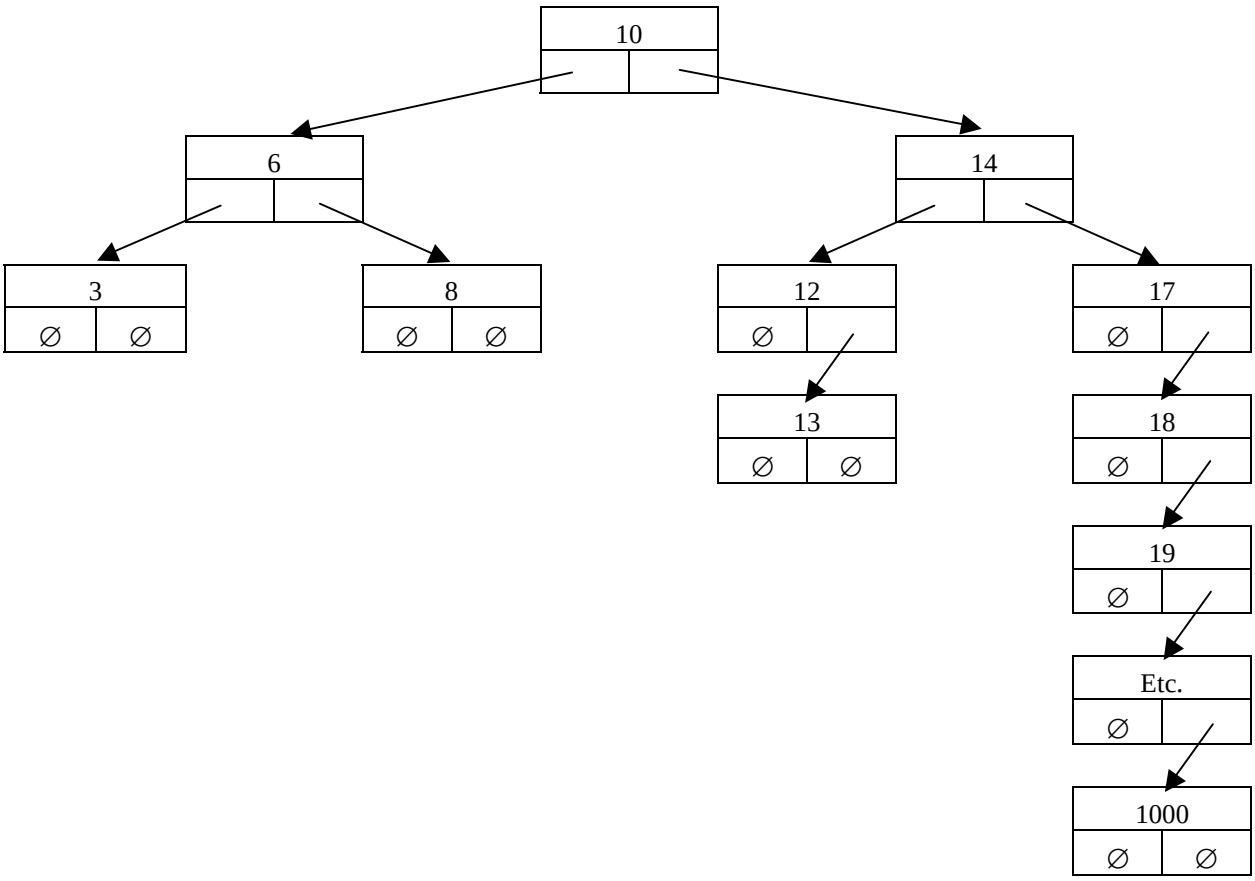


Contenu de chaque nœud

Un nœud contient la valeur et l'adresse de ses deux fils.

Déséquilibre de l'arbre binaire

Dans notre exemple, si on ajoute successivement 18, 19, 20, 21, etc. jusqu'à 1000, on va obtenir un arbre complètement déséquilibré avec une grande série mono-branche.



Autrement dit, l'arbre binaire se transforme en liste chaînée. De ce fait, la recherche de la valeur 1000 dans un tel arbre se fera en presque 1000 tests.

