Organisation des fichiers

Dr. ZAMOUCHE Djamila

Université A. MIRA - Bejaia

Faculté des Science Exactes

Département d'Informatique

Email : djamila.zamouche@univ-bejaia.dz

2024/2025

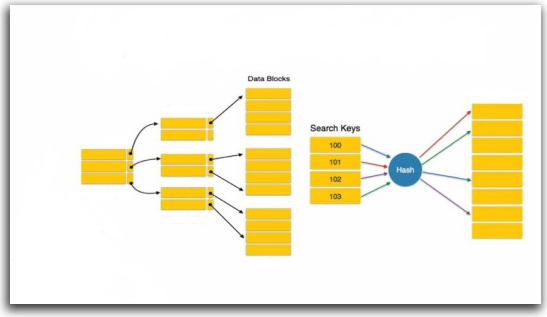


Table des matières

Objectifs

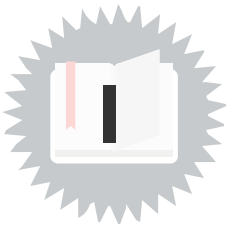
Les objectifs du cours "*Organisation des fichiers*" sont comme suit :

- Comprendre les structures de base pour l'organisation des fichiers ;
- Maîtriser les techniques d'indexation comme les index primaires et secondaires pour optimiser les recherches ;
- Maîtriser des structures de données avancées comme les arbres B et le hachage pour les fichiers volumineux ;
- Savoir sélectionner une méthode adaptée aux données et aux opérations nécessaires.

Introduction

Dans ce chapitre, nous aborderons les différentes façons dont les fichiers peuvent être organisés dans la mémoire secondaire. Ensuite, nous découvrirons les différentes stratégies d'indexation et les méthodes de hachage qui permettent un accès efficace et plus rapide à ces fichiers.

Notion d'organisation



La façon dont les enregistrements d'un fichier sont rangés sur le support.

Méthodes d'accès

C'est la façon, en fonction de l'organisation, dont les enregistrements sont accédés sur le support.

STRUCTURES SIMPLES

1. Organisation contiguë

Le fichier est un ensemble de M blocs contigus sur le disque. C'est par conséquent un **tableau** (le fichier est vu comme un tableau d'enregistrements). Le fichier peut être ordonné ou pas.

- **Fichier non ordonné (organisation Tas) :** les enregistrements sont rangés au fur et à mesure, sans ordre particulier. L'accès est alors séquentiel.
    - **Avantage :** Simplicité de gestion (insertion / suppression).
    - **Inconvénient :** Recherche non performante (séquentielle).

  Cet organisation convient pour le fichier de petite taille.
- **Fichier ordonné (organisation trié) :** les enregistrements sont rangés par ordre croissant (ou décroissant) d'une clé. L'accès est alors soit dichotomique soit séquentiel.
    - **Avantage :** Recherche plus rapide (dichotomique).
    - **Inconvénient :** Mise à jour plus contraignante (insertion à la bonne place).

2. Organisation chaînée

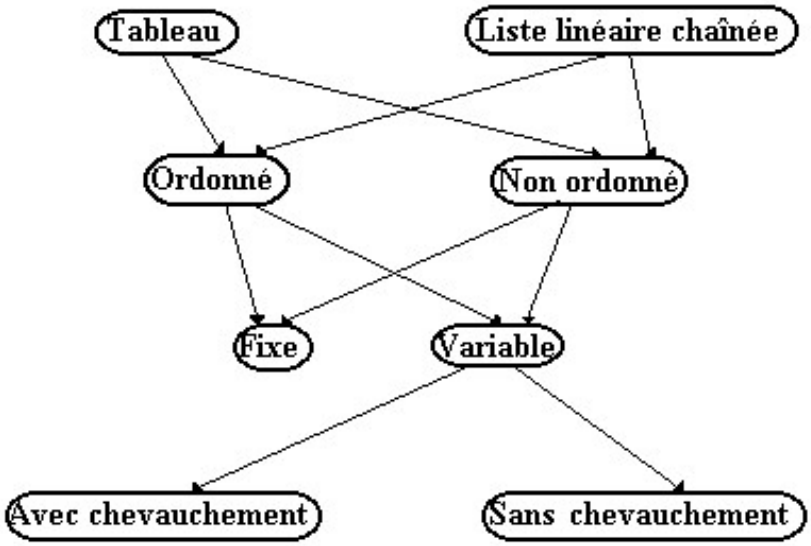
Le fichier peut être vu comme une **liste linéaire chaînée (LLC)**, c'est à dire constitué de M blocs non contigus sur le disque. Le fichier peut être ordonné ou pas.

- **Fichier non ordonné :** les ajouts d'articles se font à la fin du fichier, sans ordre particulier.
- **Fichier ordonné :** les ajouts d'articles causent des insertions de nouveaux blocs (allocation dynamique).

Dans les deux cas, l’accès est séquentiel.

3. Classification des structures simples

En combinant les différentes organisations possibles d'un fichier, on peut construire une multitude de structures simples de fichiers que l'on peut résumer dans le graphe suivant :



En suivant les flèches de haut en bas, on peut construire une douzaine de structures de fichiers. On peut citer :

1. Tableau, ordonné, fixe ;
2. Liste linéaire chaînée, non ordonnée, variable, avec chevauchement ;
3. Liste linéaire chaînée, ordonnée, fixe ;

   etc.

- *Dans le premier cas :* le fichier est vu comme un tableau. Il est ordonnée, ce qui veut dire que pour deux blocs b1 et b2 si b1 précède b2, alors tous les articles dans b1 sont inférieurs (selon la clé) aux articles du b2. De plus, à l'intérieur d'un bloc, les articles sont de longueur fixe.
- *Dans le deuxième cas :* le fichier est vu comme une liste linéaire chaînée de blocs. Les articles ne sont pas ordonnés, et sont de longueur variable. Un article peut commencer dans un bloc et finir dans le bloc suivant.

Remarque

Ces techniques sont très efficaces pour le traitement séquentiel, c'est à dire quand le seul traitement consiste à appliquer une opération pour chaque élément du fichier.

