Techniques d'indexation

Khaled Jouini

Institut Supérieur d'Informatique et des Technologies de Communication

---

j.khaled@gmail.com

Plan du cours - (1/2)

1. Structures d'indexation
   - Indexation, quéaco?
   - Index non-dense (Sparse)
   - Index dense
   - Index multi-niveaux
   - Arbre B+
   - (Table de) Hachage statique
   - (Table de) Hachage extensible
   - Index Bitmap
   - Filtres de Bloom
   - Arbre R

j.khaled@gmail.com

Plan du cours - (2/2)

2. Recherche Full-Text
   - Introduction
   - Concepts
   - Vectorisation du texte et poids des termes
   - Calcul du score avec la similarité Cosinus
   - L'algorithme Page Rank
   - Etude de cas avec Apache SolR
     - SolR, quésaco?
     - Recherche avec SolR
     - Schéma SolR
     - SolR et Big Data : SolrCloud

j.khaled@gmail.com

Section 1 - Structures d'indexation

j.khaled@gmail.com

Plan

Chapitre 1 - Structures d'indexation

j.khaled@gmail.com

Indexation, quéaco?

- Notion d'index : la même que pour un livre.
- Exemple : trouver toutes les pages où apparaît le mot Sousse/ trouver tous les clients résidant à Sousse

Livre	Informatique (Ex. Base de Données)
Mots-clés de l'index	Valeurs d'un attribut
Pages	Adresses mémoires où sont stockées ces valeurs

- Intérêt de l'**indexation** : permettre une **recherche rapide** d'une information.

j.khaled@gmail.com

L'indexation est-elle indispensable?

Recherche de l'information sans indexation

- Parcours séquentiel (**complexité linéaire**)
- Recherche par dichotomie si les données sont triées (**complexité logarithmique**)

$\Rightarrow$ Tolérable pour des "petits" espaces de recherche, mais pas pour les grands volumes d'informations.
Ex. BD d'empreintes digitales, Web, data warehouse, etc.

j.khaled@gmail.com

L'indexation est-elle indispensable?

Recherche de l'information avec un index

- Parcours de l'index puis ***accès direct*** à l'information (enregistrement)
- Mais,
  - Temps additionnel de maintenance de l'index suite aux opérations de modification (ajout/suppression/mise-à-jour des données)
  - Espace supplémentaire pour le stockage de l'index

⇒ **Index** : structure auxiliaire qui devient indispensable lorsque

1. L'espace de recherche de l'information est "grand"
2. Les opérations de recherche prédominent les opérations de modification

j.khaled@gmail.com

Plan

Chapitre 1 - Structures d'indexation

j.khaled@gmail.com

Introduction

Définition (**Clé de recherche**)

Liste des attributs sur lesquels les données sont indexées

Types de recherche dans un index

- **Recherche par clé** (recherche exacte)
  - À partir d'une valeur de la clé de recherche, trouver tous les enregistrements possédant cette valeur
  - Ex. "Trouver tous les employés ayant un salaire égal à 1000"
- **Recherche par intervalle**
  - Trouver tous les enregistrements dont la valeur d'un attribut est dans un intervalle donné
  - Ex. "Trouver tous les employés ayant un salaire entre 500 et 1000"

j.khaled@gmail.com

Index non-dense (*Sparse*)

Index non-dense (Sparse)

- S'applique aux fichiers de données triés.

  Exemple : fichier des employés trié sur le numéro de sécurité sociale (NSS)

- Construction de l'index :

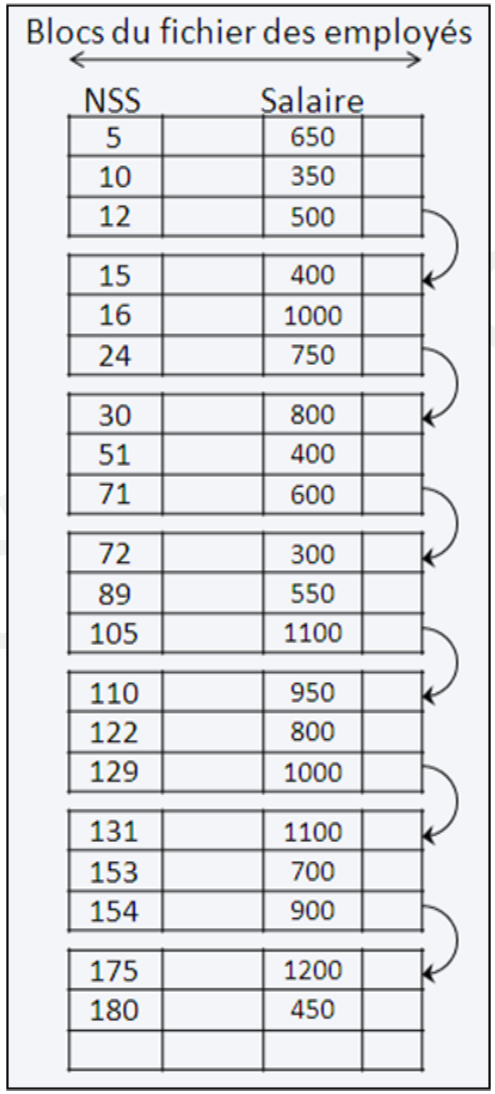
  - Clé de recherche : attribut selon lequel le fichier est trié

    Exemple : NSS.

  - Principe : associer (dans l'index) l'adresse de chaque bloc du fichier de données à la valeur de la première clé du bloc (la plus petite valeur de la clé apparaissant dans le bloc)

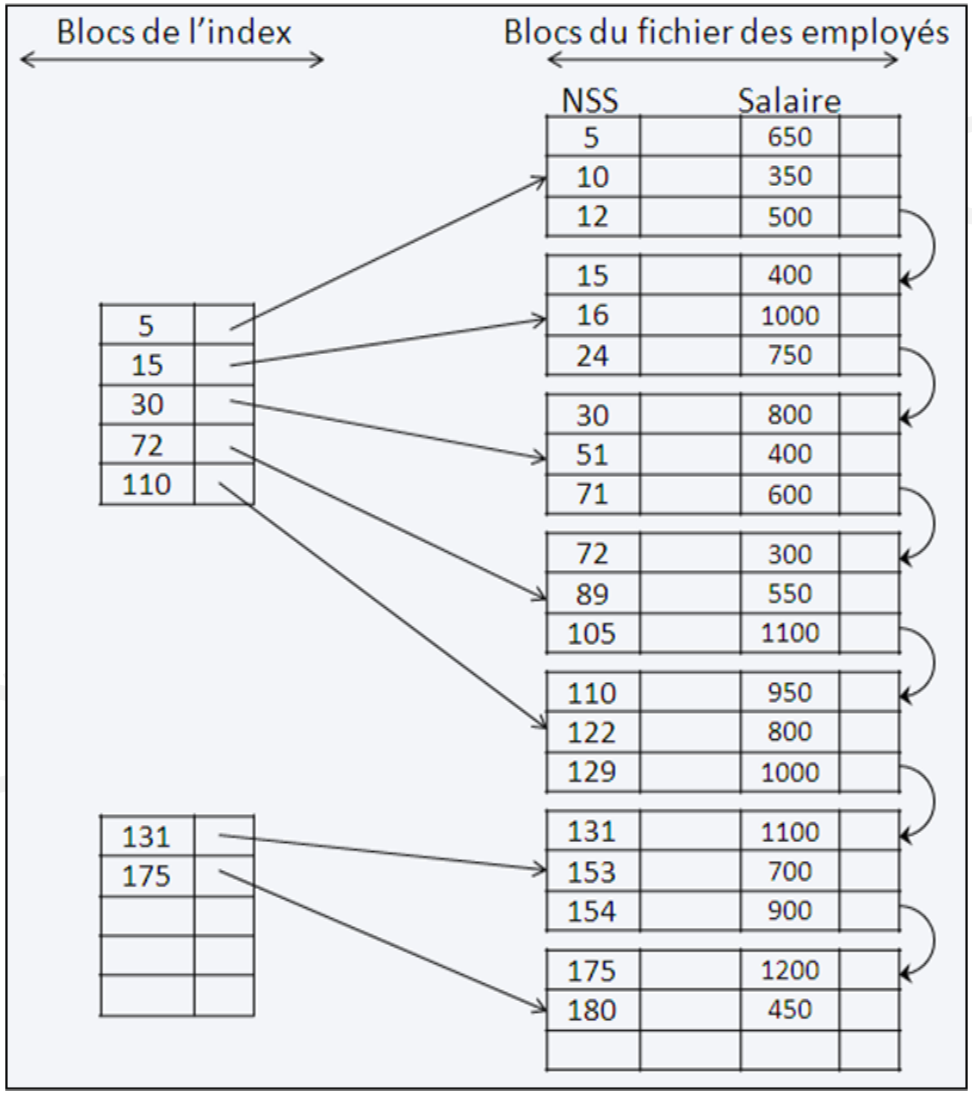
jkhaled@gmail.com

Index non-dense (*Sparse*)



j.khaled@gmail.com

Index non-dense (*Sparse*)



Index non-dense (*Sparse*)

Recherche d'une clé k

1. Recherche dans l'index la plus grande clé strictement inférieure à k
2. Suivre l'adresse correspondante

Recherche par intervalle $[k_{min}, k_{max}]$

1. Recherche dans l'index la plus grande clé strictement inférieure à k_{min}
2. Suivre l'adresse *adr* correspondante
3. Parcourir séquentiellement le fichier de données à partir du bloc *adr*
4. S'arrêter lorsque une clé > k_{max} est rencontrée

J.khaled@gmail.com

Plan

Chapitre 1 - Structures d'indexation

j.khaled@gmail.com

Index dense

Index dense

- S'applique aux fichiers de données non triés ou triés sur un attribut différent de la clé de recherche.

- Construction de l'index :

  - Associer (dans l'index) chaque valeur de la clé au bloc de données où l'enregistrement correspondant apparaît.

  $\Rightarrow$ Toutes les valeurs doivent apparaître dans l'index (d'où le nom dense).

  - Combien d'index denses et d'index non denses peut-on construire sur un même fichier ?

j.khaled@gmail.com

Index dense

Index dense

- S'applique aux fichiers de données non triés ou triés sur un attribut différent de la clé de recherche.

- Construction de l'index :

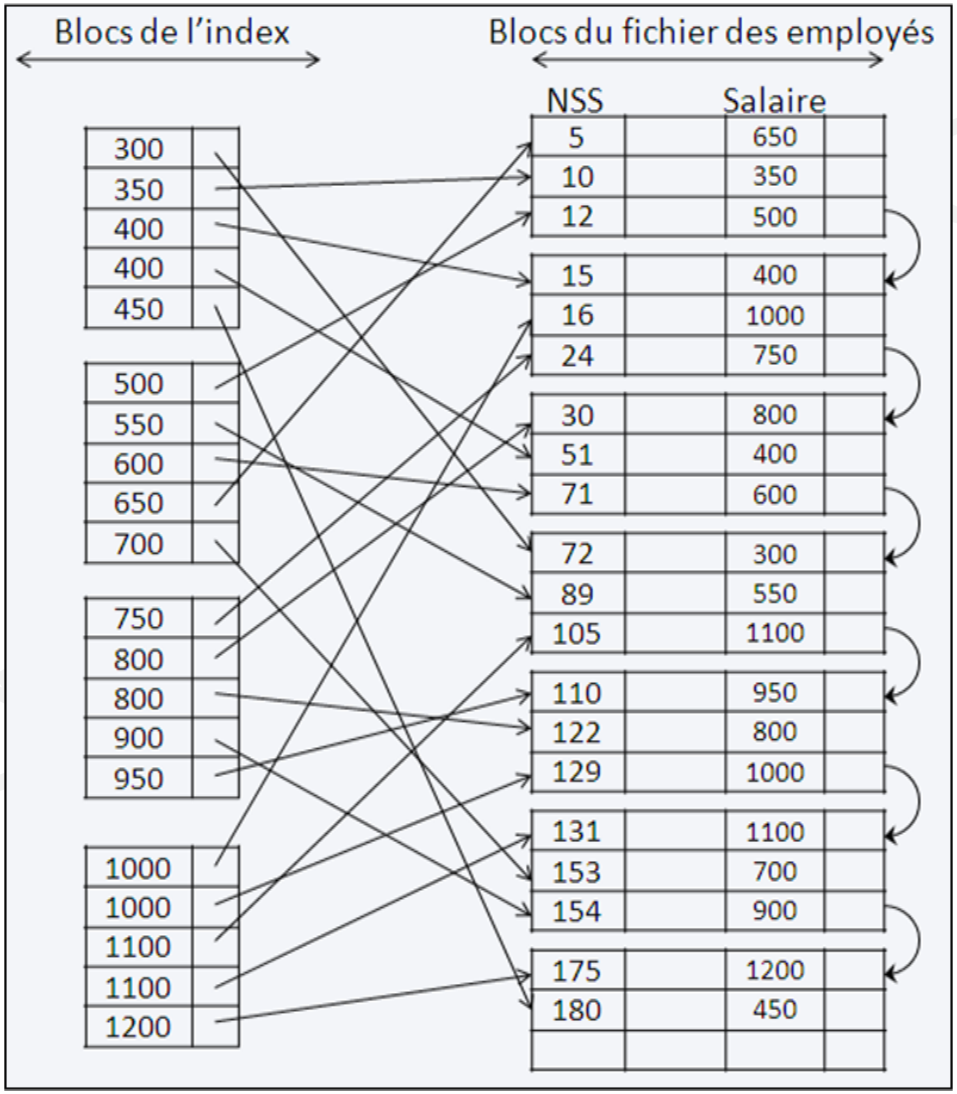
  - Associer (dans l'index) chaque valeur de la clé au bloc de données où l'enregistrement correspondant apparaît.

  $\Rightarrow$ Toutes les valeurs doivent apparaître dans l'index (d'où le nom dense).

  - Combien d'index denses et d'index non denses peut-on construire sur un même fichier ?

j.khaled@gmail.com

Index dense


Blocs de l'index
Blocs du fichier des employés
NSS
Salaire
300
350
400
400
450
500
550
600
650
700
750
800
800
900
950
1000
1000
1100
1100
1200
5
10
12
15
16
24
30
51
71
72
89
105
110
122
129
131
153
154
175
180
650
350
500
400
1000
750
800
400
600
300
550
1100
950
800
1000
1100
700
900
1200
450

Plan

Chapitre 1 - Structures d'indexation

j.khaled@gmail.com

Index multi-niveaux - a. Principe

Problème : Lorsque l'index devient à son tour volumineux, les recherches sont à nouveau pénalisées.