Solution du problème du déséquilibrage

On peut soit ajouter en maintenant l'équilibre : l'algorithme est complexe et surtout potentiellement coûteux.

On peut aussi équilibrer occasionnellement indépendamment des ajouts, par exemple la nuit.

La solution la plus adaptée dépend des usages de la BD.

Arbre B (arbre balancé, B-tree)

Définition

Principes généraux

- Un arbre B (ou arbre balancé ou B tree) est une organisation composée de nœuds reliés entre eux par des branches.
- Un nœud peut être parent et/ou enfant.
- Les branches sont orientées : elles vont du nœud parent au nœud enfant.
- Un nœud a au maximum un parent.
- L'unique nœud de l'arbre qui n'a pas de parent est appelé « racine ».
- Les nœuds de l'arbre qui n'ont pas d'enfants sont appelés « feuille ».

Principes spécifiques

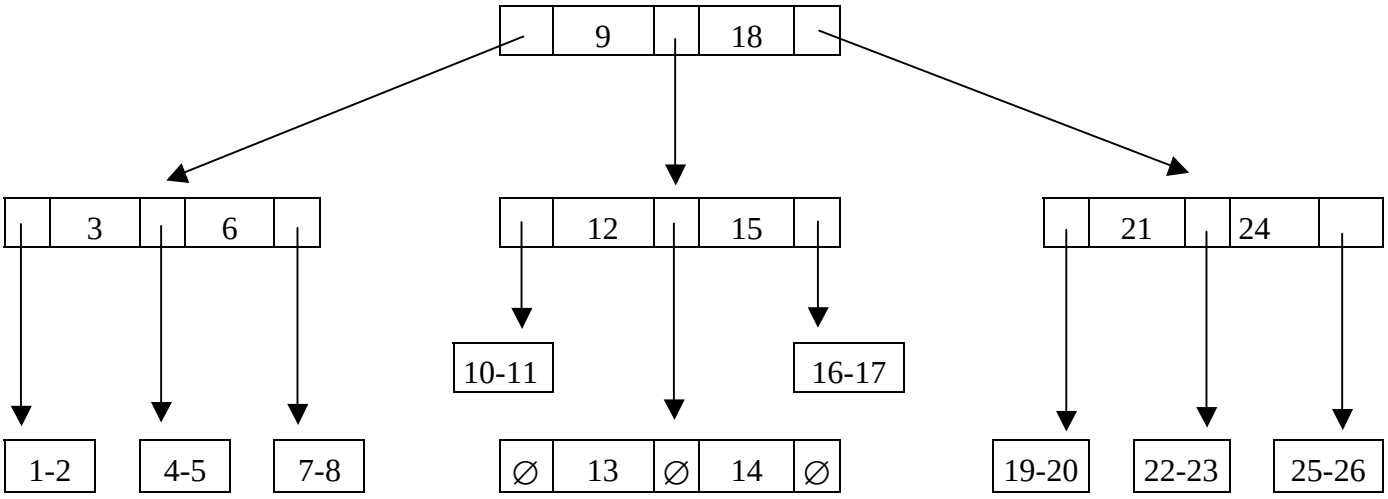
- Un arbre B a un numéro d'ordre : N.
- Les nœuds ont au maximum 2 * N + 1 enfants.
- Les nœuds non-feuilles et non racines ont au minimum N + 1 enfants.
- Chaque nœud non-feuille porte NF-1 enregistrements, avec NF = nombre de fils du nœud.
- Chaque nœud feuille porte 2* N enregistrements au maximum (2*N fiches ou 2*N tuples par exemple).

Bilan

Ordre	**Nb fils min**	**Nb fils max**	**Nb info max/noeud**
N	N+1 (sauf racine et feuilles)	2*N + 1 = NFM	Nb Fils –1 = NFM - 1
1	2	3	2
2	3	5	4
3	4	7	6
…			
10	11	21	20

Exemple

Arbre B d'ordre 1 parfaitement équilibré :



Nombre maximum d'éléments dans un arbre B

Le nombre maximum d'éléments dans un arbre B est fonction de l'ordre N de l'arbre et de la hauteur de l'arbre H. On considère qu'un arbre avec seulement un nœud racine est de hauteur 0.

