Cours 5 : méthodes d'accès

- Fonctions et structure des SGBD
- Structures physiques
  - Stockage des données
  - Organisation de fichiers et indexation
    - index
    - arbres B+
    - hachage

Objectifs des SGBD (rappel)

- Contrôle intégré des données
  - Cohérence (transaction) et intégrité (CI)
  - partage
  - performances d'accès
  - sécurité
- Indépendance des données
  - logique : cache les détails de l'organisation conceptuelle des données (définir des vues)
  - physique : cache les détails du stockage physique des données (accès relationnel vs chemins d'accès physiques)

Arbre B+

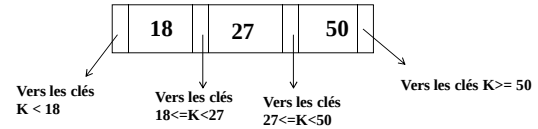
- Les arbres B+ sont des index hiérarchiques
- Ils améliorent l'efficacité des recherches
  - L'arbre est peu profond.
    - Accès rapide à un enregistrement : chemin court de la racine vers une feuille
    - Rmq: l'arbre peut être très large, sans inconvénient
  - L'arbre est toujours équilibré
    - *Balanced* tree en anglais
    - Tous les chemins de la racine aux feuilles ont la même longueur
  - L'arbre est suffisamment compact
    - Peut souvent tenir en mémoire
    - Un noeud est au moins à moitié rempli

Arbre B+ : coût d'accès

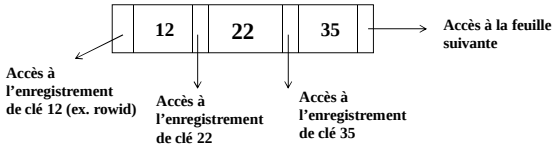
- Le coût d'accès est
  - proportionnel à la longueur d'un chemin
  - Identique quelle que soit la feuille atteinte
  - → **coût d'accès prévisible**
- Avantage :
  - permet d'estimer le coût d'accès, a priori, pour décider d'utiliser ou non un index
- Mesure du coût:
  - Nombre de nœud lus / écrits
  - Nombre de pages de données lues / écrites

Arbre B+

- Les nœuds internes servent à atteindre une feuille



- Les feuilles donnent accès aux enregistrements



5

Arbre B+: 3 types de nœuds (cas non plaçant)

- Racine
  - point d'entrée pour une recherche
- Nœud intermédiaire
  - Peut contenir une valeur pour laquelle il n'existe aucun enregistrement
- Feuille
  - Les feuilles contiennent toutes les clés pour lesquelles il existe un enregistrement
  - Les feuilles contiennent uniquement des clés de la BD

6

Ordre d'un arbre, degré d'un nœud

- Soit n le nombre de clés contenues dans un nœud
- Un arbre-B+ est d'**ordre d** si
  - Pour un nœud intermédiaire et une feuille : $d \leq n \leq 2d$
  - Pour la racine: $\mathbf{1} \leq n \leq 2d$
  - Tout nœud doit tenir dans une page :
    - 2d correspond à une page complètement remplie
- Degré sortant d'un nœud
  - Un nœud intermédiaire (et la racine) ayant n valeurs de clés a **n+1** pointeurs vers ses fils
  - Une feuille n'a pas de fils
- À part la racine, toute page est entre moitié pleine et complètement pleine (compacité, limite la hauteur de l'arbre)

7

Hauteur min et max d'un arbre B+

- Hauteur minimum quand l'arbre est plein : $\mathbf{h = \log_{2d+1}(N) + 1}$
  - 2d clés par nœud
  - N feuilles (il y a donc N*2d clés dans la table correspondante)
- Hauteur maximum quand les nœuds sont le plus vides possible : $\mathbf{h = \log_{d+1}(N) + 2}$
  - d clés par nœud, 1 pour la racine
  - 2N feuilles (2N*d clés dans la table correspondante)
- En pratique d est assez grand, donc h reste raisonnable (4 pour une relation « normale »)