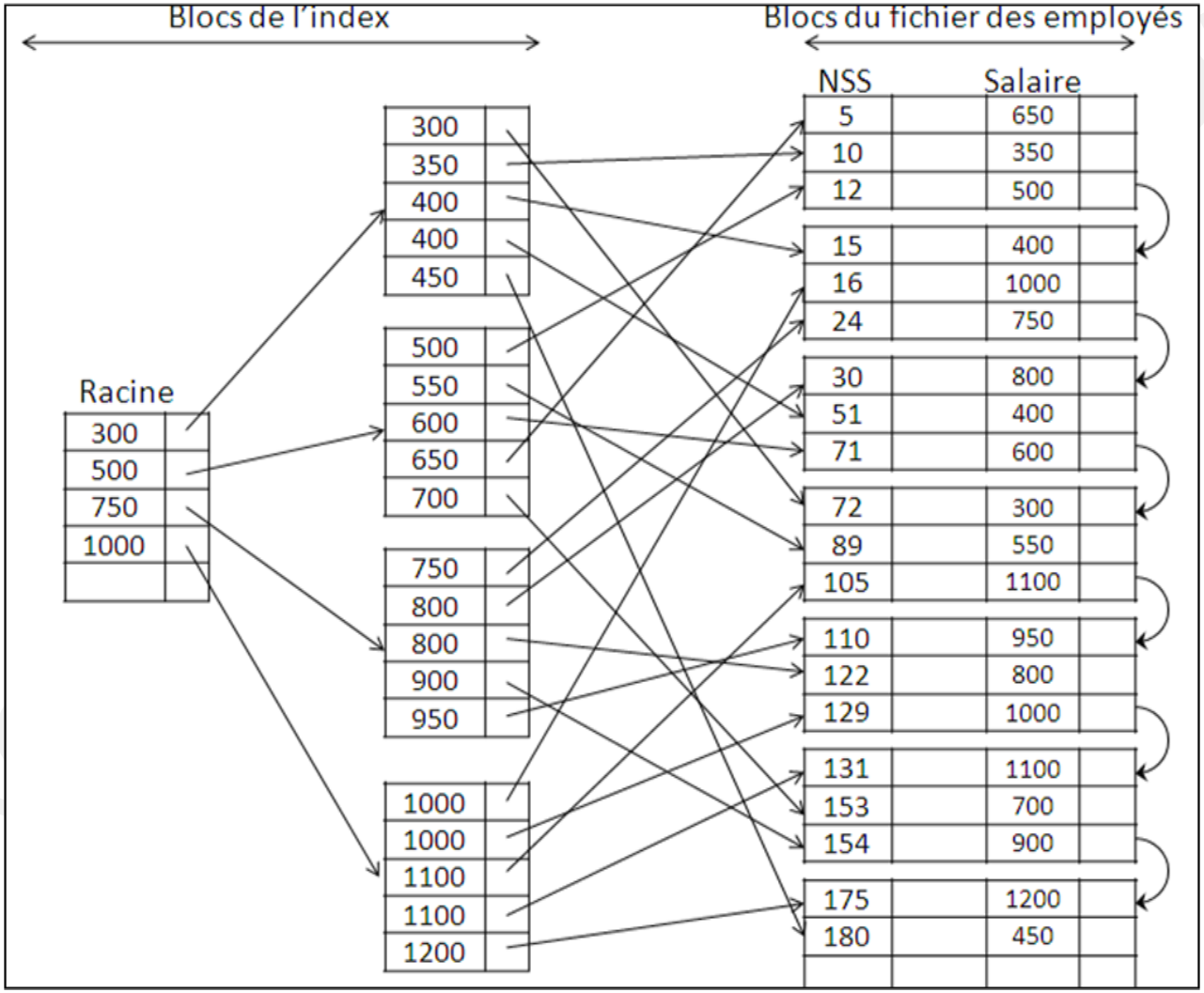
Solution :

1. Indexer les blocs de l'index. Un deuxième **niveau** d'indexation est obtenu.
2. Répéter l'opération jusqu'à ce qu'il n'y ait plus qu'un seul bloc au plus haut niveau de l'index.

Structure obtenue : **fichier séquentiel indexé** (*Indexed Sequential File*) ou structure **ISAM** (*Indexed Sequential Access Method*).

j.khaled@gmail.com

Index multi-niveaux - b. Exemple



Index multi-niveaux - c. Niveaux et noeuds

- **Index multi-niveaux** : chaque niveau partitionne davantage l'espace de recherche.
- Trois types de noeuds (blocs) :
  - **Racine** : noeud du plus haut niveau de l'index.
  - **Feuilles** (ou **terminaux**) : noeuds se trouvant au plus bas niveau de l'index. Contiennent les adresses des blocs de données.
  - **Internes** : noeuds se trouvant entre la racine et les feuilles.
- **Noeuds non terminaux** : noeuds internes + racine (si plus de 2 niveaux).
- L'index est **équilibré** (*Balanced*) : toutes les feuilles de l'index sont à la même distance de la racine.
- Dans la pratique, l'index ne dépasse pas trois niveaux, et la racine et les noeuds de second niveau sont gardés en mémoire vive.

l.khaledegmail.com

Index multi-niveaux - d. Performances

Performances

- Paramètres :

Paramètre	Signification
B	Capacité d'un noeud (nombre d'entrées par noeud)
n	Nombre de clés de recherche (ou d'enregistrements)

- **Recherche exacte** : $\mathbf{O}(\log_B(\frac{n}{B}))$
- **Recherche par intervalle** : $\mathbf{O}(\log_B(\frac{n}{B}) + \frac{a}{B})$, avec a le nombre de valeurs appartenant à l'intervalle.
- **Coût du stockage** (index non dense) : $\mathbf{O}(\frac{n}{B^2})$.
- Exemple concret : n=1 million d'enregistrements, B=100, $\frac{n}{B}$=10000 blocs, $\log_B(\frac{n}{B})$=2, $\frac{n}{B^2}$=100.

$\Rightarrow$ Très bonnes performances pour les opérations de recherche et le stockage.

j.knafod@gmail.com

Index multi-niveaux - e. Limites

Structure adaptée aux fichiers peu évolutifs. En cas de grossissement :

- Un bloc saturé déborde ⇒ un nouveau bloc de débordement est alloué.
- Peut conduire à de longues chaînes de débordement et à une dégradation des performances.
- Peut être évité par des réorganisations (recréations) périodiques de l'index (lourdes si cardinalité élevée).

⇒ besoin d'une structure permettant une **réorganisation dynamique** sans dégradation des performances.

j.khaled@gmail.com

Plan

Chapitre 1 - Structures d'indexation

j.khaled@gmail.com

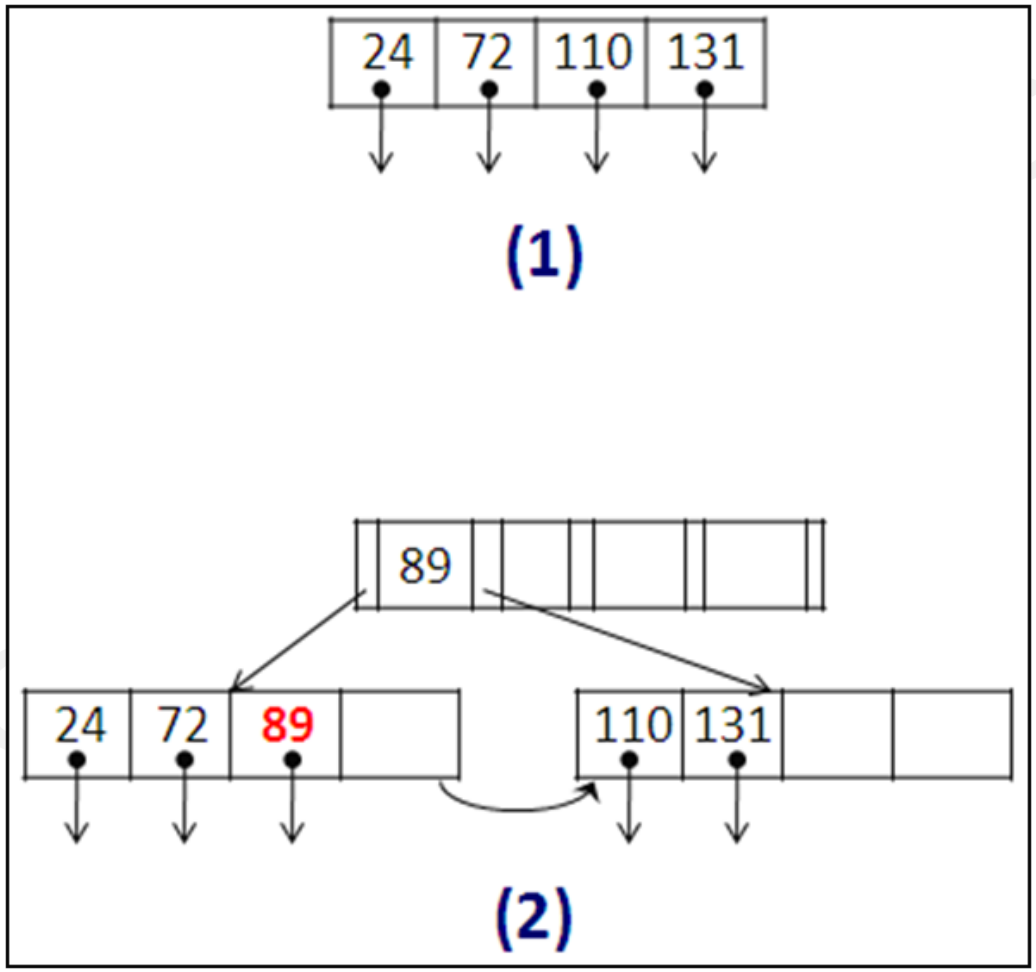
Arbre B+ - a. Vue d'ensemble

Arbre B+

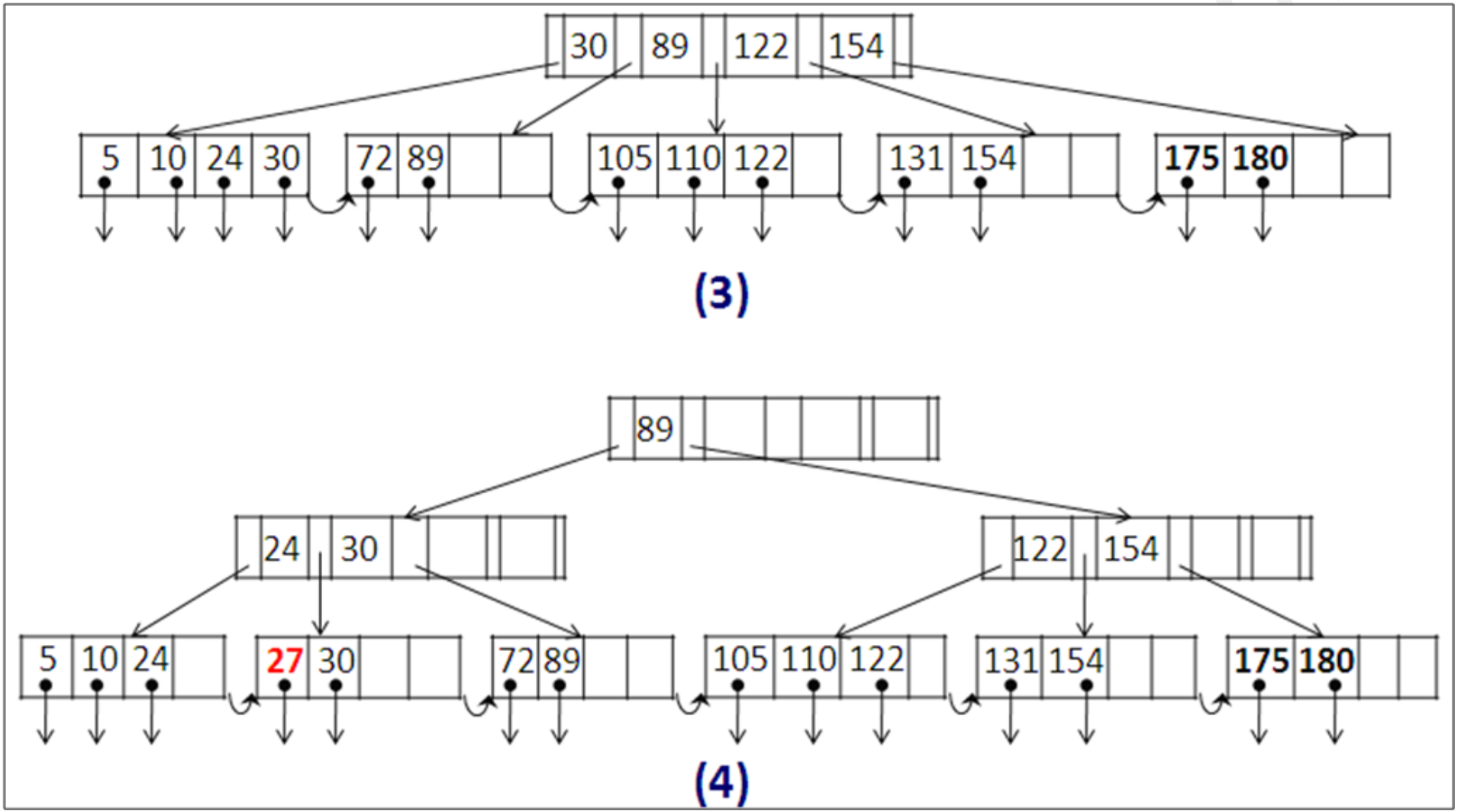
- Structure d'indexation arborescente, équilibrée (*Balanced*), inspirée de l'indexation des fichiers séquentiels.
- **Objectif** : amortissement des opérations de mises-à-jour.
- $\Rightarrow$ Création et maintenance de manière incrémentale et dynamique.

j.khaled@gmail.com

Arbre B+ - b. Exemple



Arbre B+ - b. Exemple



Arbre B+ - c. Définition

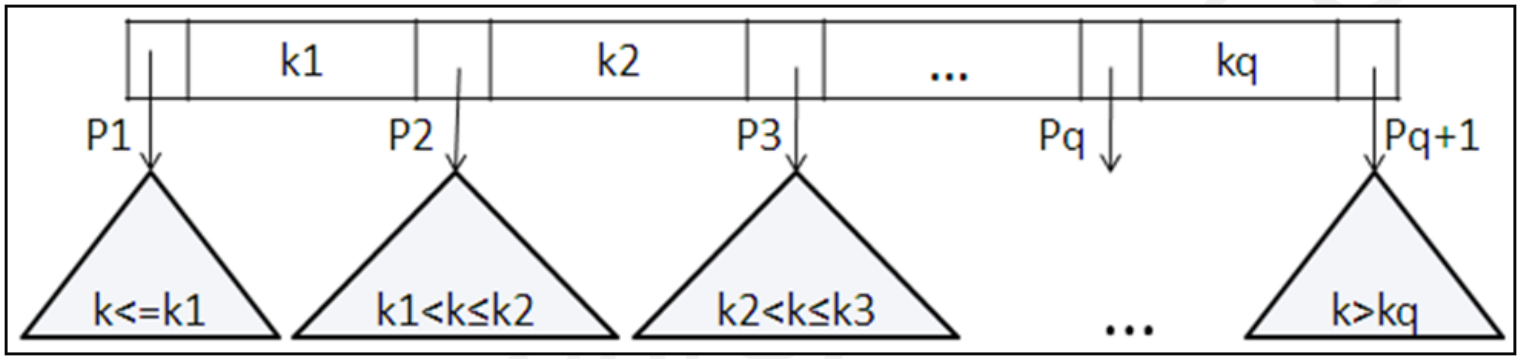
Un arbre B+ d'**ordre** *m* est un arbre équilibré tel que :

1. Chaque noeud "père" (non terminal) possède au maximum m fils.
2. Chaque noeud non terminal, excepté la racine, possède au minimum $\lceil \frac{m}{2} \rceil$ fils.
3. La racine a 0 ou au moins deux fils.
4. Un noeud non terminal contenant q clés possède $q + 1$ fils.
5. Les clés de chaque noeud sont triées.
6. Pour optimiser les recherches par intervalle :
   1. Les noeuds feuilles sont chaînés.
   2. Toutes les clés apparaissent au niveau des feuilles.

