Indexation pour la Recherche d'Information

Cours d'Indexation et Recherche d'Information

Aurélien Max

Master 2 Professionnel Informatique et MIAGE
Université Paris-Sud 11

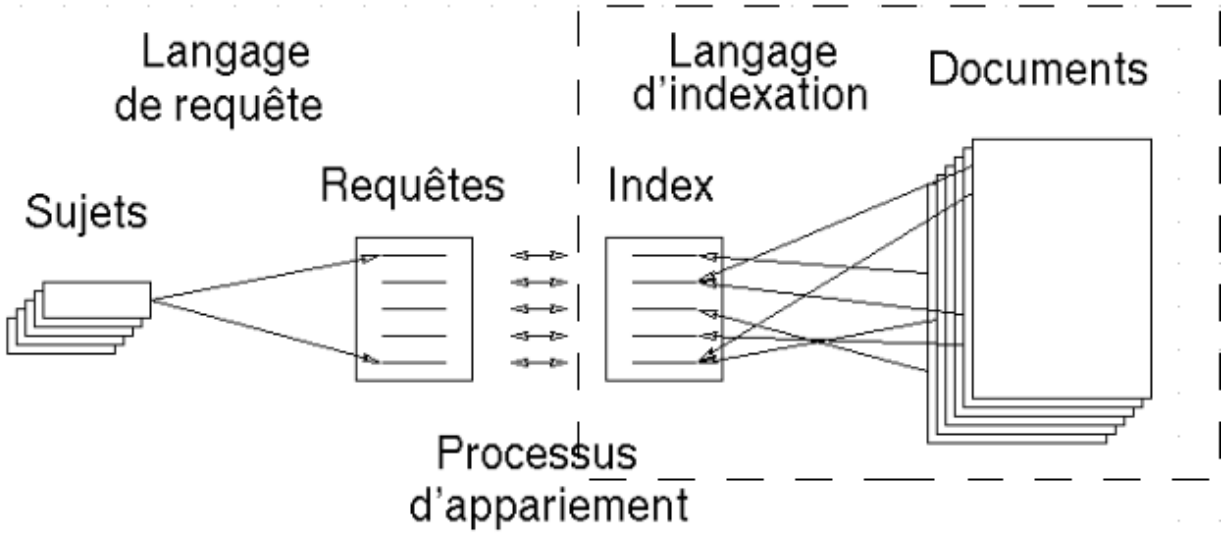
Année 2010-11

Rappel : index dans une collection de documents

- Les **index** sont utilisés pour représenter le contenu des documents (*substituts capables de les représenter*) :
  - ils ne représentent qu'une partie du contenu des documents
  - ils peuvent prendre plusieurs formes (ex : mots simples, termes, syntagmes, entrées dans un thésaurus, etc.)
  - ils sont plus ou moins difficiles à extraire
  - leur stockage requiert plus ou moins de mémoire
- Les **fichiers inverses** associent des index aux documents qui les contiennent, ex :

  abaissement de Ph $\rightarrow$ d2, d85, d22, d37
  abaissement de température $\rightarrow$ d3, d85
  abaissement de teneur $\rightarrow$ d782
  abattage à l'eau $\rightarrow$ d29, d74, d85
  ...

Processus d'indexation



Contraintes fortes de l'indexation :

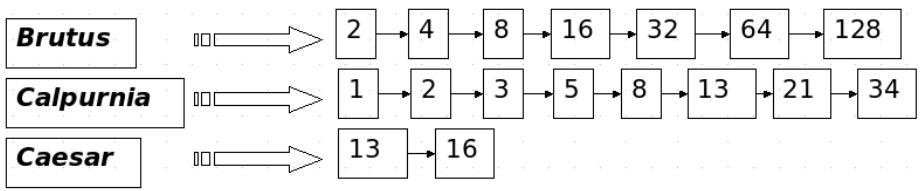
- stocker de grandes quantités d'information en limitant autant que possible l'espace utilisé
- extraire les éléments nécessaires pour le type de recherche d'information visée
- permettre un accès efficace aux index pendant la recherche
- permettre le cas échéant une mise à jour dynamique des index

Représentation du contenu des documents

- Possibilité de représenter la distribution des index dans les documents dans une matrice (termes x documents) :