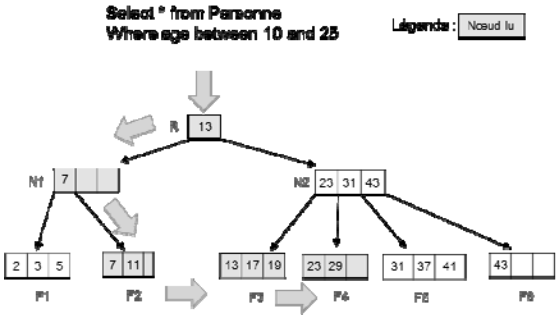
8

Arbre-B+ : chainage des feuilles

- But: supporter les requêtes d'intervalle
  - Exple de requête: ... where age between 18 and 25
  - Traverser l'index pour atteindre une borne de l'intervalle, puis parcours séquentiel des feuilles
- Chainage double pour supporter les requêtes avec une inégalité
  - Ex: ... where age < 6

Parcours du chainage

- Avantage :
  - lire un seul chemin (moins de lectures)



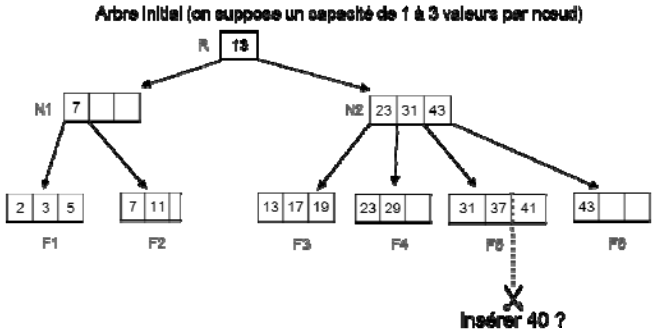
Insertion

- Rechercher la feuille où insérer la nouvelle valeur.
- Insérer la valeur dans la feuille s'il y a de la place.
  - Maintenir les valeurs triées dans la feuille
- Si la feuille est pleine (2d valeurs avant insertion), il y a éclatement. Il faut créer un nouveau nœud :
  - Insérer les d+1 premières valeurs dans le nœud original, et les d autres dans le nouveau nœud (à droite du premier).
  - La plus petite valeur du nouveau nœud est insérée dans le nœud parent, ainsi qu'un pointeur vers ce nouveau nœud.
  - Remarque : les deux feuilles ont bien un nombre correct de valeurs (elles sont au moins à moitié pleines)

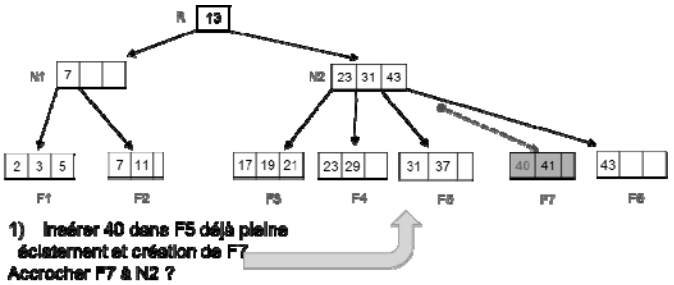
Insertion (cont.)

- S'il y a éclatement dans le parent, il faut créer un nouveau nœud frère M, à droite du premier
  - Les d premières valeurs restent dans le nœud N, les d dernières vont dans le nouveau nœud M.
  - La valeur restante est insérée dans parent de N et M pour atteindre M.
  - Rmq: M et N ont bien chacun d+1 fils
- Les éclatements peuvent se propager jusqu'à la racine et créer un nouveau niveau pour l'arbre.