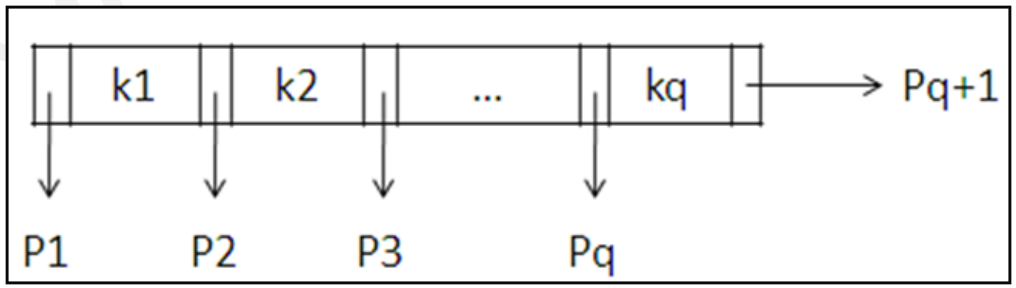
j.khaled@gmail.com

Arbre B+ - d. Structure d'un noeud

Structure d'un noeud non terminal



Structure d'un noeud feuille



Arbre B+ - e. Recherche

Recherche d'une clé k

1. Partir de la racine.
2. Soit $k_1, k_2, \ldots, k_q$ les valeurs des clés du noeud courant :
   1. Si $k \leq k_1$ (resp. $k > k_q$), la recherche continue dans le noeud référencé par p_1 (resp. p_{q+1}).
   2. Sinon, il existe $i \in [1, q[$ tel que $k_i < k \leq k_{i+1}$, la recherche continue dans le noeud référencé par p_{i+1}.
3. L'étape 2 est répétée jusqu'à ce qu'un noeud feuille soit atteint.

j.khaled@gmail.com

Arbre B+ - e. Recherche

Recherche par intervalle $[k_{min}, k_{max}]$

1. Chercher la feuille contenant k_{min}.
2. Suivre le chaînage des feuilles.
3. S'arrêter lorsqu'une clé k_i telle que $k_{max} < k_i$ est rencontrée.

j.khaled@gmail.com

Arbre B+ - f. Insertion

Insertion d'une entrée (paire) (*k*,*adr*)

1. Chercher le noeud feuille *f* où (*k*,*adr*) doit être insérée.
2. Si suite à l'insertion, *f* déborde, il est éclaté :
   1. Un nouveau noeud *f'* est créé.
   2. Soit k_{med} la clé médiane de *f'*. Les entrées de *f* dont la clé est supérieure à k_{med} sont déplacées vers *f'* (*f* et *f'* s'en trouvent à moitié pleins).
3. L'information sur l'éclatement est envoyée au niveau supérieur de l'arbre.
   1. Si *f* est une racine (n'a pas de père), un nouveau noeud racine est créé. La nouvelle racine contient l'adresse de *f*, suivie par l'entrée $[k_{med}, f']$.
   2. Si *f* a un père *p*, l'arbre de la paire $[k_{med}, f']$ y est ajoutée. Si *p* déborde, il est éclaté comme en 2, à l'exception que k_{med} n'est pas dupliquée dans le noeud éclaté.

j.khdeq@gmail.com

Arbre B+ - f. Insertion

Exercice : construisez un Arbre B+ d'ordre 4 avec les valeurs suivantes :

1. 35, 10, 15, 20, 50, 30, 45, 55, 60, 40, 25, 5
2. 5, 10, 15, 20, 25, 30, 35, 40, 45, 50, 55, 60

j.khaled@gmail.com

Arbre B+ - g. Performances

- **Paramètres :**

Paramètre	Signification
B	Capacité d'un bloc en termes du nombre total d'entrées qu'il peut contenir
b	Facteur de ramification (*fanout*) : nombre moyen d'entrées stockées dans un noeud. Estimée par $0.67 \times B$
n	Nombre de clés de recherche (ou d'enregistrements)

- **Recherche exacte** : $\mathbf{O}(\log_b(\frac{n}{b}))$
- **Recherche par intervalle** : $\mathbf{O}(\log_b(\frac{n}{b}) + \frac{a}{b})$, avec a le nombre de valeurs appartenant à l'intervalle.
- **Coût du stockage** : $\mathbf{O}(\frac{n}{b})$
- **Insertion** : $\mathbf{O}(\log_b(\frac{n}{b}))$

$\Rightarrow$ (Presque) parfait !

$\Rightarrow$ Utilisé par défaut dans la plupart des SGBDs, les moteurs d'indexation et les SGF.

i.khalife@gmail.com

Arbre B+ - g. Performances

- **Paramètres :**

Paramètre	Signification
B	Capacité d'un bloc en termes du nombre total d'entrées qu'il peut contenir
b	Facteur de ramification (*fanout*) : nombre moyen d'entrées stockées dans un noeud. Estimée par $0.67 \times B$
n	Nombre de clés de recherche (ou d'enregistrements)

- **Recherche exacte** : $\mathbf{O}(\log_b(\frac{n}{b}))$
- **Recherche par intervalle** : $\mathbf{O}(\log_b(\frac{n}{b}) + \frac{a}{b})$, avec a le nombre de valeurs appartenant à l'intervalle.
- **Coût du stockage** : $\mathbf{O}(\frac{n}{b})$
- **Insertion** : $\mathbf{O}(\log_b(\frac{n}{b}))$

$\Rightarrow$ (Presque) parfait !

$\Rightarrow$ Utilisé par défaut dans la plupart des SGBDs, les moteurs d'indexation et les SGF.

i.khalife@gmail.com

Arbre B+ - h. Arbre B+ plaçant (groupant)

Arbre B+ plaçant (groupant)

- Les feuilles contiennent les enregistrements ⇒ l'index contrôle le placement physique des enregistrements.
- **Intérêt** :
  - Recherche par clé : éviter une indirection.
  - Recherche par intervalle : maximiser la proximité.
- Il ne peut exister qu'un seul index groupant par fichier (table).
- Implanté dans la plupart des SGBDs (Oracle : *Index Organized Table*, etc.).

i.khaled@gmail.com

Plan

Chapitre 1 - Structures d'indexation

j.khaled@gmail.com

Hachage statique - a. Vue d'ensemble

Hachage

- Concurrent de l'arbre B+ :
  - Meilleur pour les recherches par clé.
  - Occupe très peu de place.
- Mais :
  - Réorganisation difficile.
  - Ne permet pas les recherches par intervalle.

j.khalea@gmail.com

Hachage statique - b. Principe

Principe

- s blocs sont alloués à la structure ($s \ll n$, avec n le nombre total d'entrées (k, adr)).
- Une **fonction de hachage h** associe chaque valeur de clé à un des s blocs. Exemple : $h(k) = k \ mod \ s$.
- Recherche d'une clé $k \Rightarrow$ calcul de $h(k)$ pour trouver l'adresse du bloc contenant k.
- h est choisie de manière à répartir uniformément les clés dans les blocs.

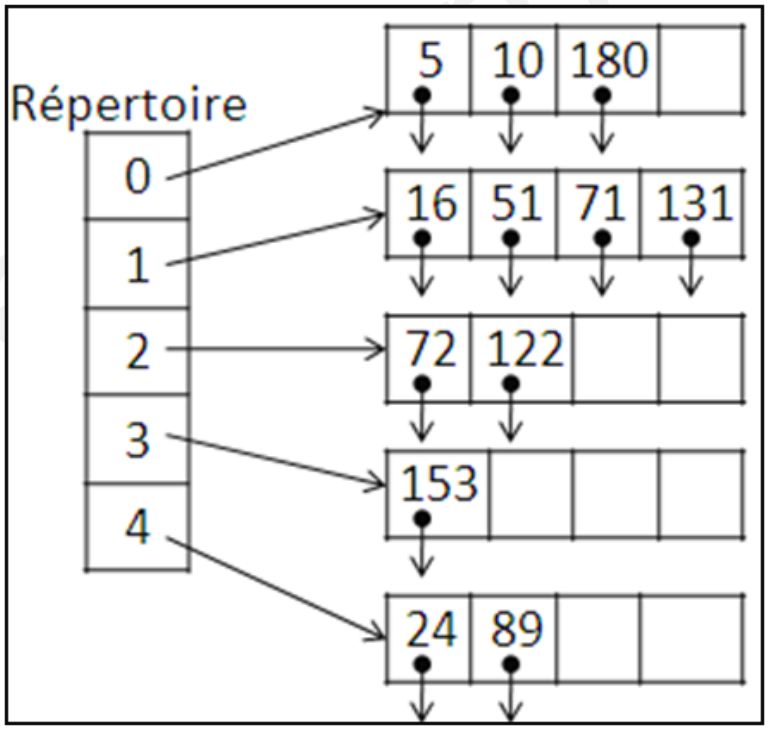
$\Rightarrow$ Simple et efficace !

i.khaled@gmail.com

Hachage statique - b. Principe

Exemple : $n = 12$, hypothèse : 4 entrées par bloc.

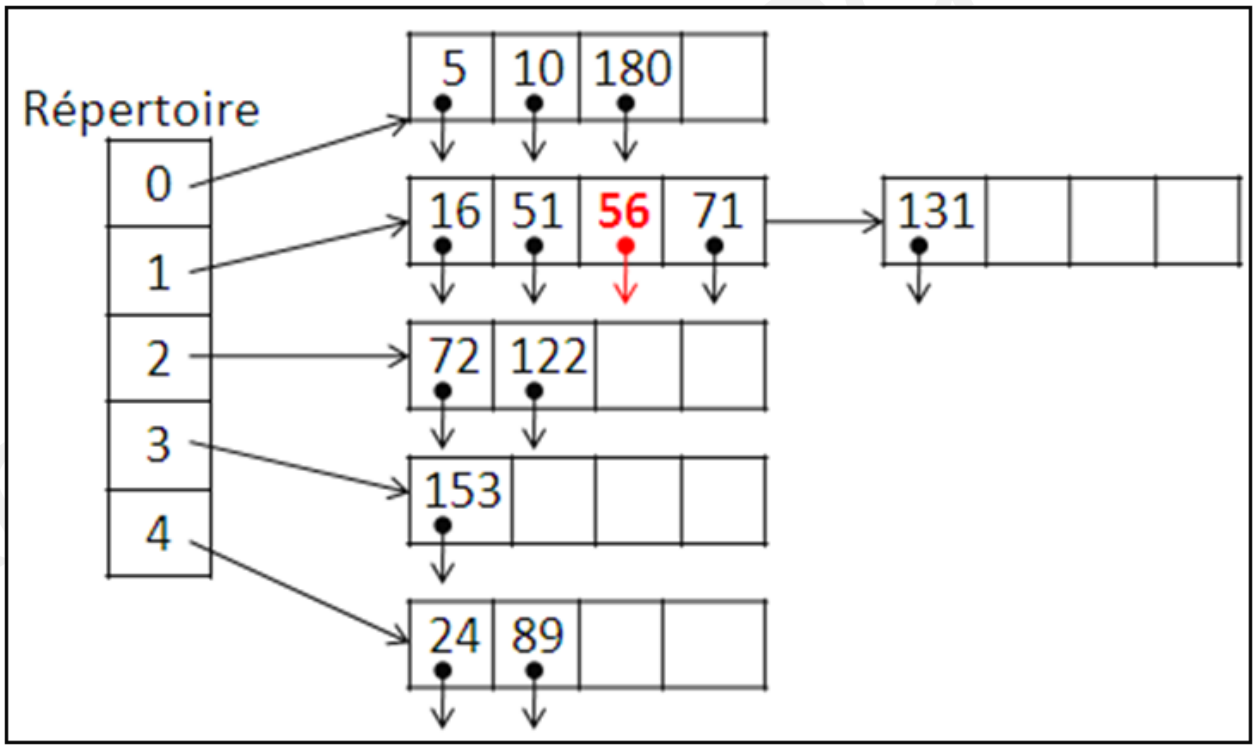
- 5 blocs sont alloués (pour garder un peu de marge).
- $h(k) = k \bmod 5$.
- Un **répertoire** de 5 entrées de 0 à 4 pointe vers les blocs.



j.khalil@gmail.com

Hachage statique - c. Limites

- Pas de recherche par intervalle.
- En cas de débordement : allocation d'un nouveau bloc, chaîné à l'ancien ⇒ dégradation des performances.



Plan

Chapitre 1 - Structures d'indexation

j.khaled@gmail.com

Hachage extensible - a. Vue d'ensemble

Hachage extensible

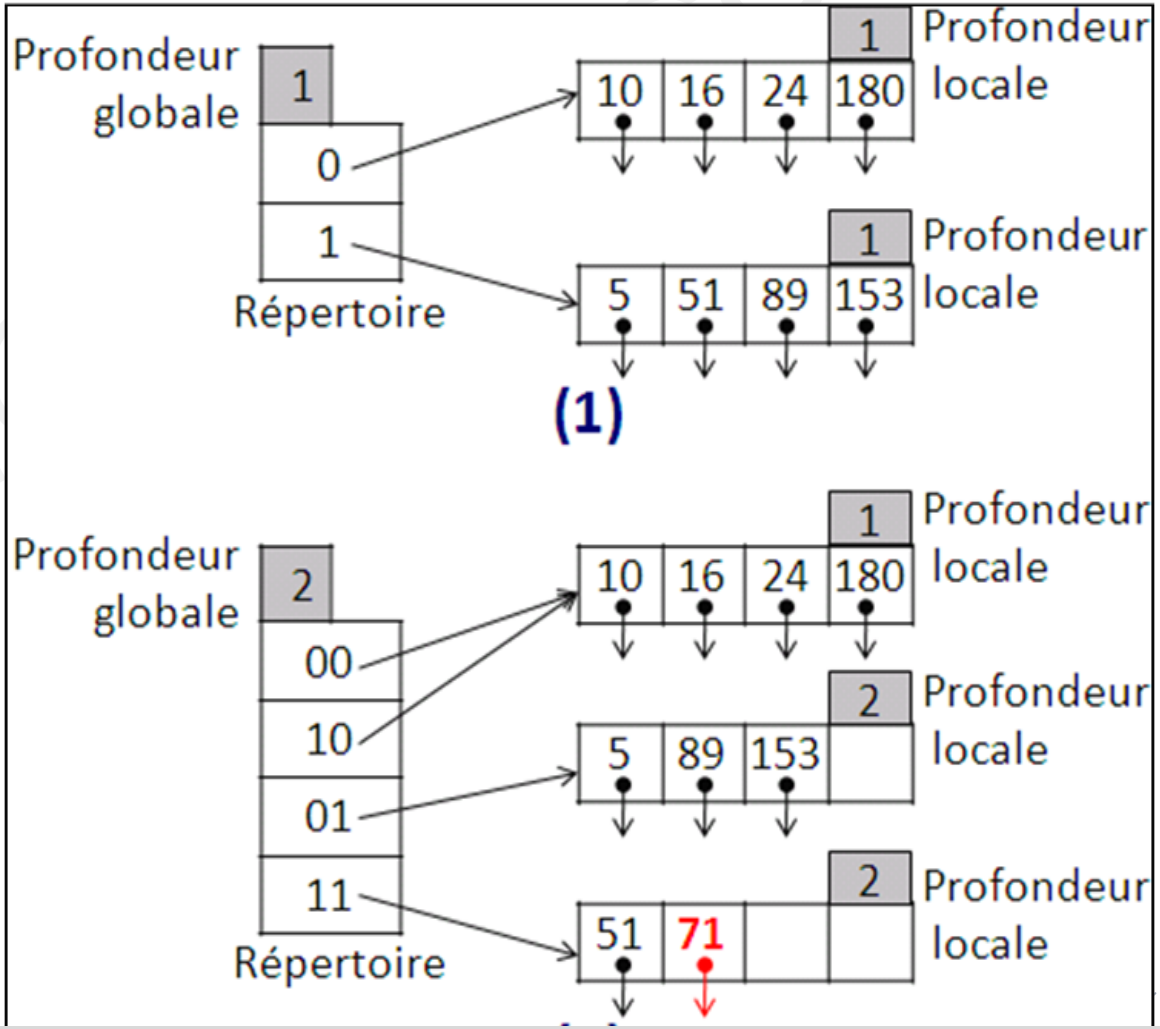
- **Objectif** : **réorganiser** la table de hachage **dynamiquement** en fonction des insertions/suppressions.
- ***h(k)*** : retourne un code binaire (Ex. $h(5) = 101$).
- Le nombre d'entrées dans le répertoire est variable :
  - Contient 2^d entrées, avec **d** **profondeur globale** du répertoire.
  - Initialement $d = 1$, puis en fonction des éclatements, $d = 2$, $d = 3$, etc.
- Les d derniers bits du résultat de h sont utilisés pour la répartition des valeurs de la clé (initialement 1 seul bit, ensuite 2 bits, etc.).

i.khaled@gmail.com

Hachage extensible - a. Vue d'ensemble

NSS	$h(NSS)$
16	10000
153	10011001
24	11000
5	101
10	1010
180	10110100
89	1011001
51	110011
71	1000111

- Plusieurs entrées du répertoire peuvent pointer vers le même bloc.
- À chaque bloc est associée une profondeur locale (quelle utilité ?).