Le nombre d'éléments maximum est donné par la formule :

$$\mathbf{2*N * Somme(i :0 à H) (2*N +1)^i}$$

Dans le cas d'un arbre d'ordre 1 (3 fils max et 2 éléments max par nœud) et de hauteur 2, on obtient :

$$2 * (3^0 + 3^1 + 3^2) = 2 * (1 + 3 + 9) = 26$$

Dans le cas d'un arbre d'ordre 2 (5 fils max et 4 éléments max par nœud) et de hauteur 5, on obtient :

$$4 * (5^0 + 5^1 + 5^2 + 5^3 + 5^4) = 4 * (1 + 5 + 25 + 625 + 3125) = 15\ 624$$

Dans le cas d'un arbre d'ordre 5 (11 fils max et 10 éléments max par nœud) et de hauteur 5, on obtient :

$$10 * (11^0 + 11^1 + 11^2 + 11^3 + 11^4) = 1\ 771\ 560$$

Le problème de l'ajout et de la suppression dans les arbres B

L'ajout et la suppression dans les arbres B posent le même problème que pour les arbres binaires : celui du maintien de l'équilibre de l'arbre.

Choix du numéro d'ordre d'un arbre

Le nœud d'un arbre va correspondre à la taille d'un bloc sur disque (par exemple 512 octets).

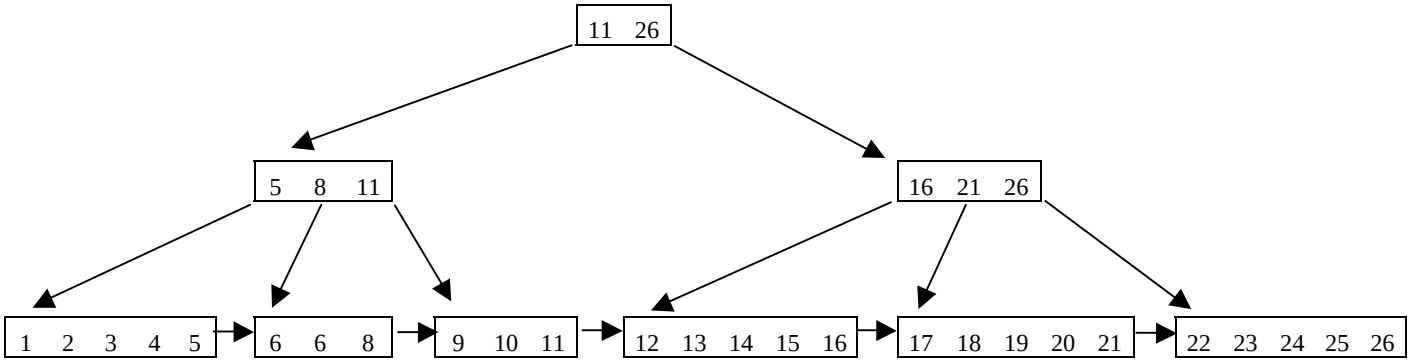
Ainsi, on limitera les accès disques au maximum.

6. Arbre B+

L'utilisation d'arbres B peut s'avérer coûteuse pour un traitement séquentiel du fait du parcours de l'arbre.

L'arbre B+ permet de pallier à cette difficulté en répétant les informations figurant dans les nœuds parents au niveau des nœuds enfants et en liant les nœuds feuilles entre eux.

De ce fait, on retrouve une structure séquentielle en parcourant directement tous les nœuds feuilles.



En pratique, on utilise les arbres B+ :

- Soit pour implémenter seulement les index. Cela correspond à la méthode IS3 (index dense trié, fichier non trié).
- Soit pour implémenter les index et les fichiers. Cela correspond aux méthodes VSAM et UFAS (index non dense trié, fichier trié).

6. Organisation des index : index hiérarchisés

Problématique

Un index est un tableau trié.