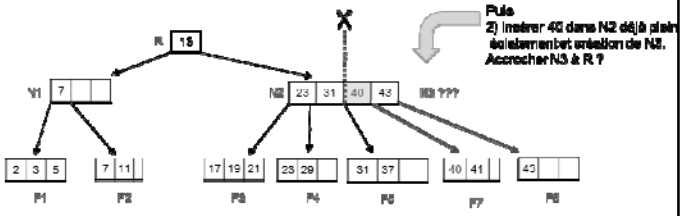
Insertion
Exemple 1 : état initial



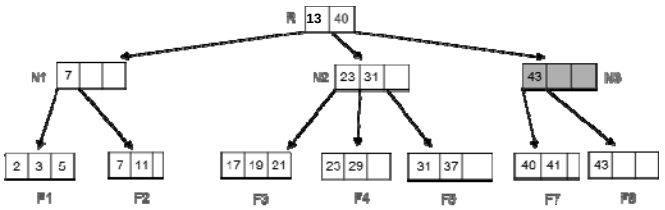
Exemple 1 (suite)



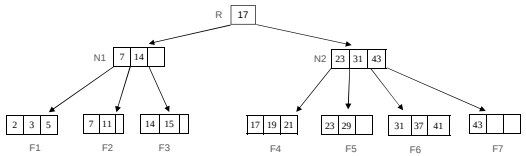
Exemple 1 (suite)



Exemple 1 (fin)



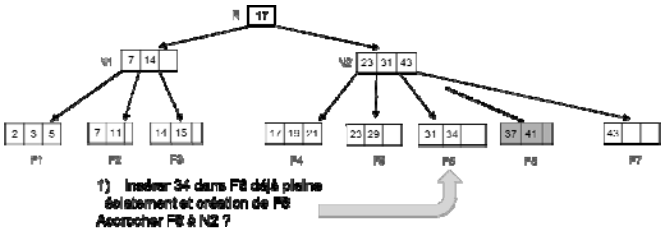
Insertion
Exemple 2 : état initial



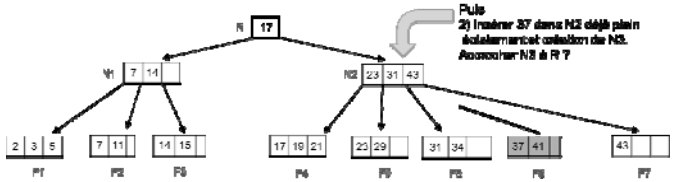
Insérer 34 ?

17

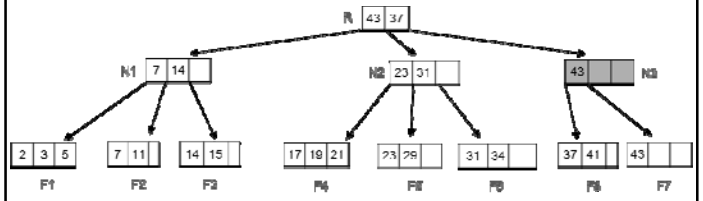
Exemple 2 (suite)



Exemple 2 (suite)



Exemple 2 (fin)



Suppression

- Supprimer la valeur (et le pointeur vers l'enregistrement) de la feuille où elle se trouve.
- Si la feuille est encore suffisamment pleine, il n'y a rien d'autre à faire.
- Sinon, redistribuer les valeurs avec une feuille **ayant le même parent**, afin que toutes les feuilles aient le nombre minimum de valeurs requis.
  - conséquence : ajuster le contenu du nœud père.
- Si la redistribution est **impossible**, il faut fusionner 2 feuilles
  - conséquence: supprimer une valeur dans le nœud père.
- Si le parent n'est pas suffisamment plein, appliquer récursivement l'algorithme de suppression.
  - Remarque1 : la propagation récursive peut entraîner la perte d'un niveau.

Résumé des opérations

- Insertion
  - simple
  - éclatement
    - d'une feuille
    - d'une feuille puis éclatement d'ancêtres
- Suppression
  - simple
  - redistribution
    - entre 2 feuilles
  - fusion
    - entre 2 feuilles
    - entre 2 feuilles puis redistribution ou fusion d'ancêtres
- Rmq
  - Toujours insérer/supprimer une clé au niveau des feuilles
  - Jamais de redistribution lors d'une insertion. L'éclatement est préférable pour faciliter les prochaines insertions.