Hachage extensible - b. Insertion

Insertion de (*k*, *adr*) dans une table de hachage de profondeur *d* :

1. Les *d* derniers bits de $h(k)$ sont utilisés pour trouver le bloc *B* où (*k*, *adr*) est à insérer.
2. Si *B* déborde :
   1. Un nouveau bloc B' est alloué.
   2. Soit d' la profondeur locale de *B*. La répartition des entrées entre *B* et B' se fait selon le $d' - 1$ avant-dernier bit : les entrées de *B* ayant ce bit à 1 sont déplacées vers B'.
   3. La profondeur locale de *B* et B' est incrémentée ($d' \leftarrow d' + 1$).
   4. Si $d' <= d$, le répertoire est mis à jour pour pointer vers B'.
   5. Sinon ($d' > d$), la taille du répertoire est doublée ($d \leftarrow d + 1$). Le répertoire est mis à jour en conséquence.

j.khaled@gmail.com

Hachage extensible - c. Remarques

- Hachage intéressant lorsque :
  - Les données sont peu évolutives.
  - Les recherches se font par clé.

  Autrement, l'Arbre B+ est meilleur.

- Dans la plupart des SGBDs, seul le hachage statique est implanté (y compris Oracle).

J.Khaled@gmail.com

Hachage extensible - d. Exercices

1. Construisez une table de hachage statique avec les valeurs ci-dessous en supposant que le nombre de blocs réservés soit 4 et qu'un bloc ne peut contenir au plus que 3 entrées.

    - 23, 39, 28, 40, 29, 24, 22, 12, 53

j.khaled@gmail.com

Hachage extensible - d. Exercices

2. Construisez une table de hachage dynamique avec les valeurs ci-dessous en supposant qu'un bloc ne peut contenir au plus que 3 entrées. Donnez une représentation de la table chaque fois qu'un nouveau bloc est alloué.

Valeur	$h(Valeur)$
23	10111
39	100111
28	11100
40	101000
29	11101
24	11000
22	10110
12	1100
53	110101

jkhaled@gmail.com

Plan

Chapitre 1 - Structures d'indexation

j.khaled@gmail.com

Index Bitmap - a. Le problème

Problème : comment indexer des données sur un attribut *A* dont la cardinalité de son domaine de valeurs est "petite" ?

Exemple, Civilité : M., Mme, Melle, Null.

- Arbre B+ : inadéquat car chaque valeur est peu sélective.
- Hachage : inadéquat car il y a beaucoup de collisions.

⇒ Or, situation fréquente surtout lorsque les données sont agrégées comme dans les entrepôts de données (*DataWarehouse*).

jkhaled@gmail.com

Index Bitmap - b. Principe

Soit n le nombre d'enregistrements à indexer $\{e_1, e_2, \ldots, e_n\}$ et A un attribut dont le domaine de valeurs comporte m valeurs possibles $\{a_1, a_2, \ldots, a_m\}$.

1. Pour chaque valeur a_j un tableau de n bits est construit. Chaque bit du tableau représente un enregistrement.

2. La i^{eme} case du tableau de a_j est mise à 1 si $e_i.A = a_j$, sinon le bit est mis à 0.

j.khaled@gmail.com

Index Bitmap - b. Principe

	NSS	Civ.	...	Sal.
e1		Melle		
e2		Mme		
e3		M.		
e4		M.		
e5		Mme		
e6		M.		
e7		M.		
e8		M.		
e9		Melle		
e10		Null		
e11		Mme		
e12		M.		

	e1	e2	e3	e4	e5	e6	e7	e8	e9	e10	e11	e12
Melle	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**0**	**0**	**0**
Mme	**0**	**1**	**0**	**0**	**1**	**0**	**0**	**0**	**0**	**0**	**1**	**0**
M.	**0**	**0**	**1**	**1**	**0**	**1**	**1**	**1**	**0**	**0**	**0**	**1**
Null	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**0**	**0**

Index Bitmap - c. Recherche

Recherche d'une clé k

1. Consultation du bitmap de k pour identifier les enregistrements ayant cette valeur.
2. Accès aux enregistrements.

Recherche multi-attributs $k_1, k_2, \ldots, k_n$

1. Application d'un opérateur logique sur les tableaux de bits associés à $k_1, k_2, \ldots, k_n$ (ET, OU, NON, XOR, etc.).
2. Accès aux enregistrements.

Calcul d'agrégat

1. Certaines requêtes peuvent être résolues sans même accéder aux données.
2. Exemple :

```
SELECT COUNT(*) FROM Emp WHERE Civilite IN ('Melle', 'Mme')
```

i.khaled@gmail.com

Index Bitmap - c. Quand utiliser un Bitmap ?

Selon Oracle :

- Petite cardinalité du domaine de valeurs : lorsque la cardinalité du domaine est 100 fois plus petite que le nombre de tuples (enregistrements).
- Très peu d'insertions/mises-à-jour.

Mais aussi :

- Entrepôts de données où les données sont agrégées par des attributs catégoriels (Ex. Date, Ville, Secteur, etc.) et les requêtes de calcul d'agrégats sont fréquentes (Ex. ventes/mois, ventes/région, etc.).
- Grand volume de données : occupe très peu de place comparativement à un Arbre B+.

j.khaled@gmail.com

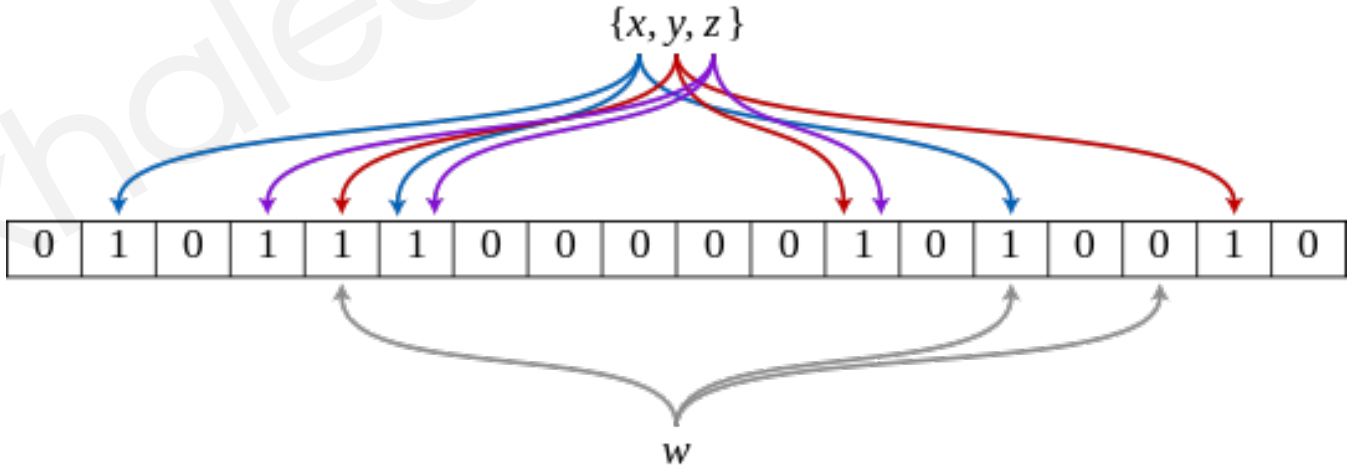
Plan

Chapitre 1 - Structures d'indexation

j.khaled@gmail.com

Filtres de Bloom

- Un filtre de Bloom n'est pas une structure d'indexation à proprement parler, mais une structure de filtrage.
- Il permet de tester rapidement :
  - l'absence d'un élément ou d'une clé k dans un ensemble (avec certitude).
  - la présence d'un élément ou d'une clé k dans l'ensemble (avec une marge d'erreur - taux de faux positifs).
- Un filtre de Bloom est composé de :
  - T : un tableau booléen de taille m, dont les cases sont initialisées à 0.
  - $(h_i)_{1 \leq i \leq H}$: H fonctions de hachage de $D \rightarrow [0, m-1]$, où D est le domaine de valeurs des éléments filtrés.



j.khaled@gmail.com

Filtres de Bloom

- Un BF permet de tester rapidement :
  - L'ajout d'une clé k à l'ensemble se fait en appliquant les H fonctions de hachage sur k, puis à mettre à 1 les cases d'indice $h_i(k)$ (pour i variant de 1 à H) dans le tableau T.
  - La vérification de l'existence d'une clé k consiste à vérifier si toutes les cases d'indices $h_i(k)$ (i variant entre 1 et k) sont à 1 dans T.
  - Exemple : http://llimllib.github.io/bloomfilter-tutorial/

j.khaled@gmail.com

Filtres de Bloom - Faux positifs

- Le taux de faux positifs dépend de H, m et K, le nombre de clés.
- Probabilité qu'un bit soit à 1 pour une seule fonction de hachage et une seule clé : $\frac{1}{m}$.
- Probabilité qu'un bit soit à 0 pour une seule fonction de hachage et une seule clé : $1 - \frac{1}{m}$.
- Probabilité qu'un bit soit à 0 pour H fonctions de hachage et une seule clé : $(1 - \frac{1}{m})^H$.
- Probabilité qu'un bit soit à 0 pour H fonctions de hachage et K clés : $(1 - \frac{1}{m})^{H.K}$.
- Probabilité qu'un bit soit à 1 pour H fonctions de hachage et K clés : $1 - (1 - \frac{1}{m})^{H.K}$.
- Probabilité qu'une clé soit testée présente alors qu'elle ne l'est pas : probabilité que H positions soient à 1 :

$$\left(1 - \left(1 - \frac{1}{m}\right)^{H.K}\right)^H$$

jkhaled@gmail.com

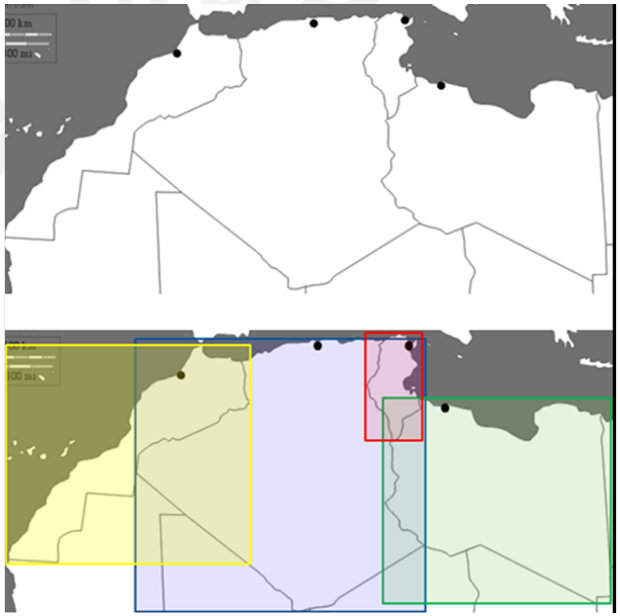
Plan

Chapitre 1 - Structures d'indexation

j.khaled@gmail.com

Arbre R - Approximation

- Arbre R : Structures d'index communément utilisée pour les données multidimensionnelles (*e.g.* données spatiales)
- Utilisation d'une approximation rectangulaire des données par des **REM** (Rectangles Englobants Minimums ou *Minimum Boundary Rectangles*).
- Avantage des REM : tous les objets sont représentés par des informations de même taille : coordonnées de 2 points $((x_1, y_1),(x_2, y_2))$.
- Moins l'espace mort introduit et le chevauchement sont importants, meilleure est l'approximation et les performances.



j.khalefa@gmail.com

Arbre R - Vue d'ensemble

Principe de l'arbre R :

1. Structure arborescente inspirée de l'arbre B+.
2. Au niveau des feuilles, grouper les REM des objets spatiaux dans des REM selon leur proximité et selon la taille des nœuds (Ex. 4 REM par nœud de l'index).
3. Au niveau suivant, grouper les REM des nœuds feuilles dans des REM encore plus grands, selon taille et proximité.
4. Répéter jusqu'à ce qu'il n'y ait qu'un seul REM pour l'espace indexé, celui de la racine.