Cependant, l'inconvénient de l'organisation simple est la lenteur d'accès aux enregistrements. Ce type d'organisation est adapté aux fichiers de taille raisonnable.

4. Exercice

Considérant les douze cas du graphe donnant les différentes organisations, pour chaque cas, développer les algorithmes de

1. Recherche,
2. Insertion,
3. Suppression.

Rappel sur les opérations de base sur les fichiers

1. Déclaration

Pour déclarer une variable de type fichier, on utilise l'instruction suivante : **Var nom_fichier : fichier de type ;**

Exemple

Var F : fichier de entier ; // un fichier qui contient des entiers.

Type Etudiant = **enregistrement**

Nom, prenom : chaine [25] ;

Moy : reel ;

Fin ;

Var F : fichier de Etudiant ;

2. Association fichier physique et fichier logique

Une variable de type fichier dans un programme est dite *fichier logique*, elle doit être associée à un fichier physique pour pouvoir y sauvegarder ou lire des données. Le fichier physique est désigné par son nom ou bien son chemin sur le support de stockage. L'association d'un fichier logique à un fichier physique se fait comme suit : **Associer (fichier logique, fichier physique) ;**

Exemple

Associer (F, 'Etudiant.txt') ;

Associer (F, 'C\:Programmes\Etudiant.dat') ;

3. Ouverture

Avant d'accéder à un fichier pour effectuer une opération de lecture ou de sauvegarde des données, il faut l'ouvrir pour le préparer à l'utilisation selon le type d'opérations à effectuer, et ceci en utilisant la fonction : **Ouvrir (nom_fichier, mode) ;**

Le mode d'ouverture indique le type d'opération à effectuer sur la variable de type fichier. Les modes d'ouverture les plus utilisés sont : lecture, écriture et Ajout.

- Lecture : permet d'ouvrir le fichier pour lire les données ;
- Ecriture : permet d'ouvrir le fichier pour y sauvegarder des données. Si le fichier physique n'existe pas, il sera créé. Si le fichier existe déjà, son contenu sera écrasé et devient donc un fichier vide comme s'il vient d'être créé ;
- Ajout : permet d'ouvrir un fichier pour y ajouter des données tout en gardant les données existantes dans le fichier.

4. Lecture et écriture

- L'opération de lecture des enregistrements d'un fichier est effectuée en utilisant la fonction :

  Lire (nom_fichier, variable) ;

  L'instruction lire permet de lire un enregistrement du fichier et de le sauvegarder dans la variable "variable" fournie en paramètre. Le pointeur de fichier s'incrémente après chaque opération de lecture, les enregistrements sont donc lus de manière séquentielle. A l'ouverture d'un fichier en mode lecture, le pointeur de fichier est positionné sur le premier enregistrement.

- L'opération d'écriture dans un fichier est effectuée en utilisant la fonction : **Ecrire (Fichier, variable) ;**

  Cette fonction permet de sauvegarder l'enregistrement contenu dans "variable" à la fin du fichier. Les enregistrements sont écrits dans le fichier l'un à la suite de l'autre.

5. Fermeture de fichier

Dans un programme, tout fichier ouvert doit être fermé avec la fonction : **Fermer(nom_fichier)** ;

Les fonctions de lecture et d'écriture ne manipulent pas directement le fichier physique, le transfert de données se fait vers une mémoire tampon. La procédure de fermeture permet de vider la mémoire tampon et écrit les données sur le support physique.

Exercice

Écrire un algorithme qui permet de sauvegarder les informations relatives à n étudiants dans un fichier 'etudiant.dat' et afficher le non et le prénom des étudiants admis. Un étudiant est caractérisé par les informations suivantes : matricule, nom, prénom, moyenne.

MÉTHODES D'INDEX

Si l'opération de recherche est très fréquente pour une application donnée, parmi les structures simples des fichiers, le choix d'un tableau ordonné est le meilleur. Par conséquent, la méthode est limitée à des fichiers dont la taille ne peut excéder quelques milliers d'articles.

Au lieu de trier le fichier, et effectuer une recherche dichotomique, on utilise l'**index**. La **recherche dichotomique** se fait alors **entièrement en mémoire**.

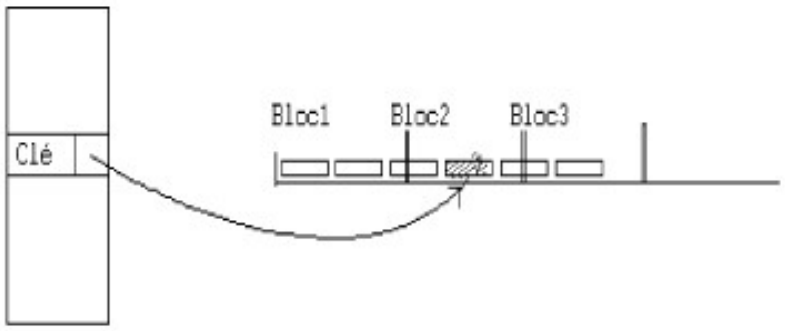
Pour ces méthode d'index, deux fichiers sont utilisés : le **fichier d'index** et le **fichier de données**.

- Le fichier de données est une structure simple **non ordonnée** (tableau ou LLC) de blocs sur le disque ;
- le fichier d'index, supposé en mémoire, est **trié** selon les clés des articles (tableau ordonné en mémoire).

1. Index primaire

L'index est un ensemble de couples (clé, adresse) rangés entièrement en mémoire centrale. Pour chaque article du fichier, une entrée dans l'index lui est associée.

Dans l'index primaire, l'index contient les adresses de chacun des enregistrements du fichier. Le fichier est un ensemble de blocs sur le disque. Le bloc contient un ensemble d'articles. Les articles peuvent être à cheval sur deux bloc logiquement consécutifs.



1.1. Opérations de base

Pour mettre au point une méthode d'index, les opérations suivantes sont nécessaire :

- Créer un fichier d'index et de données ;
- Charger le fichier d'index en mémoire ;
- Réécrire l'index après utilisation ;
- Rechercher un articles de clé donnée ;
- Insérer un article ;
- Supprimer un article ;

- Modifier un article.

a) Chargement de l'index

Le fichier index est généralement chargé en MC dans un **tableau** avant l'utilisation. Pour éviter les accès au support de stockage, le traitement s'effectue sur le tableau (ajout, suppression, recherche).