Avantages et Inconvénients

- Avantages des organisations indexées par arbre b (b+) :
  - Régularité = pas de réorganisation du fichier nécessaires après de multiples mises à jour.
  - Lecture séquentielle rapide: possibilité de séquentiel physique et logique (trié)
  - Accès rapide en 3 E/S pour des fichiers de 1 M d'articles
- Inconvénients :
  - Les suppressions génèrent des trous difficiles à récupérer
  - Avec un index non plaçant, l'accès à plusieurs enregistrements (intervalle ou valeur non unique) aboutit à lire plusieurs enregistrements non contigus. Lire de nombreuses pages non contiguës dure longtemps
  - Taille de l'index pouvant être importante.

Exercice Arbre B+

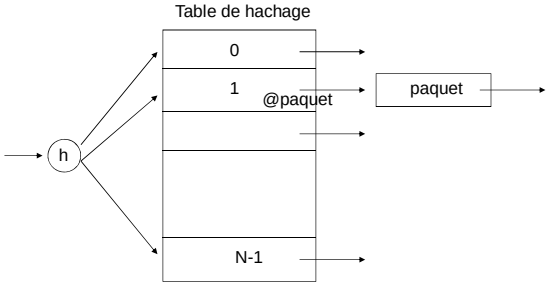
- Un arbre B+ a 3 niveaux. Chaque nœud contient 1 ou 2 clés.
- Les feuilles ont les clés 1,4, 9,16, 25, 36, ~~49~~, 54, 61, 70, 81, 84, 87, 88, 95, ~~99~~
- Les nœuds intermédiaires ont les clés 9, 54, 70, 88
- La racine contient 2 clés, les plus petites possibles parmi celles des feuilles
- Représenter l'arbre, puis insérer la clé 32

Organisations par Hachage

- Fichier haché statique (Static hashed file)
  - Fichier de taille fixe dans lequel les articles sont placés dans des paquets dont l'adresse est calculée à l'aide d'une fonction de hachage fixe appliquée à la clé.
  - On peut rajouter une indirection : table de hachage.
    - H(k) donne la position d'une cellule dans la table.
    - Cellule contient adresse paquet
    - Souplesse (ex. suppression d'un paquet)
- Différents types de fonctions :
  - Conversion en nb entier
  - Modulo P
  - Pliage de la clé (combinaison de bits de la clé)
  - Peuvent être composées

Défi : Obtenir une distribution uniforme pour éviter les collisions (saturation)

Hachage statique



Hachage statique

- Très efficace pour la recherche (condition d'égalité) : on retrouve le bon paquet en une lecture de bloc.
- Bonne méthode quand il y a peu d'évolution
- Choix de la fonction de hachage :
  - Mauvaise fonction de hachage ==> Saturation locale et perte de place
  - Solution : autoriser les débordements

Techniques de débordement

- l'adressage ouvert
  - place l'article qui devrait aller dans un paquet plein dans le premier paquet suivant ayant de la place libre; il faut alors mémoriser tous les paquets dans lequel un paquet plein a débordé.
- le chaînage
  - constitue un paquet logique par chaînage d'un paquet de débordement à un paquet plein.
- le rehachage
  - applique une deuxième fonction de hachage lorsqu'un paquet est plein, puis une troisième, etc..., toujours dans le même ordre.