j.khaled@gmail.com

Arbre R - Vue d'ensemble

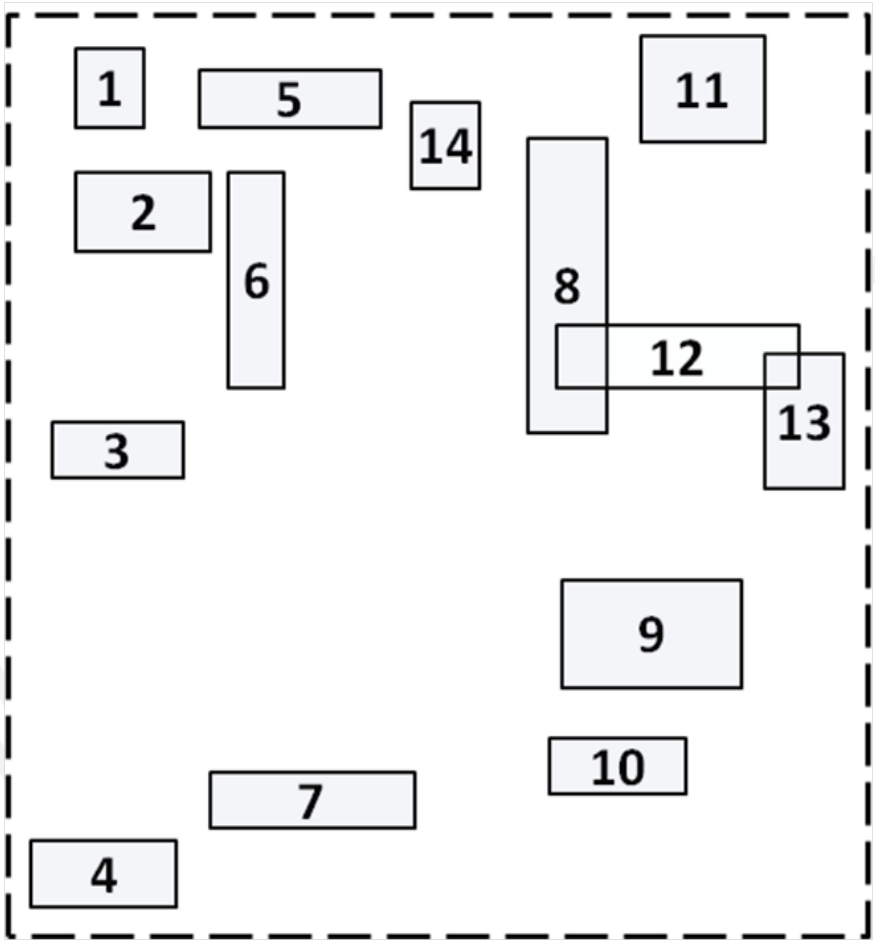
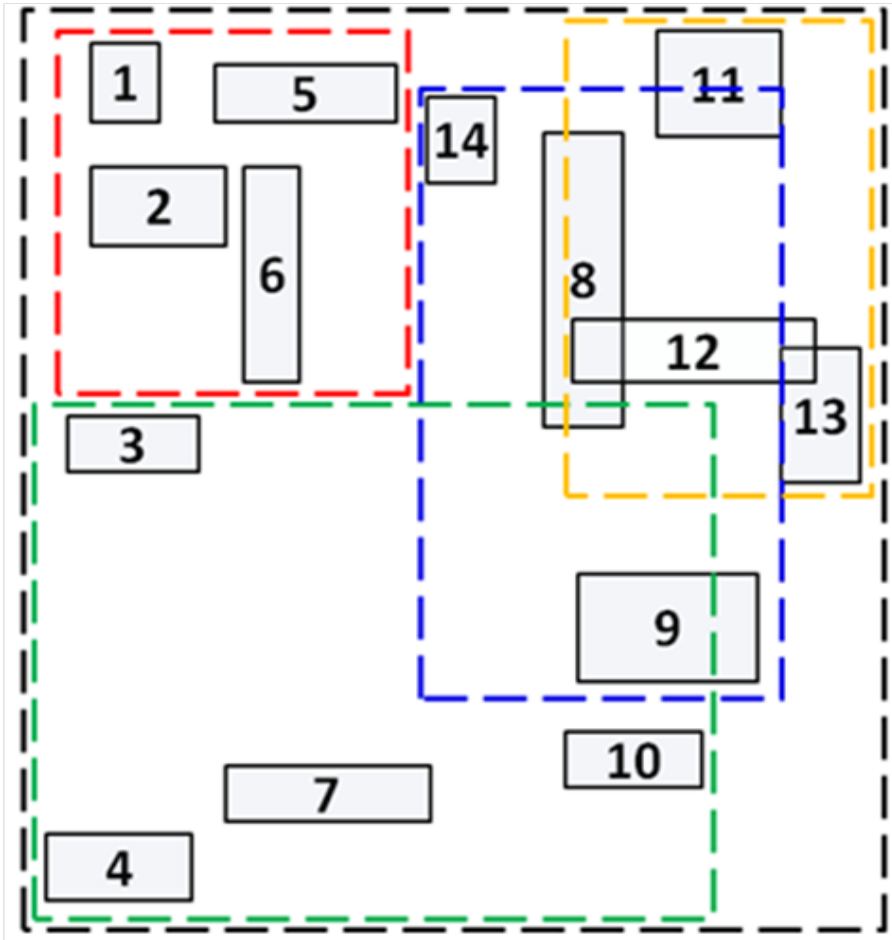
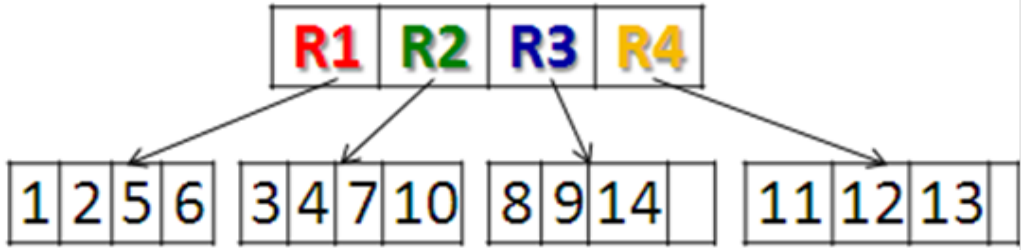


Figure: Exemple pris du livre *Introduction to Spatial Databases: with Application to GIS* Ph. Rigaux, M. Scholl, A. Voisard, 2006.

Arbre R - Vue d'ensemble

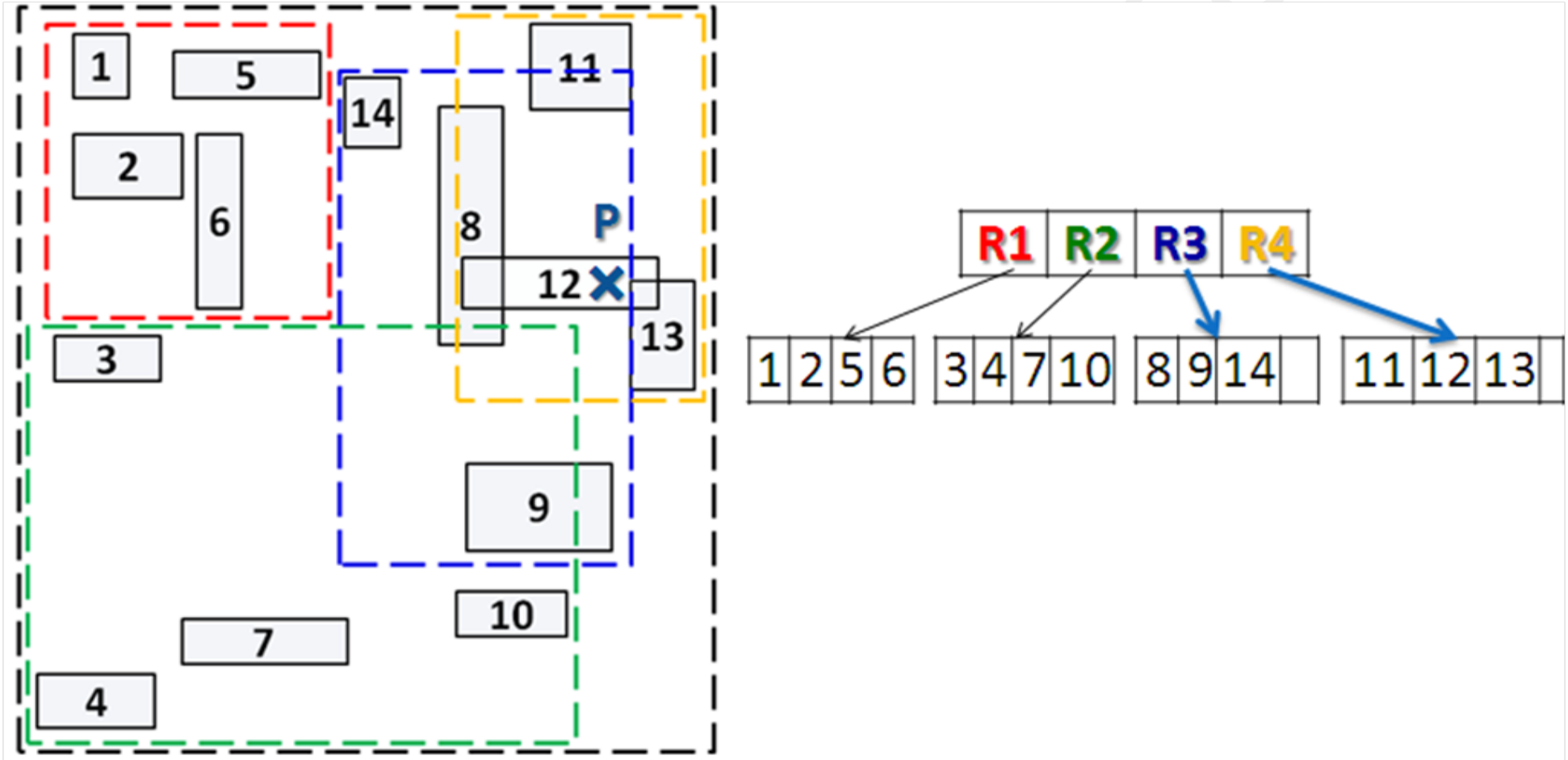
Arbre R d'ordre (2,4)





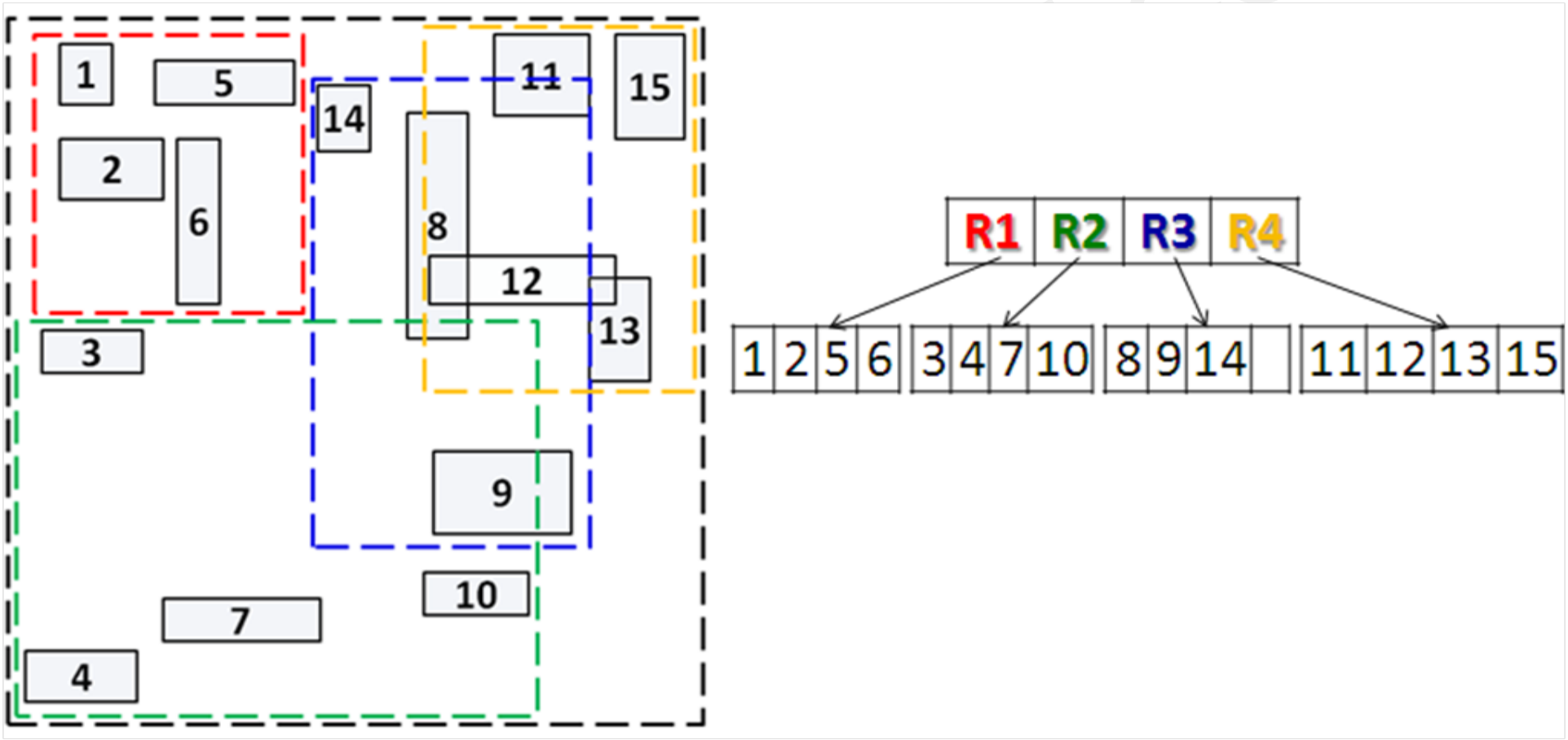
Arbre R - Vue d'ensemble

Pointé



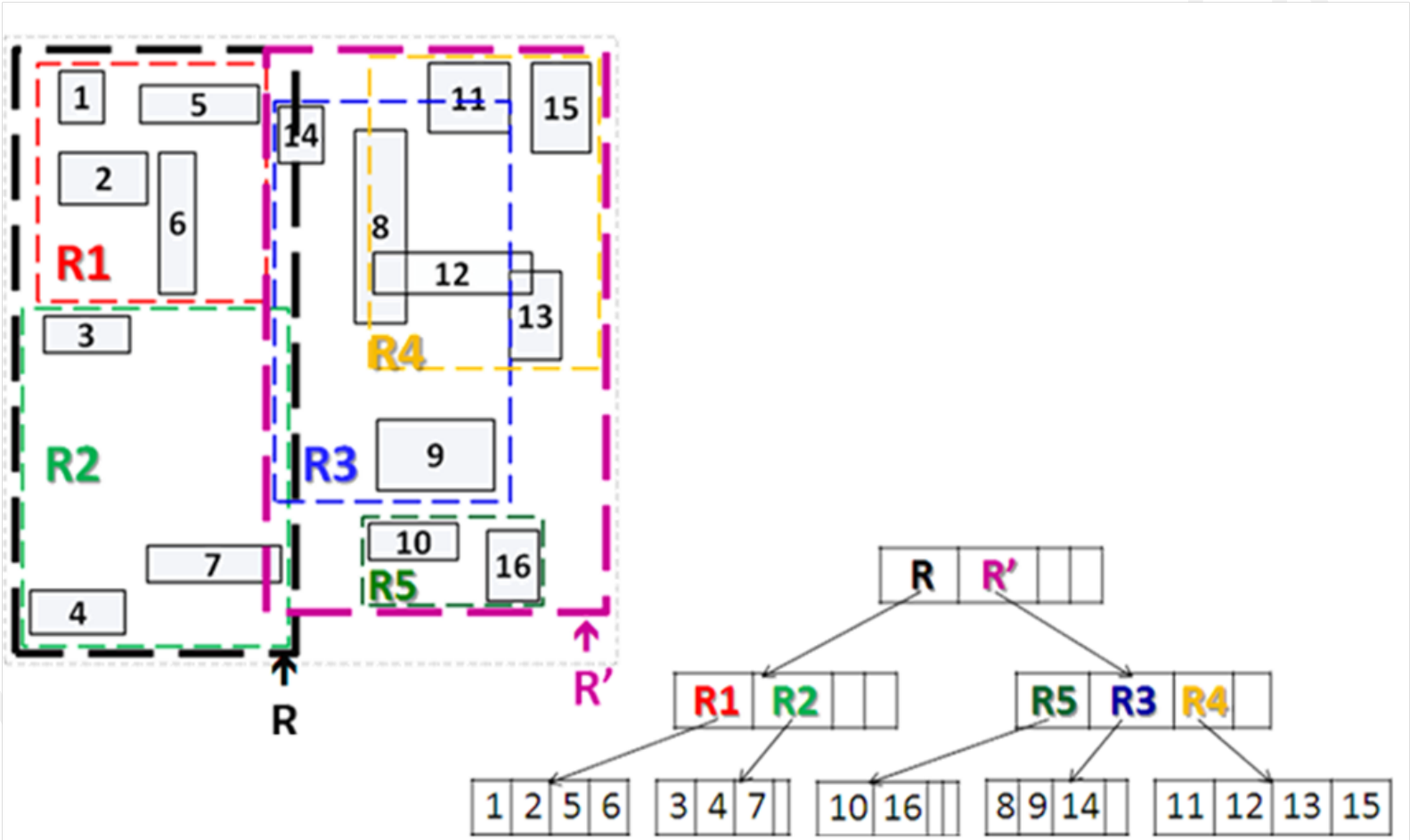
Arbre R - Insertion sans éclatement

Insertion sans éclatement : agrandissement si nécessaire de R_4 et du REM de la racine.