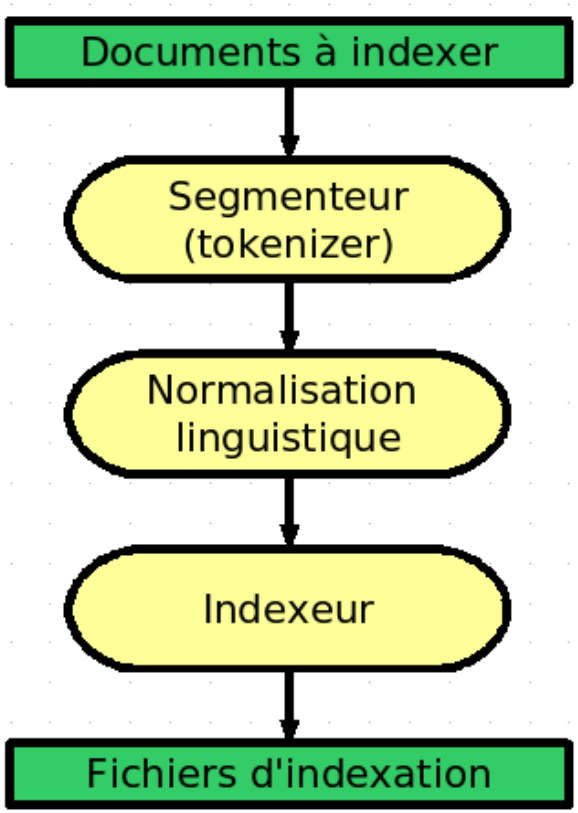
	Antony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth
Antony	1	1	0	0	0	1
Brutus	1	1	0	1	0	0
Caesar	1	1	0	1	1	1
Calpurnia	0	1	0	0	0	0
Cleopatra	1	0	0	0	0	0
mercy	1	0	1	1	1	1
worser	1	0	1	1	1	0

- La matrice peut être extrêmement creuse : on ne représente que les valeurs non nulles
- Les **fichiers inverses** associent aux termes les documents qui les contiennent
- Utilisation de listes chaînées pour représenter les listes de documents (allocation dynamique, insertion facile) :



Chaîne d'indexation

1. Segmentation des documents en unités
2. Normalisation linguistique
3. Production des fichiers d'indexation



Segmentation des documents (1/2)

- Caractéristiques des documents :
  - format de fichier (texte, HTML, PDF, etc.)
  - encodage (ASCII, ISO-LATIN-X, Unicode)
  - langue(s)
  - signes non linguistiques (formules, présentation, images, etc.)
- Une collection de documents peut contenir des textes en plusieurs langues
  - un index par langue ou un index unique
  - nécessite des techniques d'identification de langue
- Niveaux d'indexation :
  - documents entiers
  - sous-parties (voir le cours sur les documents semi-structurés)
  - sous-ensemble de documents (ex : site web)

Segmentation des documents (2/2)

Sur quels indices baser la segmentation ?

- utilisation de la ponctuation et de listes de séparateurs
- cas particuliers, ex :
  aujourd'hui, *pomme de terre*, *Le Mans*, *127.0.0.1*, *M. Durand*, *14/07/1789*
- certaines langues posent des difficultés supplémentaires, ex :
  - les mots composés allemands peuvent être très complexes, ex : *Levensversicherungsgesellschaftsangestellter* (*employé d'une société d'assurrance vie*)
  - le chinois et le japonais ne séparent pas les mots par des espaces, ce qui peut mener à plusieurs segmentations
  - le japonais autorise l'utilisation de plusieurs familles de caractères
  - l'arabe et l'hébreu sont écrits de droite à gauche, mais certains éléments tels que les nombres sont écrits de gauche à droite
  - etc.