A chaque ajout ou suppression d'un élément dans la table d'origine, l'index doit être mis à jour.

Le problème revient alors à celui de l'ajout et de la suppression dans un tableau trié : à chaque opération, il faut déplacer les éléments du dessous. Si l'index est très grand, l'opération sera coûteuse.

On va donc choisir d'organiser l'index non plus dans un tableau mais dans un arbre.

Notion d'index hiérarchisé

Un index organisé dans un arbre est appelé index hiérarchisé ou index d'index.

HACHAGE

1. Présentation

Les méthodes d'accès par hachage sont basées sur l'utilisation d'une **fonction de calcul** (la fonction de hachage) **qui, appliquée à la clé, détermine l'adresse relative d'un paquet** (bucket en anglais) ou se trouve l'article. (c. Gardarin , Bases de données, p. 73).

On distingue hachage statique et hachage dynamique.

Le hachage statique concerne les fichiers de taille fixe.

Le hachage dynamique concerne les fichiers de taille variable.

2. Le hachage statique

Principe

Du fait de la taille constante du fichier, la méthode est simple.

Le fichier est divisé en P paquets de taille T fixe. L'adresse relative de chaque paquet est donnée par la formule : AR = NB * T, avec NB : n° du bloc en question.

Ensuite, dans le paquet, les articles sont rangés dans leur ordre d'arrivée. En tête de chaque paquet, on trouve l'adresse du premier octet libre du paquet.

Dans le paquet, l'accès est séquentiel ou direct si on ajoute un autre système d'accès dans le paquet.

Fonction de hachage

Il faut calculer le numéro du paquet à partir de la clé d'un article.

La technique la plus répandue est celle du modulo. La clé doit être un entier. La technique consiste alors à prendre pour numéro de paquet le reste de la division de la clé par le nombre de paquets.

Le choix de la fonction de hachage est primordial pour assurer une équi-répartition des articles dans les N paquets. Ce choix doit être guidé par la distribution, rarement uniforme, des clés dans le fichier.

Problème des collisions (ou débordement)

Il y a collision ou débordement quand un paquet est plein et qu'on veut y mettre un nouvel article.

Plusieurs solutions techniques existent.

Le plus souvent, ces techniques utilisent un paquet de débordement, éventuellement avec une deuxième fonction de hachage pour accéder à l'article dans ce paquet.

Dans tous les cas, la gestion du débordement dégrade les performances et complique les algorithmes.

Conclusion

Avantages

Méthode simple et performante. Une lecture d'article s'effectue en une entrée-sortie.

Inconvénients

La taille du fichier doit être fixée a priori

Le débordement dégrade les performances en augmentant le nombre d'entrée-sortie.

3. Le hachage dynamique

Principe

L'objectif des organisations par hachage dynamique est de minimiser le coût des collisions en substituant aux techniques de débordement un accroissement dynamique de la taille du fichier.

Du fait de la taille variable du fichier, la méthode est plus complexe.

Le principe est d'utiliser une fonction de hachage de la clé qui génère une chaîne de N bits où N est grand (par exemple 32).

Lors de la première implantation du fichier haché, seuls les M premiers bits sont utilisés (avec M petit devant N). pour calculer le numéro du paquet.

Quand un premier paquet déborde, une nouvelle région est allouée au fichier et les articles du paquet plein sont répartis entre le paquet plein et le nouveau paquet.

Les bits M+1, M+2, etc. de la fonction de hachage sont successivement utilisés.

TP

1. BD buveurs 1

Charger la BD : BuveursMyISAM.txt

Pour chaque table :

Vérifier le code : show create table

Observer les index : show index

Répondre à requête : tous les buveurs ayant bu un Bordeaux de degré supérieur ou égal à 13

Donnez une solution avec une requête unique.

Donner une solution avec des vues correspondant à un arbre linéaire optimisé.

Donner une deuxième solution avec des vues correspondant à un autre arbre linéaire optimisé.

2. BD buveurs 2

Charger la BD : BuveursMyISAM avec index.txt

Pour chaque table :

Vérifier le code : show create table

Observer les index : show index