Arbre R - Insertion avec éclatement

Insertion avec éclatement



Arbre R - Définition

Un Arbre R (R-Tree) d'ordre (m, M) est un arbre équilibré, tel que :

- Chaque nœud est associé à une partie rectangulaire de l'espace de sorte que :
  - Le rectangle associé à un nœud feuille est le REM des REMs des objets spatiaux que le nœud référence.
  - Le rectangle associé à un nœud interne est le REM des REMs de ses fils.
  - Le rectangle associé à la racine est le REM de l'espace indexé tout entier.
- Chaque entrée est une paire (rem, adr), où :
  - Noeud interne : rem est le REM associé à un nœud fils d'adresse adr.
  - Noeud feuille : rem est le REM d'un objet spatial d'adresse adr.
- Chaque nœud contient au plus M entrées et au moins $m \leq \frac{M}{2}$ entrées (excepté la racine qui peut contenir 0 ou 2 ou m entrées).

j.khaleq@gmail.com

Arbre R - Recherche

Recherche des objets ayant une intersection non vide avec un objet requête ayant pour REM R :

1. Partir de la racine.
2. Tester l'intersection de R avec chacune des entrées du nœud courant.
3. Chaque fois que l'intersection n'est pas vide, suivre l'adresse vers le nœud fils correspondant.
4. Comme les entrées d'un nœud se chevauchent, il est possible de suivre plusieurs adresses à la fois (même pour une requête de type pointé).

j.khaled@gmail.com

Arbre R - Insertion

Insertion dans un arbre R :

- Même principe que pour un arbre B+ :
  1. Parcours de l'arbre pour localiser la feuille où (le REM de) l'objet est à insérer.
  2. Insertion de l'entrée dans la feuille et éclatement si la feuille déborde.
  3. Propagation de l'information sur l'insertion si nécessaire (dans quels cas est-ce nécessaire?).
- Étapes supplémentaires pour un arbre R :
  - À chaque niveau (en cas de chevauchement), choisir l'entrée dont le REM nécessite le minimum d'agrandissement.
  - En cas d'éclatement :
    - Minimiser le chevauchement ⇒ éviter (autant que possible) de suivre plusieurs chemins lors d'une recherche.
    - Plusieurs stratégies alternatives : linéaire, quadratique, exhaustive, etc.

j.khaled@gmail.com

Arbre R - Insertion

Stratégie linéaire :

- Objectif (pas toujours atteint) : choisir parmi les alternatives de répartition des entrées entre les deux nouveaux nœuds, celle minimisant l'espace total occupé par les deux REM correspondants.

- Principe :
  1. Choisir les deux REM les plus distants comme graines pour les nouveaux nœuds.
  2. Affecter chaque autre REM au nœud dont le REM nécessite le minimum d'agrandissement.

j.khaled@gmail.com

Arbre R - Remarques

- L'arbre R est l'une des structures les plus populaires pour l'indexation des données multi-dimensionnelles.
- Les expérimentations montrent qu'il fonctionne assez bien lorsque la dimensionalité est inférieure à 20. Au-delà, le chevauchement des REM (hypercubes) dégrade sérieusement les performances.
- L'ordre des insertions des REM influe sur les performances.
- L'arbre R a inspiré plusieurs autres méthodes : R+tree, R*tree, X-tree, m-tree, packed R-tree, etc.
- Implémenté dans la plupart des systèmes GPS et des SGBDs modernes : Oracle, PostgreSQL, MySQL, etc.

i.khaled@gmail.com

Section 2 - Recherche Full-Text

j.khaled@gmail.com

Plan

Chapitre 2 - Recherche Full-Text

j.khaled@gmail.com

Introduction à la Recherche Full-Text

- La recherche full-text (recherche d'information, IR) permet de retrouver des documents pertinents dans de grandes collections de textes.
- Utilisée dans les moteurs de recherche, les bases de données, et les systèmes d'indexation documentaire.
- Elle est particulièrement utile pour la gestion de grandes quantités de données non structurées.
- L'objectif est de classer les documents selon leur pertinence par rapport à une requête utilisateur.

j.khaled@gmail.com

Vue d'ensemble de la Recherche d'Information (RI)

- RI se concentre sur la récupération d'informations pertinentes en fonction d'un besoin.
- Les documents sont généralement peu structurés (texte libre), par opposition à des bases de données traditionnelles où les données sont structurées.
- **Recherche structurée** : utilise des bases de données relationnelles où les données sont organisées et bien définies. Les requêtes SQL retournent des réponses précises.
- **Recherche full-text** : s'applique aux documents non structurés où l'objectif est de trouver des textes ou des informations spécifiques à partir de mots-clés.
- Les recherches full-text se basent souvent sur des algorithmes comme TF-IDF pour déterminer la pertinence des résultats.

j.khlaa@gmail.com

Fonctionnement de la Recherche Full-Text

- Analyse des documents pour extraire des mots-clés ou des termes importants.
- Ces termes sont indexés pour permettre une recherche efficace.
- Lorsqu'une requête est soumise, les termes de la requête sont comparés aux termes indexés.
- Un algorithme de classement, comme TF-IDF ou la similarité cosinus, est utilisé pour classer les documents par pertinence.

j.khaled@gmail.com

Plan

Chapitre 2 - Recherche Full-Text

j.khaled@gmail.com

Concepts clés

- **Requête** : soumise par l'utilisateur, elle peut être des mots-clés, une phrase, ou même un document complet.
- **Document** : un texte ou fichier à rechercher dans une collection, potentiellement indexé avec des métadonnées.
- **Classement** : une liste ordonnée des documents en fonction de leur pertinence par rapport à la requête.
- **Indexation** : processus de transformation des documents en une structure de données facilitant leur recherche rapide.

j.khaled@gmail.com

Précision et Rappel

- Pertinence de la recherche : Précision et Rappel.
- **Précision (Precision)** : proportion de documents pertinents parmi les documents retournés.

$$\text{Précision} = \frac{\text{Documents pertinents retrouvés}}{\text{Documents retrouvés}}$$

- **Rappel (Recall)** : proportion de documents pertinents retrouvés parmi tous les documents pertinents.

$$\text{Rappel} = \frac{\text{Documents pertinents retrouvés}}{\text{Total de documents pertinents}}$$

- Un bon système de recherche doit équilibrer précision et rappel, mais il est souvent difficile de maximiser les deux simultanément.

i.khaled@gmail.com

Qualité de la recherche

Exercice : Donnez la précision et le rappel dans les cas suivants :

1. Un utilisateur fait une recherche dont il sait qu'elle devrait retourner 10 documents. Le système retourne 30 documents, dont 5 seulement sont parmi ceux que l'utilisateur attend.
2. Un utilisateur cherche un document unique et il est parmi les 30 documents retournés par le système.

j.khaled@gmail.com

Indexation

Indexation : Comment trouver rapidement (*i.e.* sans recourir à un balayage séquentiel) les documents pertinents? Comment comparer rapidement 2 documents?

Un premier outil : la **matrice d'incidence**, appelée également **index bitmap**

	Antoine et Cléopâtre	Jules César	La tempête	Hamlet	Othello	Macbeth
Antoine	1	1	0	0	0	1
Brutus	1	1	0	1	0	0
César	1	1	0	1	1	1
Calpurnia	0	1	0	0	0	0
Cléopâtre	1	0	0	0	0	0
mercy	1	0	1	1	1	1
worser	1	0	1	1	1	0

Qu'est-ce qu'un Index Inversé ?

- Un **index inversé** est une structure de données qui associe chaque terme d'une collection de documents aux documents dans lesquels il apparaît.

- Contrairement à une matrice d'incidence qui associe les termes à des documents via une représentation binaire, l'index inversé stocke directement les identifiants des documents contenant chaque terme.

- Utilisé dans tous les moteurs de recherche modernes, car il permet de retrouver rapidement les documents pertinents pour un terme donné.

- Structure : chaque terme est associé à une liste de documents (appelée *posting list*).

j.khalid@gmail.com

Comment fonctionne un Index Inversé ?

- Lors de l'indexation, les documents sont analysés pour extraire les termes, et chaque terme est associé aux documents où il apparaît.
- Exemple : Pour une requête contenant plusieurs termes ("arbre" et "hachage"), l'index inversé permet de retrouver rapidement les documents où les deux termes sont présents.
- Cela permet de répondre aux requêtes booléennes et de faire des recherches plus complexes en combinant plusieurs listes de postings.
- **Index inversé : permet de retrouver rapidement les documents, mais pas de les classer**!

jkhalod@gmail.com

Plan

Chapitre 2 - Recherche Full-Text

j.khaled@gmail.com

TF-IDF: Introduction

- **TF-IDF (Term Frequency - Inverse Document Frequency)** est une technique clé en recherche d'information pour pondérer l'importance des termes dans un document en fonction de leur fréquence dans l'ensemble des documents.

- Les termes fréquents dans un document spécifique et rares dans la collection globale sont considérés comme plus importants pour la recherche.

- Cet algorithme aide à écarter les termes communs (ex : "le", "de") qui n'apportent pas de valeur ajoutée à la recherche.

j.khaled@gmail.com

Term Frequency (TF)

- Le **TF (Term Frequency)** mesure la fréquence d'apparition d'un terme donné dans un document.

$$TF(t, d) = \frac{n_{t,d}}{\sum_k n_{k,d}}$$

où $n_{t,d}$ est le nombre d'occurrences du terme t dans le document d, et $\sum_k n_{k,d}$ est le nombre total de termes dans d.

- Un terme qui apparaît fréquemment dans un document a un TF élevé.

j.khalid@gmail.com

Inverse Document Frequency (IDF)

- L'**IDF (Inverse Document Frequency)** mesure la rareté d'un terme dans une collection de documents.

$$IDF(t, D) = \log\left(\frac{|D|}{|\{d \in D : t \in d\}|}\right)$$

où $|D|$ est le nombre total de documents, et $\{d \in D : t \in d\}$ est le nombre de documents contenant le terme t.

- Plus un terme est rare dans la collection, plus sa valeur IDF est élevée, ce qui signifie qu'il est plus important pour différencier les documents.

j.khaled@gmail.com

Pondération TF-IDF

- La pondération TF-IDF combine le TF et l'IDF pour évaluer l'importance globale d'un terme dans un document.

$$\textit{TF-IDF}(t, d, D) = \textit{TF}(t, d) \times \textit{IDF}(t, D)$$

- Cette formule favorise les termes qui apparaissent fréquemment dans un document tout en étant rares dans l'ensemble des documents, les rendant plus importants pour la recherche.

j.khaled@gmail.com

Plan

Chapitre 2 - Recherche Full-Text

j.khaled@gmail.com

Quantification de la similarité

La notion de pertinence ou de similarité peut être traduite en termes quantitatifs.

L'approche classique consiste à :

- Définir un espace métrique E associé à une fonction de distance m_E;
- Déterminer une fonction f qui projette l'espace des documents dans E;
- La fonction f associe à chaque document un objet de cet espace, souvent sous forme de vecteur appelé **descripteur** ou **vecteur de caractéristiques**;
- La fonction f s'applique également à la requête q, qui est traitée comme un document;
- Enfin, la pertinence ou la similarité entre un document d et une requête q est évaluée comme l'inverse de la distance entre $f(d)$ et $f(q)$: $sim(d, q) = \frac{1}{m_E(f(d), f(q))}$

La notion de similarité est également désignée sous le terme **score**.

j.khaled@gmail.com

Approche simpliste

Première approche (simpliste)

- Considérons le vocabulaire $V = \{t_1, t_2, \ldots, t_n\}$, qui inclut tous les termes présents dans les documents.
- La fonction $f(d) = v$ génère pour chaque document d un descripteur v. Ce dernier est un vecteur binaire, où chaque coordonnée i est égale à 1 si le terme t_i apparaît dans d, et 0 sinon.
- On définit E comme un espace vectoriel de vecteurs de dimension n, avec des coordonnées binaires $E = \{0, 1\}^n$.
- La distance euclidienne entre deux vecteurs v_1 et v_2 est donnée par :

$$m_E(v_1, v_2) = \sqrt{(v_1^1 - v_2^1)^2 + (v_1^2 - v_2^2)^2 + \ldots (v_1^n - v_2^n)^2}$$

- La similarité est définie ainsi :

$$sim(v_1, v_2) = \begin{cases} \infty & \text{si } v_1^i = v_2^i \ \forall i \\ \frac{1}{m_E(v_1, v_2)} & \text{sinon} \end{cases}$$

j.khaled@gmail.com

Exercice

- Considérons les documents suivants :

  d_1: Le loup rôde près de la bergerie.

  d_2: Les moutons broutent paisiblement dans le pre.

  d_3: Un loup a attrape un mouton dans la bergerie, mais le chien est intervenu.

  d_4: Trois loups s'approchent de la bergerie.

- Le vocabulaire utilisé est : "loup", "mouton", "bergerie", "pré", "chien". L'espace vectoriel correspondant est $E = \{0, 1\}^5$

1. Construisez la fonction qui associe chaque document à un vecteur dans $\{0, 1\}^5$, en le représentant sous forme de matrice d'incidence.
2. Calculez le score pour chaque document en fonction des recherches suivantes et déterminez le classement :

   q_1 : "chien"

   q_2 : "chien et bergerie"

j.khaled@gmail.com

Pourquoi la Distance Euclidienne ne convient pas pour les Textes

- La distance euclidienne mesure la différence absolue entre deux vecteurs dans un espace métrique.
- **Problème** : elle est sensible à la longueur des documents.
  - Deux documents de longueur différente peuvent avoir une grande distance euclidienne, même s'ils partagent des termes similaires.
  - Les variations de longueur amplifient les différences, faussant la mesure de similarité.
- La similarité cosinus contourne ce problème en se basant sur l'angle entre les vecteurs, ignorant la magnitude.

jkhaled@gmail.com

Similarité Cosinus: Introduction

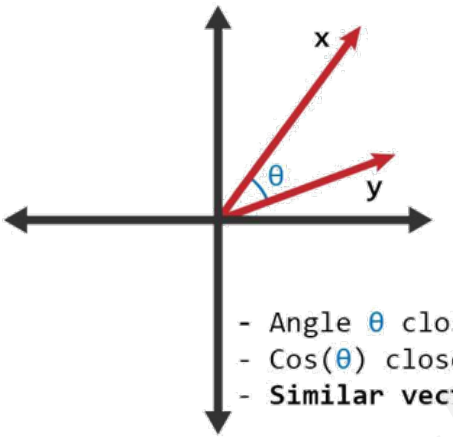
- La similarité cosinus est une mesure utilisée pour calculer la similarité entre deux vecteurs dans un espace vectoriel, comme ceux obtenus après le calcul du TF-IDF pour des documents.