Type article=**enregistrement**

clé :**entier**;

info :**type_de_info**;

fin;

entree=**enregistrement**

clé :**entier**;

adresse :**entier**;

fin;

fichier=**fichier** de entree;

Écrire la procédure de chargement de l'index.

b) Recherche

Pour rechercher un article de clé donnée dans le fichier, on commence par :

1. faire une recherche dichotomique sur l'index ;
2. Si la clé est trouvée, le bloc contenant l'article est ramené en mémoire centrale afin de récupérer l'article ;
3. Si ce dernier est à cheval sur deux blocs, le deuxième bloc est ramené pour récupérer le reste de l'article.

Écrire la fonction de recherche dans l'index.

c) Insertion

Une insertion d'articles est réalisée comme suit :

1. Rechercher dans le fichier index le numéro de l'article à insérer ;
2. Si la clé n'est pas trouvée dans l'index, elle est y insérée avec décalage ;
3. L'article est inséré en fin de fichier.

Écrire la procédure d'insertion.

Remarque

Sur l'index le décalage est obligatoire, ce qui n'est pas très grave car le traitement se fait entièrement en mémoire (aucun accès disque).

S'il n'existe pas de place dans le bloc où l'article devrait être placé, il est inséré dans la zone de débordement.

Complément

La zone de débordement est un ensemble de blocs en général en bout de fichier, qui contient tous les enregistrements qui n'ont pas trouvé place dans leur blocs.

d) Suppression

- **Suppression logique :** une suppression est généralement logique en utilisant un indicateur d'effacement. Elle est donc extrêmement rapide.
- **Suppression physique :** elle consiste à éliminer physiquement les articles supprimés. Généralement, on construit un autre fichier.
- Mise à jour de l'index.

e) Sauvegarde de l'index

Si l'index à été modifié, l'index doit être sauvegarder en réécrivant comme un **nouveau fichier**.

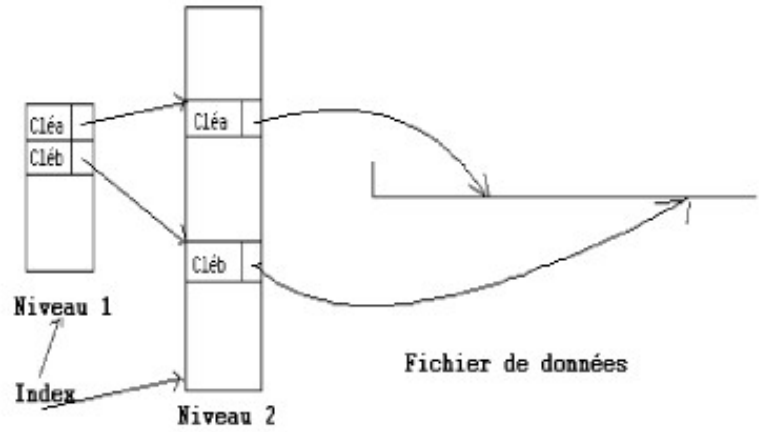
Écrire la procédure de sauvegarde.

2. Index multi-niveaux

Afin d'accélérer la recherche en mémoire centrale, on crée deux niveaux d'index :

- Dans le niveau 1, on range certaines clés,
- Dans l'index du niveau 2, on range toutes les clés.

Ici, on utilise un index de l'index. Cette solution consiste à traiter l'index comme un fichier séquentiel indexé.



2.1. Procédure pour localiser un enregistrement

Pour rechercher un article de clé donnée :

1. On effectue une recherche sur l'index du niveau 1 pour déterminer l'entrée dont la clé est <= à celle de l'enregistrement désiré ;
2. L'adresse correspondante à l'enregistrement trouvé renvoie à un bloc de l'index du niveau 2 ;
3. On parcours ce bloc jusqu'à tomber sur l'entrée dont la clé est >= à celle de l'enregistrement désiré ;
4. L'adresse correspondante renvoie au bloc du fichier de données qui contient l'enregistrement recherché.

Remarque

Il est clair que les deux recherches dichotomiques sur les deux petits vecteurs (**index du niveau 1** et **une partie du niveau 2**) sont beaucoup plus rapide qu'une seule recherche dichotomique sur un grand fichier (index du niveau 2).

2.2. Type de fichiers d'index

Il existe deux types de fichiers d'index :

1. **Index dense :** Fichier de données non ordonné, et fichier index ordonné contient toutes les clés).
2. **Index non dense :** Fichier de données ordonné, et fichier index ordonné ne contient pas toutes les clés.

Remarque

Généralement pour ces méthodes, l'index est supposé en RAM pendant l'exploitation du fichier.

2.3. Accès séquentiel indexé

Définition

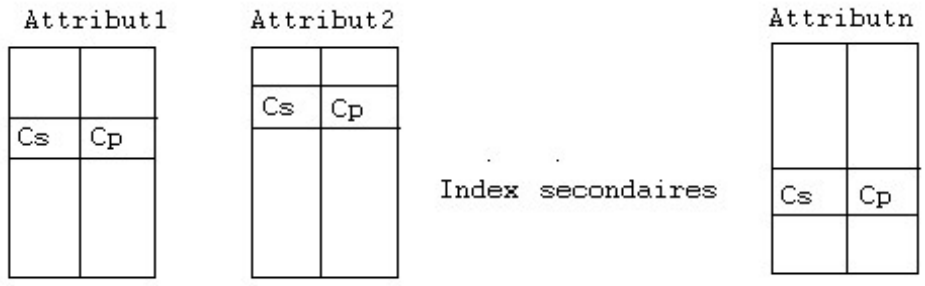
Dans l'accès séquentiel indexé, pour localiser un article, on utilise un index, et pour avoir le suivant, on ne repasse pas par l'index.