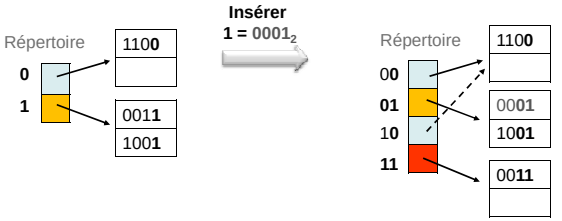
Le chaînage est la solution la plus souvent utilisée. Mais si trop de débordement, on perd tout l'intérêt du hachage (séquentiel)

Hachage dynamique

- Hachage dynamique :
  - techniques permettant de faire grandir progressivement un fichier haché saturé en distribuant les enregistrements dans de nouvelles régions allouées au fichier.
- Deux techniques principales
  - Hachage extensible
  - Hachage linéaire

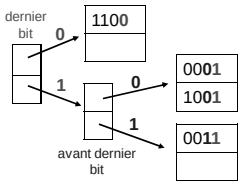
Hachage extensible

- Ajout d'un niveau d'indirection vers les paquets (tableau de pointeurs), qui peut grandir (considérer + de bits) : répertoire
- Jamais de débordement
  - Accès direct à tout paquet via le répertoire (i.e, une seule indirection)



Hachage extensible

- Répertoire similaire à un arbre préfixe (*trie*)
  - Si on considère les bits en commençant par le dernier (i.e, celui de poids faible)



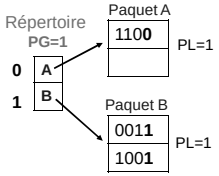
➲ Suffixe utilisé pour l'indexation = profondeur

Hachage extensible : notations

- Le répertoire est noté **R[***P*$_0$, *P*$_1$, *P*$_2$,..., P$_k$ **] PG=***pg* avec
  - *P*$_i$ ... les noms d'un paquet,
  - *pg* la profondeur globale.
- Rmq : le répertoire contient k cases avec k = 2^{pg}
- Un paquet est noté *P*$_i$**(***v*$_j$, ..., ...**) PL=***pl* avec
  - *P*$_i$ le nom du paquet, par exemple A,B, ... ,
  - *V*$_j$ les valeurs que contient le paquet,
  - *pl* la profondeur locale.
- On peut aussi préciser le contenu d'une case particulière du répertoire avec
  - R[i]= L (avec R[0] étant la 1ère case)
- La valeur v est dans le paquet référencé dans la case R[v modulo 2^{pg}]

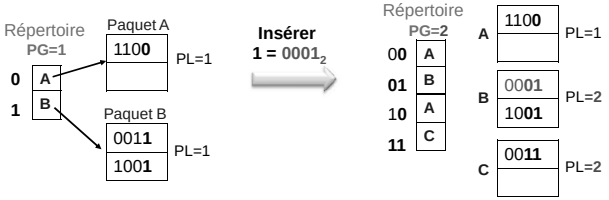
Hachage extensible
Création du répertoire

- Etat initial : N valeurs à indexer dans des paquets pouvant contenir p valeurs.
  - Il faut au moins N/p paquets
  - La taille initiale du répertoire est k=2^{PG} tq $2^{PG} \geq N/p$
  - On a k paquets donc PL =PG pour tous les paquets initiaux



Hachage extensible: Insertion

- Insertion de v dans le paquet Pi
- Cas 1) Pi est n'est pas plein, insertion immédiate dans Pi
- Cas 2) Pi est plein et PLi < PG alors éclater Pi
  - Créer un nouveau paquet Pj
  - Incrémenter les profondeurs locales de Pi et Pj (PL = PL+1)
  - Répartir les valeurs de Pi et v entre Pi et Pj
    - Si Pi est encore plein, réappliquer l'algo d'insertion : cas 2) ou 3)
- Cas 3) Pi est plein et PLi = PG alors doubler le répertoire
  - Recopier le contenu des k premières cases dans les k nouvelles cases suivantes
  - PG = PG+1 puis on retombe sur le cas 2)