- Elle permet d'évaluer la proximité entre un document et une requête en mesurant l'angle entre les vecteurs de termes.

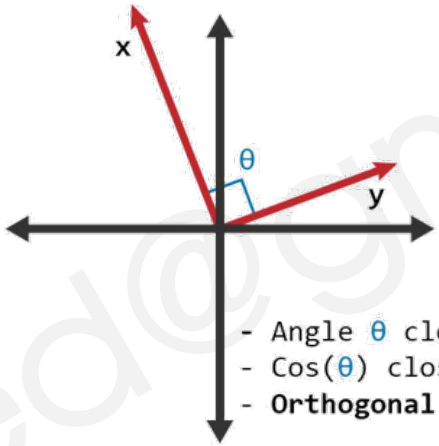
- Une valeur de cosinus proche de 1 signifie que les vecteurs sont similaires (petit angle), tandis qu'une valeur proche de 0 indique peu de similarité.

j.khaled@gmail.com

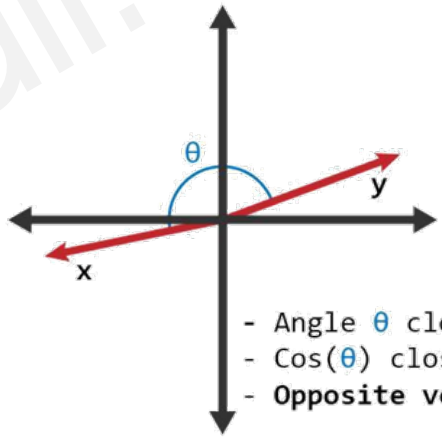
Classement par Cosinus



- Angle θ close to 0
- Cos(θ) close to 1
- **Similar vectors**



- Angle θ close to 90
- Cos(θ) close to 0
- **Orthogonal vectors**



- Angle θ close to 180
- Cos(θ) close to -1
- **Opposite vectors**

j.khaled@gmail.com

Similarité cosinus : définition et explication intuitive

Formule de la similarité cosinus :

$$\text{sim}_{\cos}(A, B) = \frac{A \cdot B}{||A||||B||}$$

Explication :

- La similarité cosinus mesure l'angle entre deux vecteurs A et B dans un espace vectoriel.
- Elle est souvent utilisée pour comparer des documents représentés sous forme de vecteurs de termes (comme le modèle TF-IDF).
- Intuitivement, plus l'angle entre les vecteurs est petit, plus les documents sont similaires. Si deux vecteurs pointent dans la même direction, leur similarité cosinus sera proche de 1 (parfaite similarité).
- Si l'angle est de 90° (les vecteurs sont orthogonaux), la similarité cosinus est égale à 0, indiquant qu'ils sont complètement différents.
- Cette méthode est particulièrement utile lorsque l'importance est accordée aux directions relatives (la structure du contenu) plutôt qu'à la magnitude (la longueur du texte ou la fréquence brute des termes).

j.kroliq@gmail.com

Exemple de calcul de la similarité cosinus

Vecteurs :

$$A = (2, 3), \quad B = (1, 4)$$

Étape 1 : Calcul de la norme des vecteurs

$$\|A\| = \sqrt{2^2 + 3^2} = \sqrt{13}, \quad \|B\| = \sqrt{1^2 + 4^2} = \sqrt{17}$$

Étape 2 : Calcul du produit scalaire

$$A \cdot B = 2 \times 1 + 3 \times 4 = 2 + 12 = 14$$

Étape 3 : Calcul de la similarité cosinus

$$\text{sim}_{\cos}(A, B) = \frac{A \cdot B}{\|A\|\|B\|} = \frac{14}{\sqrt{13} \times \sqrt{17}} \approx \frac{14}{14.93} \approx 0.94$$

Conclusion : Les vecteurs A et B ont une similarité cosinus élevée (0.94), ce qui signifie qu'ils sont très similaires.

j.khaled@gmail.com

Exercice : Calcul de TF-IDF et de la similarité cosinus

Documents :

- d_1 : "Le chien garde la bergerie"
- d_2 : "Le loup attaque la bergerie"

Étape 1 : Calculer le TF-IDF pour chaque document et chaque terme.

- Vocabulaire : "chien", "garde", "bergerie", "loup", "attaque"
- Term Frequency (TF) :
  $\text{TF}_{\text{chien},d_1} = 1$, $\text{TF}_{\text{bergerie},d_1} = 1$, etc.
- Inverse Document Frequency (IDF) :
  $\text{IDF}_{\text{bergerie}} = \log \frac{2}{2} = 0$, $\text{IDF}_{\text{loup}} = \log \frac{2}{1} = \log 2$, etc.

Étape 2 : Représenter chaque document comme un vecteur TF-IDF.

$$d_1 = (0, \text{TF-IDF}_{\text{bergerie}}, \dots), \quad d_2 = (0, \text{TF-IDF}_{\text{loup}}, \dots)$$

Étape 3 : Calculer la similarité cosinus entre les vecteurs TF-IDF.

$$\text{sim}_{\cos}(d_1, d_2) = \frac{d_1 \cdot d_2}{\|d_1\|\|d_2\|}$$

Question : Calculez la similarité cosinus entre d_1 et d_2.

j.khaled@gmail.com

Plan

Chapitre 2 - Recherche Full-Text

j.khaled@gmail.com

Introduction à PageRank

- *PageRank* est un algorithme utilisé par Google pour classer les pages web dans ses résultats de recherche.
- Il mesure l'importance d'une page en fonction du nombre et de la qualité des liens entrants (backlinks).
- Plus une page est liée par d'autres pages importantes, plus son score PageRank sera élevé.
- L'algorithme est basé sur la théorie des graphes et utilise des méthodes de calcul matriciel.

j.khaled@gmail.com

Le Modèle de PageRank

- Le web est modélisé comme un graphe dirigé, où les *nœuds* représentent des pages web et les *arcs* représentent les liens hypertexte.

- L'algorithme PageRank attribue une *probabilité* à chaque page qui représente la probabilité qu'un utilisateur visite cette page à un moment donné.

- Cette probabilité est mise à jour de manière itérative en tenant compte des liens entrants et du facteur de *saut aléatoire*.

j.khaled@gmail.com

La Formule de PageRank

- La formule de base pour calculer le PageRank d'une page P est :

$$PR(P) = (1 - d) + d \times \sum_{i=1}^{N} \frac{PR(P_i)}{L(P_i)}$$

- Où :
  - d est le *facteur d'amortissement* (souvent 0.85).
  - P_i sont les pages pointant vers P.
  - $L(P_i)$ est le nombre de liens sortants de la page P_i.
- Le PageRank est donc une somme pondérée des PageRanks des pages qui pointent vers P, ajustée par le facteur d.

j.khaled@gmail.com

Explication du Facteur d'Amortissement

- Le *facteur d'amortissement d* représente la probabilité qu'un utilisateur suive un lien hypertexte sur une page donnée.
- Avec une probabilité $1 - d$, l'utilisateur *saute* vers une autre page aléatoirement.
- Ce facteur empêche certaines pages d'accumuler trop d'importance si elles sont fortement liées.
- Typiquement, $d = 0.85$ est utilisé dans la majorité des implémentations.

j.khaled@gmail.com

Calcul Itératif de PageRank

- Le PageRank est calculé de manière *itérative* en partant d'une distribution initiale de probabilités.
- À chaque itération, le PageRank de chaque page est mis à jour en fonction des PageRanks des pages qui pointent vers elle.
- Le calcul s'arrête lorsque les changements dans les PageRanks sont suffisamment petits (critère de convergence).
- Exemple :

$$PR^{(k+1)}(P) = (1-d) + d \times \sum_{i=1}^{N} \frac{PR^{(k)}(P_i)}{L(P_i)}$$

j.khaled@gmail.com

Exemple Simple de Calcul de PageRank

- Considérons un graphe avec 3 pages : A, B et C.
- Les liens sont :
  - A pointe vers B et C.
  - B pointe vers C.
  - C pointe vers A.
- Après plusieurs itérations, les scores convergeront vers les valeurs finales.

j.khaled@gmail.com

Limites de l'Algorithme PageRank

- PageRank est efficace pour les petits graphes, mais devient coûteux en calcul pour des graphes très grands (comme le web).
- Il ne prend pas en compte le *contenu* des pages, seulement la structure des liens.
- Peut être manipulé par des techniques de *spamdexing*, où des sites web créent artificiellement des liens pour augmenter leur PageRank.

j.khaled@gmail.com

Plan

Chapitre 2 - Recherche Full-Text

j.khaled@gmail.com

Solr, quésaco?

- Solr est une plateforme open-source de recherche full-text basée sur Apache Lucene, similaire à ElasticSearch.
- Elle est conçue pour indexer et rechercher efficacement des données non structurées, comme des textes.
- Solr supporte des requêtes complexes, des recherches facettées, et fournit une interface RESTful pour interagir avec les données.

j.khaled@gmail.com

Solr, quésaco?

Solr v.s. Relational Database		
Lucene	Solr	Relational DB
Text Search	Fast and sophisticated	Minimal and slow
Features	Few, targeted to text search	Many
Deployment Complexity	Medium	Medium
Administration Tools	Minimal open source projects	Many open source & commercial
Monitoring Tools	Weak	Very Strong
Scaling Tools	Automated, medium scale	Large scale
Support Availability	Weak	Strong
Schema Flexibility	Must in general rebuild	Changes immediately visible
Indexing Speed	Slow	Faster and adjustable
Query Speed	Text search is fast & predictable	Very dependent on design & use case
Row Addition/Extraction Speed	Slow	Fast
Partial Record Modification	No	Yes
Time to visibility after addition	Slow	Immediate
Access to internal data structures	High	None
Technical knowledge required	Java (minimal), web server deployment, IT	SQL, DB-specific factors, IT
Regular maintenance tasks		

Installation

- Pré-requis : une version récente de Java.
- Téléchargez la dernière version stable de SolR : https://downloads.apache.org/solr/solr/
- Décompressez l'archive dans un répertoire (appelé ici `soldir`).
- Ajoutez `soldir/bin` à la variable système `PATH`.
- Pour démarrer en mode standalone (single node) :

```
1    solr start
```

- Pour démarrer en mode cluster :

```
1    solr start -c
```

- Accédez à l'interface d'administration à l'adresse : http://localhost:8983/solr

j.khalid@gmail.com

Interface REST pour l'insertion de documents

- Utilitaire Java fourni par Solr :

```
java -Dc=nomCore -jar post.jar *.json
```

- Console d'administration Solr (nécessite un champ version).

- Interface REST (via HTTP) en utilisant curl :

```
curl http://localhost:8983/solr/moviesCore/update/json?commit=true \
    --data-binary @/moviesSolr.json -H "Content-type:application/json"
```

- **DataImportHandler** pour les sources de données externes (bases de données, Hadoop, etc.).

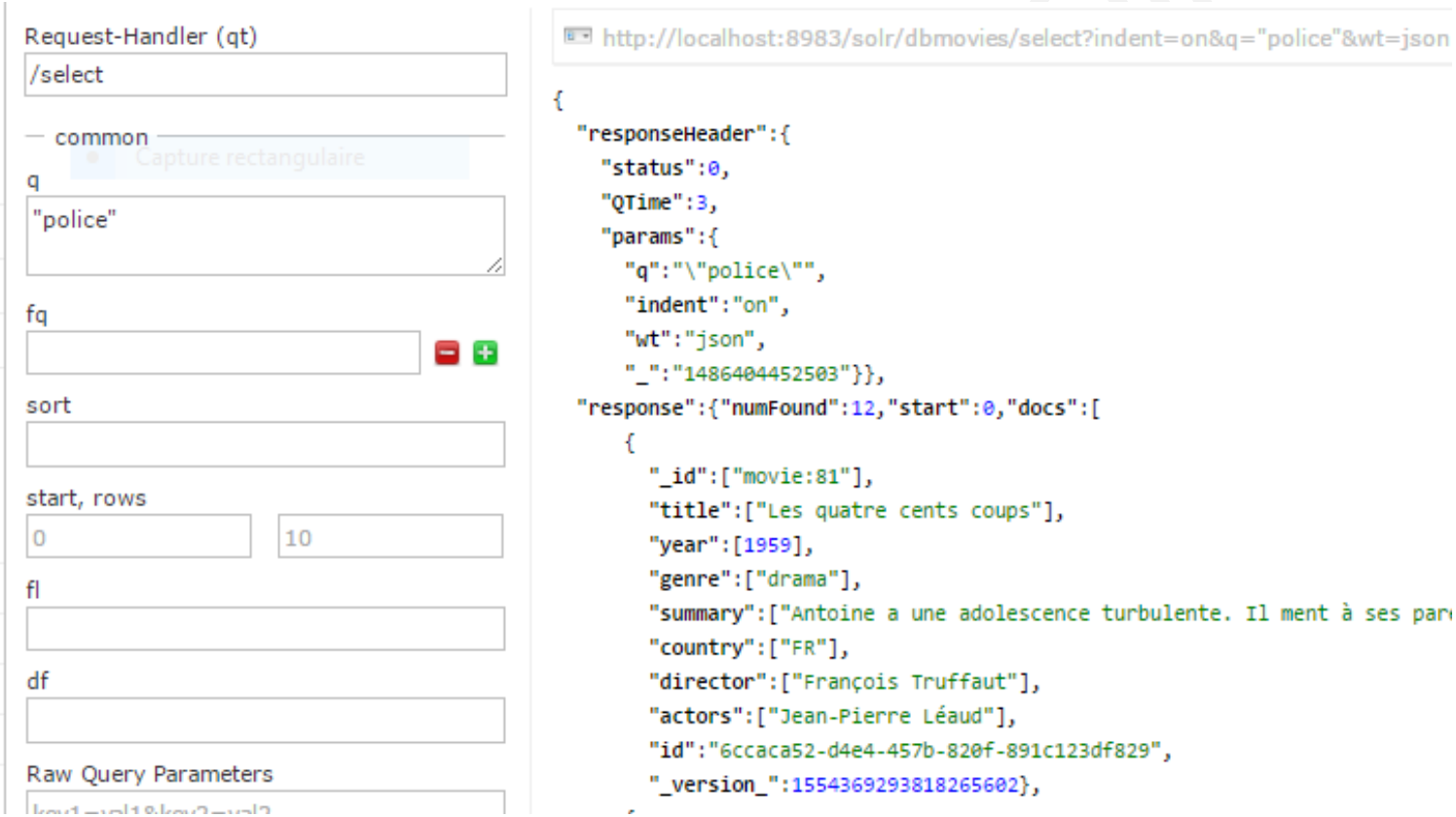
jkhadd@gmail.com

Recherche dans SolR

- Requête dans un navigateur ou via curl :

```
1    http://localhost:8983/solr/nomCore/select?q=summary:police
```

- Via la console d'administration de SolR :



Recherche avec débogage

- Solr permet d'obtenir des détails sur l'exécution d'une requête avec l'option `debugQuery`.

```
http://localhost:8983/solr/loups/select?debugQuery=on&indent=on&q=Moutons%20loups
    &wt=json
```


```
  "debug":{
    "rawquerystring":"moutons loups",
    "querystring":"moutons loups",
    "parsedquery":"_text_:moutons _text_:loups",
    "parsedquery_toString":"_text_:moutons _text_:loups",
    "explain":{
      "B":"\n1.5262812 = sum of:\n  1.5262812 = weight(_text_:moutons in 1) [SchemaSimilarity], result
idf(docFreq=3, docCount=12)\n      1.1631589 = tfNorm, computed from:\n        1.0 = termFreq=1.0\n
fieldLength\n",
      "E":"\n1.2979885 = sum of:\n  1.2979885 = weight(_text_:loups in 4) [SchemaSimilarity], result of
docCount=12)\n      0.98918 = tfNorm, computed from:\n        1.0 = termFreq=1.0\n        1.2 = paramet
      "H":"\n1.2979885 = sum of:\n  1.2979885 = weight(_text_:loups in 7) [SchemaSimilarity], result of
docCount=12)\n      0.98918 = tfNorm, computed from:\n        1.0 = termFreq=1.0\n        1.2 = paramet
      "F":"\n1.1515269 = sum of:\n  1.1515269 = weight(_text_:moutons in 5) [SchemaSimilarity], result
idf(docFreq=3, docCount=12)\n      0.87756354 = tfNorm, computed from:\n        1.0 = termFreq=1.0\n
20.897959 = fieldLength\n"
```

Schéma SolR: à quoi ça sert?