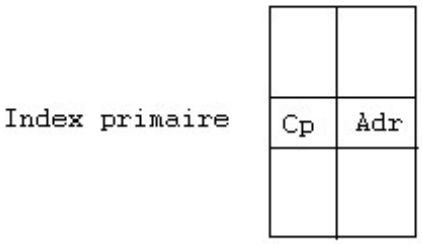
3. Index secondaire

Si on veut récupérer l'article d'un fichier qui a tel ou/et tel attribut, par exemple, si l'article d'un fichier bibliothèque possède les champs suivants : côte, titre, année, auteur, on peut vouloir faire les requêtes suivantes :

- Retrouver tous les livres parus en 2000.
- Retrouver tous les livres parus en 2000 de l'auteur X.

Une solution simple consiste à créer des **index secondaires** pour chaque attribut.





Dans la méthode d'index secondaire,

- On crée des index secondaires pour chaque attribut ;
- Les index secondaires contiennent des couples (clés secondaires, clés primaires) ;
- On met pas les adresse des articles dans les index secondaires ;
- On passe toujours par l'index primaire, ce qui permet d'éviter de réarranger tous les index secondaires en cas de suppression.

Quelques définitions *Complément*

Voici la signification des termes propres à l'indexation.

- Une **clé** (d'indexation) sert d'identifiant pour chaque article du fichier de données. En toute rigueur, il faudrait toujours distinguer la clé (les noms d'attributs) de la valeur de la clé (celles que l'on trouve dans un enregistrement).
- Une **adresse** est un emplacement physique sur le support, qui peut être soit celle d'un bloc, soit un peu plus précisément celle d'un enregistrement dans un bloc.
- Une **entrée** (d'index) est un enregistrement constitué d'une paire de valeurs. La première est la valeur de la clé, la seconde une adresse.
- Un **index** est un fichier structuré dont les enregistrements sont des entrées.

3.1. Récupération des articles possédant les attributs X et Y

On suppose l'existence des index sur les champs X et Y. La procédure suivante permet de retrouver les articles en questions :

- L'index sur X donne l'ensemble des clés primaires avec l'attribut X ;
- L'index sur Y donne l'ensemble des clés primaires avec l'attribut Y ;
- L'intersection donne l'ensemble des clés primaires des articles contenant les attributs X et Y.

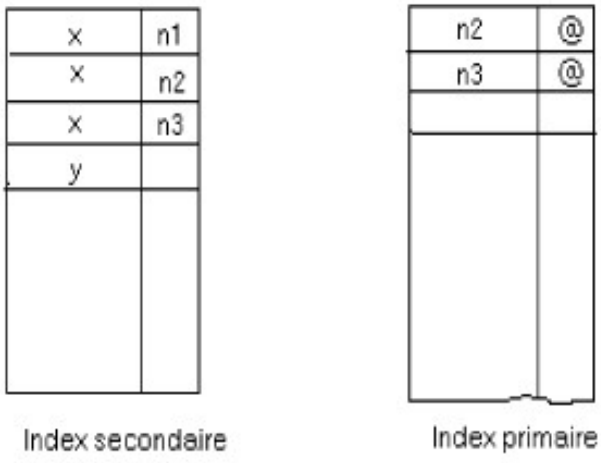
3.2. Avantages des méthodes d'index

Les méthodes d'index ont les avantages suivants :

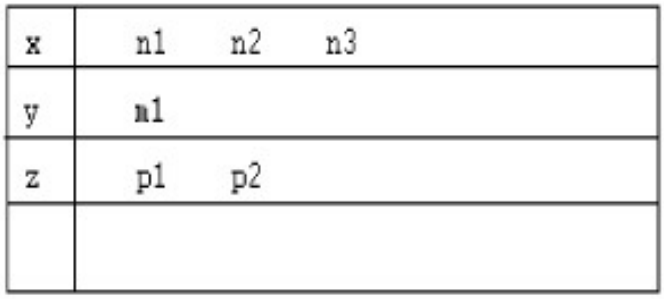
- Faire la recherche dichotomique sur le fichier d'index est beaucoup plus rapide que sur le fichier de données ;
- Permettent la recherche dichotomique même pour les articles de longueur variable.

3.3. Organisation d'index secondaire

- **Première solution :** Les clés secondaires sont dupliquées, ce qui peut rendre l'index volumineux.

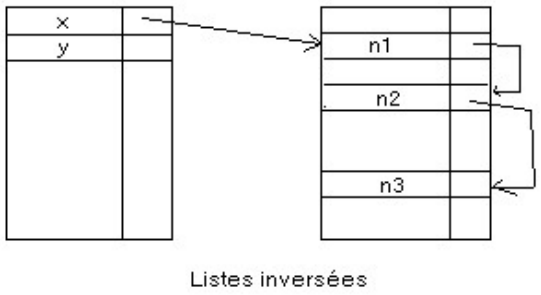


- **Deuxième solution :** Pour ne pas répéter les clés secondaires, cette solution consiste à organiser l'index secondaire.



C'est à dire par un tableau, mais, il y a problème de la taille (nombre de clés primaires par clés secondaires).

- **Troisième solution :** Cette solution consiste à utiliser les "**listes inversées**". Ici, index secondaire référençant des listes chaînées de clés primaires.



On les appelle ainsi car pour retrouver un article on commence par la fin, c'est à dire : clé secondaire, puis la clé primaire avant d'arriver à l'article lui-même.

STRUCTURES D'ARBRES

Jusqu'à présent, nous avons étudié les index primaire et secondaire basés sur des pointeurs et la recherche dichotomique. Cependant, ces approches ont des limites lorsque le fichier est très volumineux, de sorte qu'il n'est pas possible de garder l'index en RAM. Et le fait de conserver l'index dans la mémoire secondaire est coûteux, car :

1. la recherche dichotomique nécessite beaucoup d'accès disque ;
2. l'index doit être ordonné (pour pouvoir effectuer une recherche dichotomique).

Nous allons donc explorer une meilleure façon d'effectuer des insertions et des suppressions d'éléments en ne procédant qu'à une réorganisation locale : les structures arborescentes.