Hachage extensible
Suppression et fusion

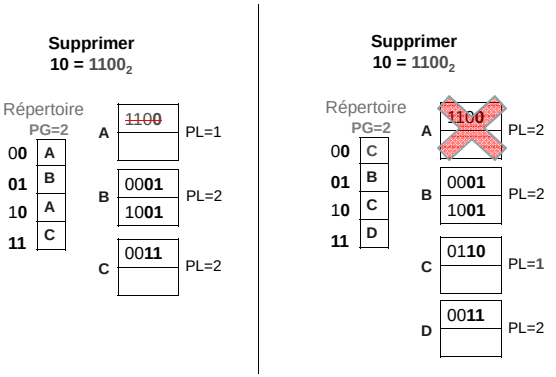
- Lors d'une suppression, si un paquet Pi devient vide et si PLi=PG alors
- on tente de fusionner Pi avec le paquet Pj référencé dans la case ayant le même suffixe que celle qui référence Pi
  - Suffixe commun (en base 2) de longueur PG -1
  - Exple si PG=3 et PLi=3, les cases ayant le même suffixe (de longueur 2) sont :
    - R[0] et R[4]
    - R[1] et R[5]
    - ...
    - R[3] et R[7]
- ~~Si Pi = Pj alors pas de fusion et Pi reste vide~~
- ~~Sinon~~ supprimer Pi et décrémenter la profondeur locale de Pj et mettre le répertoire à jour (le pointeur de Pi doit maintenant pointer Pj)

- Si pour tous les paquets restants on a PL < PG alors on peut diviser le répertoire par 2 et décrémenter PG (les deux moitiés du répertoire sont identiques). En pratique, pas toujours fait

- Rmq: aucune fusion si PLi <= PG – 1 (il n'y a pas de paquet ayant un suffixe commun de longueur PG-1). Le paquet Pi reste vide

Hachage extensible
Exemples de suppression

-



Hachage extensible (suite)

- Avantage : accès à un seul bloc (si le répertoire tient en mémoire)
- Profondeur locale/globale
- Inconvénient :
  - interruption de service lors du doublement du répertoire.
  - Peut ne plus tenir en mémoire.
  - Si peu d'enregistrement par page, le répertoire peut être inutilement gros

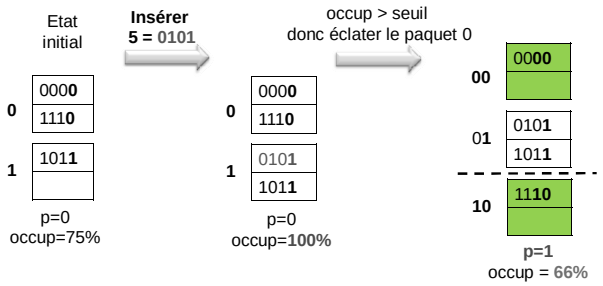
37

Hachage linéaire (1)

- Garantit que le nombre moyen d'enregistrements par paquet ne dépasse pas un certain seuil (ex. taux d'occupation moyen d'un paquet < 80%)
- Ajouter les nouveaux paquets au fur et à mesure, en éclatant chaque paquet dans l'ordre, **un par un** du premier au Nième paquet.
- Avantage par rapport au hachage extensible : pas besoin de répertoire
  - Plus rapide si les données sont uniformément réparties dans les paquets
- Inconvénient: débordement quand le seuil n'est pas atteint et le paquet est plein.
- Il faut une suite de fonction de hachage qui double le nombre de paquets à chaque fois :
  - Ex : N paquets initialement, h(x) fonction de hachage initiale
  - $h_i(x) = h(x) \bmod (2^i N)$
- Il faut marquer quel est le prochain paquet à éclater (noté p)
- Quand les N paquets ont éclaté, on recommence avec N' = 2N

Hachage linéaire (2)

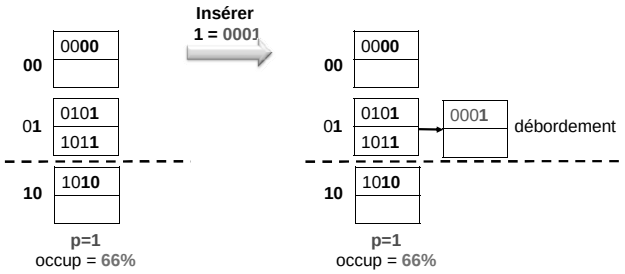
- Exemple avec 2 paquets initiaux. N=2, seuil = 80%
- $h_0(x) = h(x) \bmod 2$, $h_1(x) = h(x) \bmod 4$
- p=0 : le prochain paquet à éclater est le paquet 0