- Détermine les champs du document à indexer et la manière dont ils doivent être analysés.
- Supporte plusieurs types de champs (entiers, chaînes de caractères, texte, etc.).
- Permet des transformations et calculs sur les valeurs des champs.

```
1    <field name="summary" type="text_fr" indexed="true" required="false
     " stored="false"/>
```

```
1    <fieldType name="text_fr" class="solr.TextField">
2        <analyzer>
3            <tokenizer class="solr.StandardTokenizerFactory"/>
4            <filter class="solr.StopFilterFactory" ignoreCase="true" words="lang/
     stopwords_fr.txt"/>
5            <filter class="solr.SynonymFilterFactory" synonyms="synonyms.txt"
     ignoreCase="true" expand="false"/>
6            <filter class="solr.LowerCaseFilterFactory"/>
7            <filter class="solr.ElisionFilterFactory" articles="lang/
     contractions_fr.txt"/>
8            <filter class="solr.FrenchLightStemFilterFactory"/>
9        </analyzer>
```

Schéma SolR : à quuoi ça sert?

- Un champ `copyField` permet de copier la valeur d'un champ source vers un autre champ.

- Cela permet de combiner plusieurs champs dans un seul champ de recherche full-text.

```
1 <copyField source="title" dest="filmInfo"/>
2 <copyField source="description" dest="filmInfo"/>
```

Ce champ permet de rechercher à la fois dans `title` et `description`.

jkhadd@gmail.com

Exemple de fichier `schema.xml`

```xml
<?xml version="1.0" encoding="UTF-8"?>
<schema name="example" version="1.5">
    <uniqueKey>_id</uniqueKey>
    <fieldType name="int" class="solr.TrieIntField"/>
    <fieldType name="long" class="solr.TrieLongField"/>
    <fieldType name="string" class="solr.StrField"/>
    <fieldType name="text_fr" class="solr.TextField">
        <analyzer>
            <tokenizer class="solr.StandardTokenizerFactory"/>
            <filter class="solr.StopFilterFactory" ignoreCase="true" words="lang/stopwords_fr.txt"/>
            <filter class="solr.SynonymFilterFactory" synonyms="synonyms.txt" ignoreCase="true" expand="false"/>
            <filter class="solr.LowerCaseFilterFactory"/>
            <filter class="solr.ElisionFilterFactory" articles="lang/contractions_fr.txt"/>
            <filter class="solr.FrenchLightStemFilterFactory"/>
        </analyzer>
    </fieldType>
    <field name="_version_" type="long" indexed="true" stored="true"/>
    <field name="id" type="string" indexed="true" required="true" stored="true"/>
    <field name="country" type="string" indexed="true" required="true" stored="true"/>
    <field name="filmInfo" type="text_fr" multiValued="true" indexed="true" stored="true"/>
    <field name="summary" type="text_fr" indexed="true" required="false" stored="false"/>
    <field name="title" type="string" indexed="true" required="true" stored="true"/>
    <field name="year" type="long" indexed="true" stored="true"/>
    <copyField source="summary" dest="filmInfo"/>
    <copyField source="title" dest="filmInfo"/>
</schema>
```

Modification du schéma avec l'API Schema

- Le schéma des documents indexés peut être :
  - Défini manuellement par l'utilisateur via `schema.xml`.
  - Généré automatiquement par Solr (*managed-schema*).
- Dans les versions récentes de SolR, il est recommandé d'utiliser l'API Schema pour modifier le schéma de manière dynamique.

```
curl -X POST -H "Content-type:application/json"
     --data-binary '{"add-field":{"name":"nouveau", "type":"text_general", "
     stored":true}}' http://localhost:8983/solr/moviesCore/schema
```

j.khaled@gmail.com

SolR et Big Data : SolrCloud

- SolR : pas conçu initialement comme un système distribué
- Cet aspect n'est véritablement apparu qu'avec la version 4. Cela se ressent dans la conception, un peu moins simple et intégrée que celle de MongoDB.
- Une **grappe de noeuds Solr** (SolrCloud) est constituée de :
  - **serveurs Solr** individuels, tout à fait conforme à ce que nous avons déjà vu; et
  - d'un **serveur ZooKeeper** souvent **répliqué** (typiquement 3), prenant en charge les tâches liées à la distribution.
- Il est possible d'utiliser la version embedded de ZooKeeper ou de distribuer solr avec une grappe zookeeper externe.
- Cette deuxième option est à privilégier dans un environnement de production.

l.khaled@gmail.com

SolR et Big Data : SolrCloud

- Zookeeper : système maintenu par la fondation Apache (initialement sous projet de Hadoop)
- Objectif : fournir des services génériques dans un système distribué.
- Ces services sont, pour l'essentiel, celles d'un noeud-maître assurant la coordination des noeuds-esclaves, soit, concrètement :
  - **gestion** centralisée **de la configuration** (*e.g.* schéma SolR) ;
  - **gestion des échanges de messages** (avec des files d'attente) ;
  - **contrôle de la disponibilité** des serveurs ;
  - **gestion de la reprise sur panne**, et notamment élection d'un nouveau maître ;
  - **distribution des tâches** (*load balancing*).

j.knafeg@gmail.com

ZooKeeper

- Grappe gérée par ZooKeeper : 2 types de noeuds , le leader et les followers

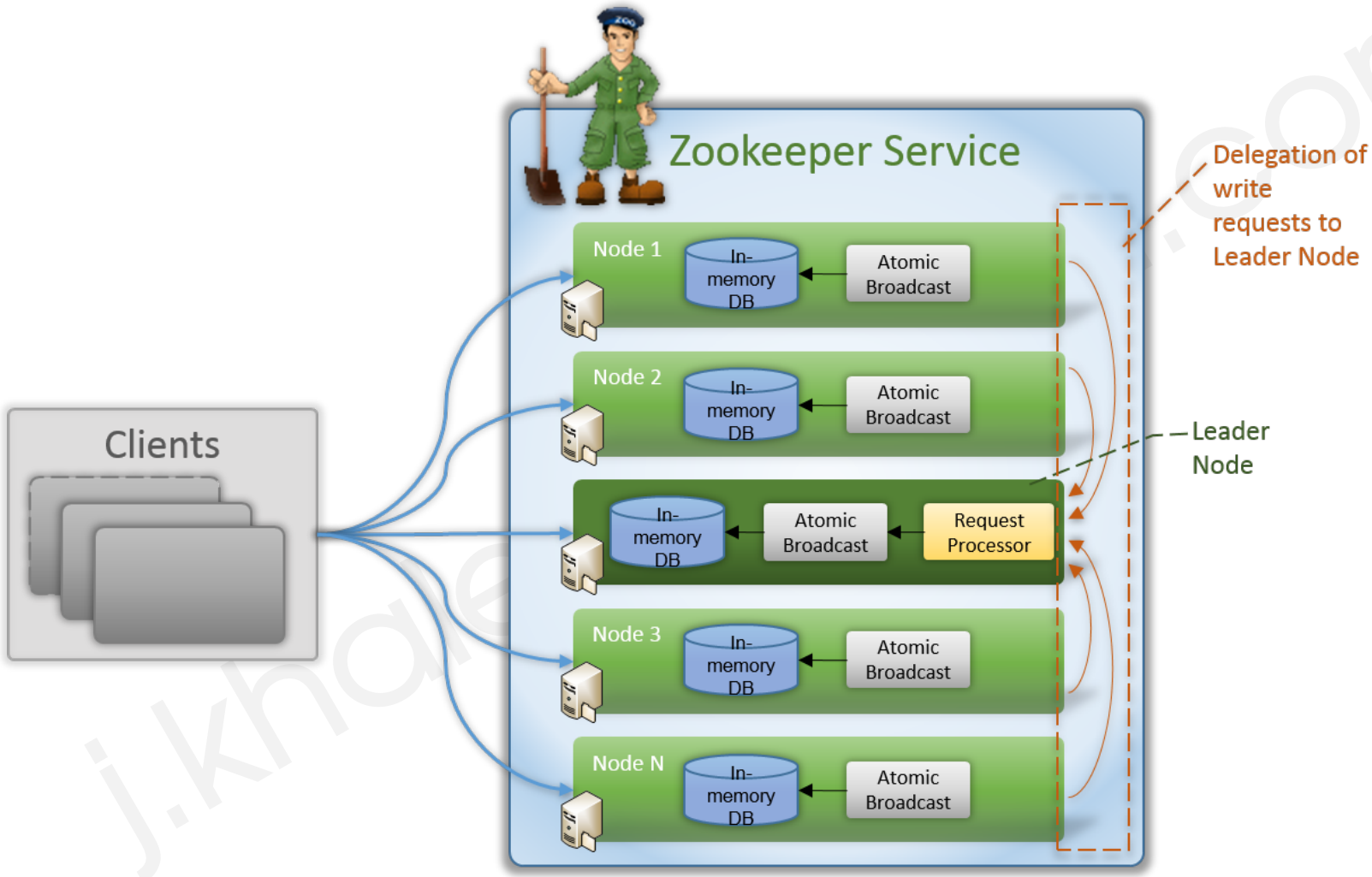
- **Leader Node** (Master)
  - **Les écritures se font uniquement sur le noeud Leader**.
  - Choisi par **élection** (PAXOS). Il est préférable que le **nombre de noeuds soit impair** (Pourquoi?)

- **Follower Nodes** (slaves)
  - répondent aux **read requests** mais **transfèrent les write requests** au leader

  - reçoivent et **appliquent les messages de synchronisation envoyés par le Leader**

j.khaled@gmail.com

ZooKeeper


Zookeeper Service
Clients
Node 1
In-memory DB
Atomic Broadcast
Node 2
In-memory DB
Atomic Broadcast
In-memory DB
Atomic Broadcast
Request Processor
Node 3
In-memory DB
Atomic Broadcast
Node N
In-memory DB
Atomic Broadcast
Delegation of write requests to Leader Node
Leader Node

Session illustrative

1. Lancement d'un serveur zookeeper.

1. Téléchargez zookeeper http://zookeeper.apache.org/releases.html.
2. Décompressez zookeeper dans un répertoire que nous appelons dans la suite <zooDir>.
3. Ajoutez une variable d'environnement `ZOOKEEPER_HOME` pointant vers <zooDir>
4. Ajoutez <zooDir>/bin à la variable d'environnement `PATH`
5. Editez le fichier <zooDir>/conf/zoo_samples.cfg. Dans la ligne *dataDir*, mettez le chemin vers le répertoire où vous souhaitez que zookeeper écrive ses données, *e.g.* dataDir=/usr/zookeeper/ (le répertoire doit être créé au préalable)
6. Enregistrez le fichier sous le nom <zooDir>/conf/zoo.cfg. Puis, lancez zookeeper avec la commande `zKserver.cmd`
7. Dans une autre fenêtre, vérifiez que zookeeper s'est bien lancé en lançant un client `zkCli.cmd`. Si tout se passe bien vous devez avoir `[zk: localhost:2181<CONNECTED> 0].`

ikhlef@gmail.com

Session illustrative

2. Upload d'une configuration existante

- Si vous avez déjà un core à distribuer, commencez par uploader sa config dans zookeeper.

- Zookeeper se charge de distribuer la config au reste des noeuds solr.

```
solr zk -z localhost:2181 upconfig -n moviesCoreConf
-d C:\<SolrDir>\server\solr\moviesCore\conf
```

  - `-n` : le nom que nous donnons à la config dans zk.

  - `-d` : le répertoire contenant la configuration de l'index existant.

j.khaled@gmail.com

Session illustrative

3. Lancement des serveurs SolR et connexions à Zookeeper

- Première méthode (2 instances indépendantes dans des répertoires différents)

1. Lancez le premier serveur :

```
solr start -c -z localhost:2181
```

2. Dézippez l'archive de solr dans un autre répertoire <solr2>. Ceci va constituer notre deuxième shard

3. Sous <solr2> \bin, lancez le deuxième serveur (avec un port différent) :

```
solr2\bin>solr start -c -z localhost:2181 -p 8984
```

jkhaled@gmail.com

Session illustrative

3. Lancement des serveurs SolR et connexions à Zookeeper

- Deuxième méthode : faire tourner les 2 instances sur la même machine

1. Sous le répertoire racine de SolR, créez 2 répertoires, *noeud*1 et *noeud*2.

2. Chacun de ces répertoires doit au minimum contenir un fichier solr.xml. Vous pouvez copier ce fichier à la main à partir de *server/solr* ou alors le copier dans zookeeper qui se chargera de le distribuer aux noeuds. *e.g.*

```
solr zk cp C:\<solDir>\server\solr\solr.xml zk:/solr.xml -z localhost
    :2181
```

3. Lancez le premier serveur, puis le second (avec des ports différents)

```
solr start -cloud -s C:\..solr-7.1.0\noeud1 -p 8983 -z localhost:2181
solr start -cloud -s C:\<solDir>\noeud2 -p 8984 -z localhost:2181
```

jmale@gmail.com

Session illustrative

4. Création d'une collection partitionnée et répliquées

1. Avec curl

```
curl http://localhost:8983/solr/admin/collections?
action=CREATE&name=shardedMovies&numShards=4&replicationFactor=2&
maxShardsPerNode=10&collection.configName=moviesCoreConf
```

2. ou

```
solr create -c shardedMovies2  -n moviesCoreConf -s 4 -rf 2
```

5. Vous pouvez si vous le souhaitez modifier une collection existante

1. avec curl

```
http://localhost:8983/solr/admin/collections?action=MODIFYCOLLECTION&
name=moviesCore&numShards=4&replicationFactor=2&maxShardsPerNode=10&
collection.configName=moviesCoreConf
```

2. ou (API v2)

```
http://localhost:8983/api/c/moviesCore -H 'Content-type:application/json' -d
    '"{ modify: { replicationFactor: "3", autoAddReplicas: false }}"'
```