1. Les arbres B

Définition

Les arbres B (où B signifie "équilibré") sont une structure d'arbre qui garantit l'équilibrage de l'arbre. Dans un arbre B d'ordre n :

- La racine a au moins deux fils ;
- Chaque nœud, autre que la racine, a entre n/2 et n fils ;
- Chaque nœud peut être caractérisé par : maximum (n-1) valeurs de clés.

Structure de l'arbre B

- Dans chaque nœud, on trouve les valeurs des clés des enregistrements du fichier de données ;
- Le niveau le plus bas représente les feuilles ;
- Les niveaux de l'arbre B situés au-dessus des feuilles, sont les niveaux internes ;
- L'arbre est équilibré, tous les chemins de la racine vers les feuilles ont la même longueur. (Tous les nœuds feuille sont au même niveau)

Si A1, A2, ... An sont les n sous arbres issus d'un nœud donné avec les clés k1, k2, ...,Kn-1 dans l'ordre ascendant, alors :

- toutes les clés dans A1 sont < k1 ;
- toutes les clés dans Aj (j=2,3, ...n-1) sont > kj-1 et < kj ;
- toutes les clés dans An sont > kn-1.

1.1. Utilisation pour le stockage de données

Deux façons de ranger les articles :

1. Dans les nœuds avec les clés (**B-arbre = fichier**) → **Fichier arborescent**.
2. Séparément, donc un pointeur additionnel dans le nœud vers l'information (**B-arbre = Index**) → **Index arborescent**

B-arbre (1)	B-arbre (2)
Nœud = (cle + Pteur+ Info)	Nœud = (cle + Pteur + Adresse)

Remarque

- Si le fichier est grand, on préfère d'utiliser l'organisation (2).

Fondamental

Un nœud de l'arbre = un bloc sur le disque.

1.2. Technique d'insertion

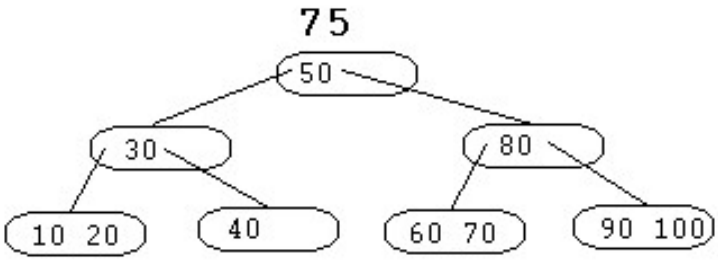
L'insertion dans un arbre B suit une procédure dite d'éclatement. L'insertion est ascendante (*Bottom-Up*)

Méthode

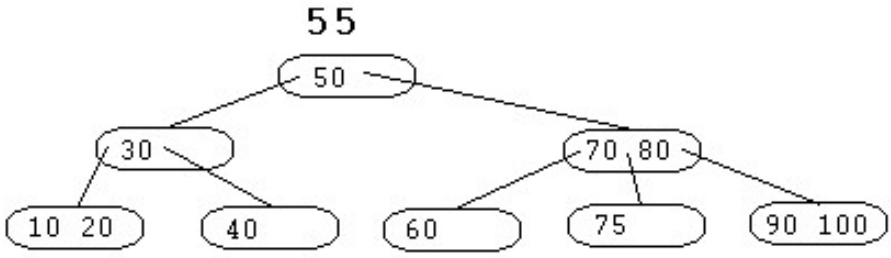
1. Comparer la clé à insérer avec la clé de la racine ;
2. Si la clé est plus petite, aller dans le sous arbre gauche. Sinon, aller dans le sous arbre droit ;
3. Si non trouvé, on est sur une feuille :
    - Si la feuille contient moins de (n-1) clés, insérer la clé ;
    - Sinon, **Éclatement :**
        1. Diviser la feuille en deux n1 et n2 ;
        2. Les clés inférieures à la clé médiane sont dans le nœud à gauche, les clés supérieures à la valeur de clé médiane sont dans le nœud droit ;
        3. La clé médiane est montée vers le père.

Exemple

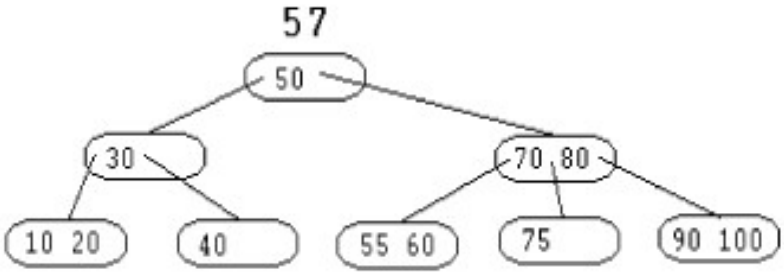
Considérons l'arbre B suivant d'ordre 3 (au maximum 2 clés et 3 pointeurs) :



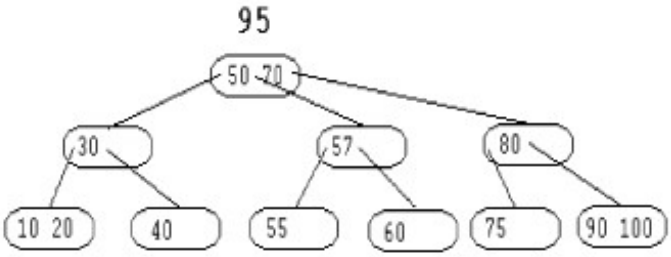
La clé 75 provoque l'éclatement du nœud (60 70). La clé du milieu, c'est à dire 70 est transférée dans le nœud père comme le montre la figure suivante :



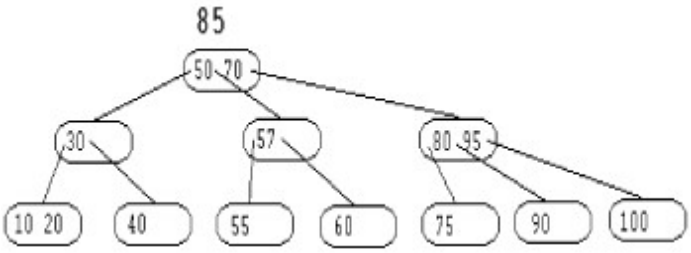
La clé 55 est insérée dans le nœud contenant la clé 60. L'arbre devient :