39

Hachage linéaire (3)

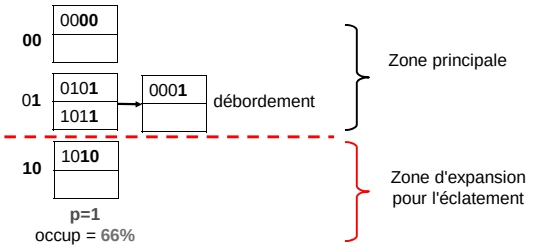
- Insérer 1 dans le paquet 1
- Débordement du paquet (pas d'éclatement car le taux d'occupation reste inférieur au seuil)



40

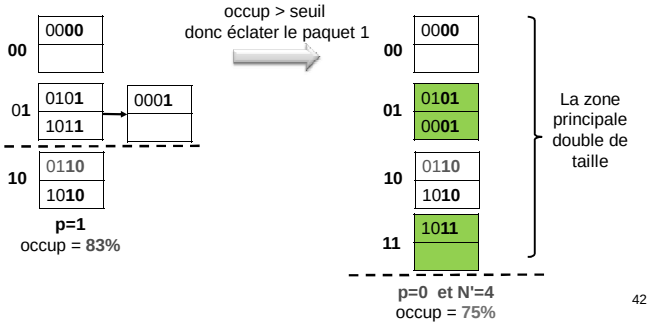
Hachage linéaire (4)

- Accès à l'enregistrement de clé c ?
- Soit i = h_0(c) et i'= h_1(v)
- Si i >= p alors lire le paquet i sinon lire le paquet i'



Hachage linéaire (5)

- Insérer 6 = 0110 dans le paquet 10_2
- Le dernier paquet de la zone principale éclate
- donc on repart à 0 après avoir agrandi la zone principale



Exercice Hachage extensible

- Chaque paquet **contient au plus 2 valeurs.**

- **Question 1.** On considère un répertoire R de profondeur globale PG=1. Avec 2 paquets P0 et P1 R={PO, P1}. Initialement les deux paquets contiennent:
- **P0**(4,8) **P1**(1,3)
- Insérer la valeur 12.
- Quelle est la profondeur globale après insertion ?
- Détailler le contenu du répertoire et des paquets modifiés ou créés, et leur profondeur locale (PL).

Conclusion

- Les fichiers séquentiels sont efficaces pour le parcours rapide, l'insertion et la suppression, mais lents pour la recherche.
- Les fichiers triés sont assez rapides pour les recherches (très bons pour certaines sélections), mais lents pour l'insertion et la suppression.
- Les fichiers hachés sont efficaces pour les insertions et les suppressions, très rapides pour les sélections avec égalité, peu efficaces pour les sélections ordonnées.
- Les index permettent d'améliorer certaines opérations sur un fichier. Les Arbres-B+ sont les plus efficaces.

Nombre de clés dans les feuilles

- Dépend de l'ordre d et du nombre de niveaux p
- Nombre maxi de clés dans l'arbre
- Arbre à 1 niveau (arbre réduit à sa seule racine): 2d clés maxi
- Arbre à 2 niveaux :
  - racine: 2d clés maxi
  - 2d+1 feuilles, soit 2d×(2d+1) clés maxi dans les feuilles
- Arbre à p niveaux :
  - Nbre maxi de clés dans les feuilles: $2d(2d+1)^{(p-1)}$
- En pratique, un arbre B+ a rarement plus de 4 niveaux car d est grand (de l'ordre de la centaine)
- Nombre mini de clés dans les feuilles :

45