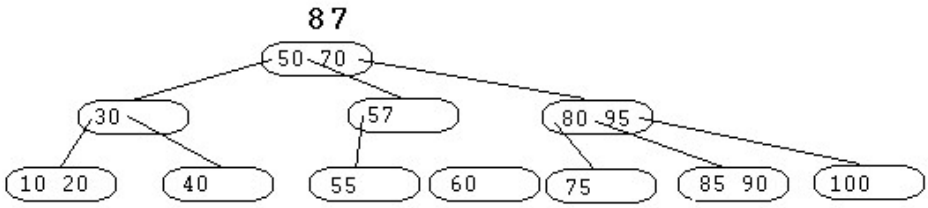
La clé 57 provoque l'éclatement du nœud (55 60). Le nœud père (70 80) est à son tour éclaté. La clé 70 migre vers le nœud racine.



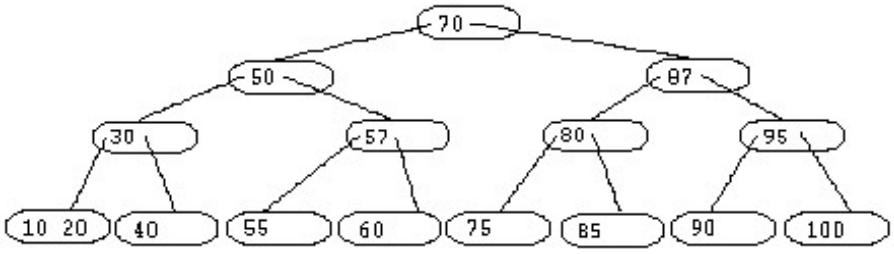
La clé 95 provoque l'éclatement du nœud (90 100). La clé 90 est donc déplacée dans le nœud contenant la clé 80 comme le montre la figure suivante :



La clé 85 est insérée dans le nœud contenant la clé 90. L'arbre devient :



La clé 87 provoque l'éclatement du nœud (85 90). Le nœud (80 95) est éclaté à son tour. Le nœud racine est aussi éclaté et le niveau de l'arbre augmente. On obtient ainsi l'arbre suivant :



- C'est une ***construction Bottom-Up***, c'est à dire que l'arbre est construit de bas en haut. Par conséquent, l'équilibrage est garanti.
- A travers cet exemple, on peut donc constater que l'arbre B permet des réorganisations locales lors des insertions. AU lieu de tout réorganiser globalement comme dans les index classiques, l'arbre B procède par réorganisations locales.

  Par exemple, si une feuille dépasse sa capacité (n-1) clés après une insertion, on la divise en deux feuilles, et cette réorganisation est locale car elle ne concerne que quelques nœuds.

1.3. Technique de suppression

Lors de la suppression dans un arbre B, il faut supprimer un article tout en préservant la qualité de l'arbre, c'est à dire en gardant **au moins n div 2 clés** dans le nœud. La suppression est ascendante (*Bottom-Up*)

Méthode

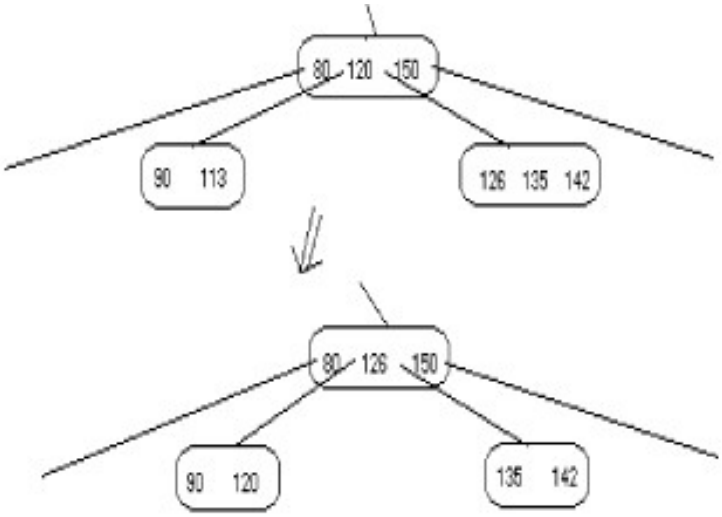
Soit "X" le nœud contenant la clé à supprimée ;

1. Examiner les frères à droite et à gauche ;
2. Si l'un des frères (gauche ou droit) contient plus de n div 2 clés, alors **Emprunt :**
   - La clé, soit K_S, dans le nœud père qui sépare entre les deux frères est ajoutée au nœud "X" et la dernière ou la première clé(première si frère droit, dernière si frère gauche) est ajoutée au père à la place de K_S (on fait une redistribution) ;
3. Si les deux frères contenaient exactement n div 2 clés, alors **Fusion :**
   - Le nœud "X" et l'un de ses frères seront concaténés en un seul nœud qui contient aussi la clé séparatrice de leur père ;
4. Si le père contient seulement n div 2 clés et par conséquent il n'a pas de clé à donner, alors il peut emprunter de son père et frère ;
5. Si les frères du père n'ont pas de clés à donner, le père et son frère peuvent aussi être concaténés et une clé est prise du grand père ;
6. Si tous les antécédents d'un nœud et leurs frères contiennent exactement n div 2 clés, un clé sera prise de la racine. Si la racine avait plus d'une clé, ceci termine le processus ;
7. Si la racine contenait une seule clé, elle sera utilisée dans la concaténation. Le nœud racine est libéré et le niveau de l'arbre est réduit d'une unité.

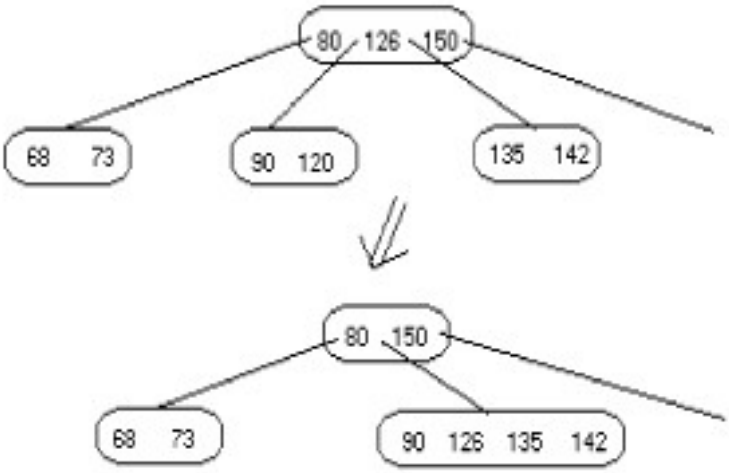
Exemple

Considérons l'arbre B suivant d'ordre 5 :

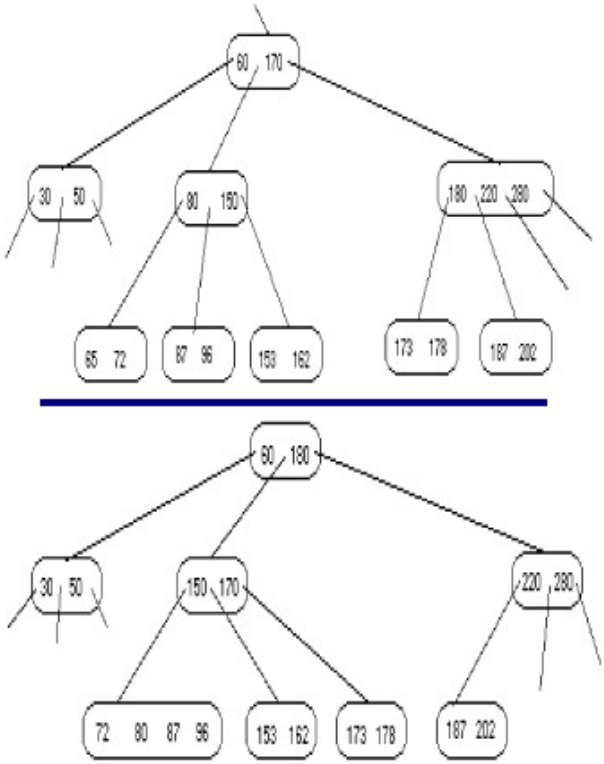
2. Suppression de la clé 113 :



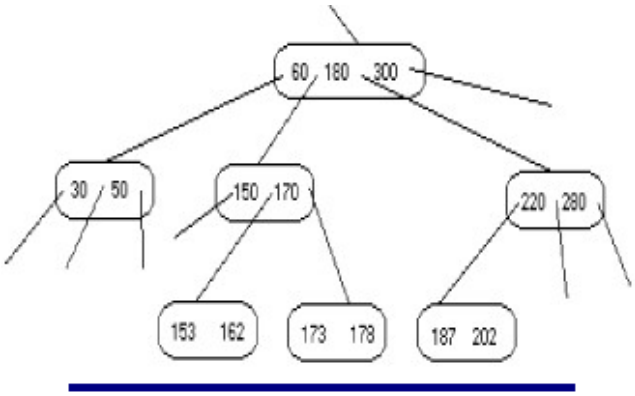
3. Suppression de la clé 120 et concaténation :

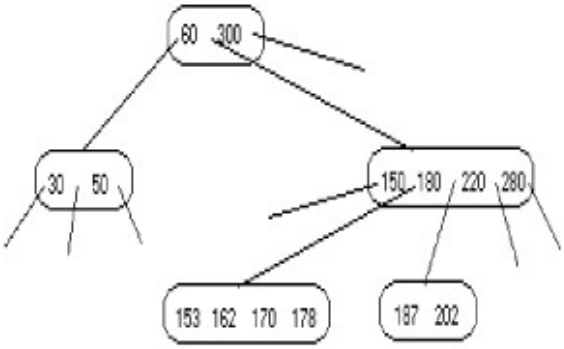


4. Suppression de 65, concaténation et emprunt :



5. Suppression de 173 :

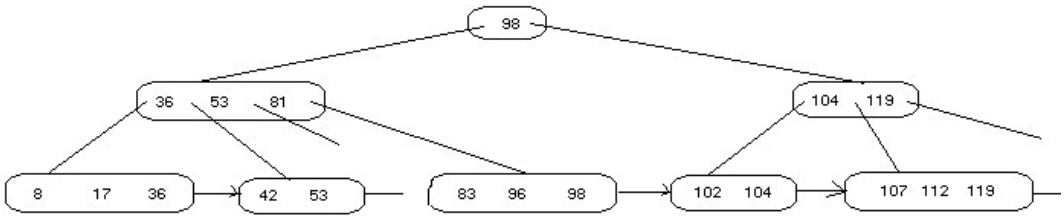




1.4. Amélioration des Arbres B

a) Arbres B+

Dans un arbre B+ toutes les clés sont maintenues au niveau des feuilles. Les articles(ou les pointeurs vers les articles) sont au niveau des feuilles. Les nœuds feuilles sont chaînés.



- La liste liée des nœuds feuille est appelée **ensemble des séquences**.
- La recherche ne s'arrête pas quand la clé est trouvée comme dans le cas des arbres B. La recherche se termine donc toujours au niveau d'un nœud feuille (le signe < est remplacé par le signe <=).
- Chaque niveau de l'arbre est un index au niveau suivant. Le dernier niveau, l'ensemble des séquences, est un index au fichier lui-même.

HACHAGE

Dans cette partie, nous verrons d'autres méthodes nous permettent d'effectuer des recherche avec une complexité souvent constante, i.e., de l'ordre de O(1).

1. Fonction de hachage

Fonction de hachage — *Définition*

Une fonction de hachage h(x) permet de calculer pour une donnée x la position qu'elle devrait occuper dans une table T. En d'autres termes, cette fonction permet de localiser en un temps constant l'adresse de la donnée x dans la table T.

La fonction la plus simple serait le modulo : **h(x) = x % N** (où N est la taille de la table T).

Exemples de fonction de hachage :

D'autres exemples de fonctions de hachage bien connus comprennent :

- Fowler-Noll-Vo (FNV)
- Jenkins hash function
- djb2 (par Daniel J. Bernstein)
- CityHash
- MurMurHash
- xxhash

  etc.

FNV1-a en C — *Exemple*

```c
#define FNV1_PRIME_32 0x01000193
#define FNV1_OFFSET_32 2166136261U
static uint32_t fnv1a_32(const void *input, size_t len){
const unsigned char *data = input;
const unsigned char *end = data + len;
uint32_t hash = FNV1_OFFSET_32;
for(; data != end; ++data){
hash ^= *data;
hash *= FNV1_PRIME_32;
}
return hash;
}
```

Collision — *Définition*

Soit h une fonction de hachage, et soient x et y deux données différentes. Si h(x) = h(y), nous disons qu'il y a collision car ces deux données se verront assigner la même position dans la table T.

Pour résoudre ces collisions, plusieurs méthodes de résolution de collisions ont été proposées.

2. Méthodes de résolution des collisions

Les approches proposées pour remédier à ce problème des collisions, peuvent être divisées en deux grandes familles : celles qui **adaptent les structures de données**, et celles qui **adaptent les algorithmes d'insertion**.

2.1. Résolution de collisions par le chaînage (closed addressing)

Ces techniques consistent à modifier la table pour qu'une case puisse accueillir plus d'une valeur en quelque sorte.

a) Hachage séparé

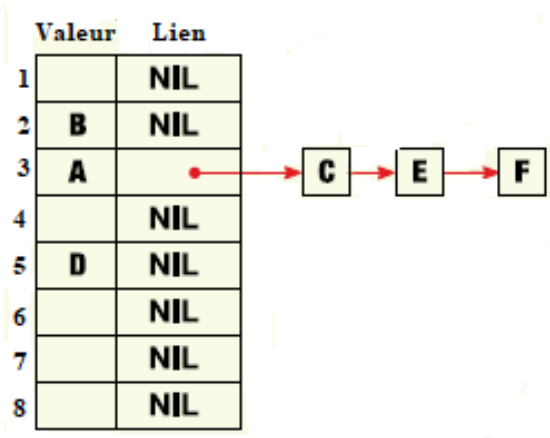
Quand des valeurs sont destinées à occuper une case déjà occupée par une autre valeur, elles sont rangées dans une liste linéaire chaînée séparée. Autrement dit, à la case i, nous avons une liste linéaire contenant les valeurs x pour lesquelles h(x) = i.

Exemple

Soient 6 valeurs à hacher (A, B, C, D, E, F) avec une fonction de hachage h telles que :

h(A) = 3, h(B) = 2, h(C) = 3, h(D) = 5, h(E) = 3, h(F) = 3.

L'insertion de ces données dans une table de 8 éléments donne :



Dans cet exemple, les valeurs C, E, et F ayant une valeur de hachage égale à 3, sont insérées dans une liste rattachée à la case 3. (La tête de cette liste est rangée à la position 3 du tableau)

b) Chaînage interne

Dans cette technique, le champ "Lien" ajouté au niveau de chaque élément du tableau contient la prochaine position de la table à examiner pour la recherche d'un élément donné. Les cases sont donc chaînées entres elles avec leurs indices.

Exemple

Soient 6 valeurs à hacher (A, B, C, D, E, F) avec une fonction de hachage h telles que :

h(A) = 3, h(B) = 2, h(C) = 3, h(D) = 5, h(E) = 3, h(F) = 3.

L'insertion de ces données dans une table de 8 éléments donne :

	Valeur	Lien
1		-1
2	B	-1
3	A	4
4	C	6
5	D	-1
6	E	7
7	F	-1
8		-1

2.2. Résolution de collisions par l'adressage ouvert (open addressing)

Dans ces techniques de résolution de collisions, la table n'est pas altérée, mais l'algorithme d'insertion (et de recherche) l'est.

a) Essai linéaire (linear probing)

La position d'un élément x dans la table T est retrouvée après des recherches de cases vides dans le voisinage.

La formule générale est h(x, i) = (x + ci) % N

La variante la plus simple des essais linéaires est celle avec: c = 1. i est le numéro de la tentative, et N la taille de la table T

En d'autres termes, si la case h(x) n'est pas vide, tester si la case h(x) + c est vide, sinon, passer à la prochaine et ainsi de suite. Si on atteint la dernière case on revient à la première etc.

Exemple

L'insertion des données précédentes dans une table T donne :

	Valeur
1	
2	B
3	A
4	C
5	D
6	E
7	F
8	

b) Double hachage

Dans cette méthode, au lieu que la séquence soit linéaire, elle est construite par une autre fonction de hachage h'. Autrement dit, s'il y a collision sur une case k, on calcule un pas p par une autre fonction de hachage. Donc, deux fonction de hachage sont utilisées, h(x) et h'(x).

3. Hachage statique et hachage dynamique

- Quand la taille de la table T est constante, nous parlons de **hachage statique**.
- Une autre classe de hachage, dite **hachage dynamique**, consiste à élargir la table T quand son taux de chargement (**d=nombres de valeurs / nombre de case**) atteint un certain seuil (exemple 70%). À l'élargissement de la table T, il est donc nécessaire de recalculer (et de déplacer éventuellement) les valeurs de hachage de toutes les données qui s'y trouvent.

4. Les fichiers avec hachage

Nous pouvons envisager des méthodes de hachage pour les fichiers aussi. La **clé** d'un enregistrement sera alors hachée pour localiser le **bloc** dans lequel l'enregistrement se trouve.

Si le bloc (pouvant contenir B enregistrements) est plein, nous adoptons une méthode de résolution de collusions comme celles entreprises avec les tables en MC.

Fichier avec hachage et essai linéaire. ? ***Exemple***

Pour la recherche par exemple, nous hachons la clé pour obtenir le bloc dans lequel elle se trouverait.

Si ce bloc est plein et que la clé n'y est pas nous effectuons des essais linéaires pour la localiser.

Choix d'une organisation

Le choix dépend des traitements et des accès a effectuer, du support et des différents taux (consultation, mise à jour...)