Normalisation textuelle

- Possibilité de normaliser les éléments avant l'indexation, ex :
  - suppression des points dans les acronymes (ex : *U.S.A.* $\rightarrow$ *USA*)
  - suppression des accents (ex : *météo* $\rightarrow$ *meteo*)
  - suppression de certaines majuscules (ex : *Et* $\rightarrow$ *et*)
  - normalisation de valeurs particulières (peut être difficile !), ex :
    - les dates, ex : *14 juillet 1789* $\rightarrow$ *14/07/1789*
    - les valeurs monétaires, ex : *$400* $\rightarrow$ *400 dollars*
    - les organisations, ex : *Organisation des Nations Unies* $\rightarrow$ *ONU*
- Le choix des différentes normalisations peut se baser sur les usages des utilisateurs
- Possibilité de ne pas normaliser dans les index :
  - taille d'index plus importante
  - besoin de mécanismes **d'expansion** lors de la recherche

Normalisation linguistique

- Possibilité d'appliquer des techniques de **correction**, ex :
  - réaccentuation (ex : *meteo* $\rightarrow$ *météo*)
  - correction orthographique (ex : *inofrmation* $\rightarrow$ *information*)
  - correction grammaticale (ex : *les informations recherché* $\rightarrow$ *les informations recherchées*)
- Possibilité de ramener plusieurs éléments à une même forme, ex :
  - **racinisation** (mots de même racine), ex :
    malade, malades, maladie, maladies, maladive $\rightarrow$ *malad*
  - **lemmatisation** (mots de même lemme), ex :
    produis, produit, produisons, ... $\rightarrow$ *produire*
  - **normalisation phonétique** (mots de même prononciation)
    chebyshev $\rightarrow$ *tchebycheff*
  - utilisation d'autres classes d'équivalence (ex : synonymes, thésaurus), ex :
    tuer, assassiner, massacrer, trucider $\rightarrow$ *tuer*

Normalisation linguistique : racinisation (stemming)

- Ramener les mots à leur racine
- Utilisation de règles dépendantes de la langue
- Exemple : algorithme de Porter pour l'anglais
  - ex : *automates, automatic, automation* $\rightarrow$ *automat*
  - conventions et phases de réduction
    - exemple de convention : appliquer en priorité les règles qui s'appliquent aux suffixes les plus longs
    - exemples de règles :
      sses $\rightarrow$ *ss* ; *ies* $\rightarrow$ *i* ; *ational* $\rightarrow$ *ate* ; *tional* $\rightarrow$ *tion*
- Exemple de racinisation pour le français :
  malade, malades, maladie, maladies, maladive $\rightarrow$ *malad*
- Normalisation linguistique :
  - diminution significative de la taille des index
  - nombreuses possibilités d'erreurs
  - les classes importent plus que les racines elles-mêmes
  - impossibilité de distinguer plusieurs formes via les index

Limites de la racinisation

- Exemple (Manning *et al.*) :
  - L'algorithme de Porter réalise les transformations suivantes : *operate operating operates operation operative operatives operational* $\rightarrow$ *oper*
  - Perte de précision attendue pour des requêtes telles que :

    `operational and research`

    `operating and system`

    `operative and dentistry`
- Possibilité d'avoir recours à une analyse linguistique (**lemmatisation**) donnant les **lemmes** (entrées des dictionnaires)
  - nécessite un lemmatiseur pour la langue concernée
  - impact plus important pour les langues fortement fléchies

Racinistion et lemmatisation : évaluation

Évaluation de l'utilisation de racines et de lemmes sur l'anglais (Moreau *et al.*, 2007) :

	P(20)	P(100)	P(1000)	R(20)	R(100)	R(1000)
formes	42.10	23.70	4.63	20.76	45.48	71.51
racines	44.90	25.36	5.40	22.01	46.22	78.74
lemmes	41.90	25.46	5.42	20.24	46.98	77.79

Normalisation linguistique : correction orthographique

- Les erreurs orthographiques peuvent être des erreurs de saisie ou de reconnaissance automatique de caractères (OCR)
- Approches possibles :
  - correction dans les index, mais impossible de le faire de façon (semi-)interactive (utilisation de dictionnaires)
  - correction dans les requêtes (utilisation des index)
- Deux approches pour la correction automatique :
  - correction des mots en isolation (ex : *inofrmation*)
    - calcul de distance d'édition : coût de transformation d'une chaîne en une autre avec les opérations d'insertion, de suppression et de substitution
    - possibilité de pondérer les opérations pour prendre en compte des erreurs fréquentes : saisie (ex : $a \rightarrow q$), reconnaissance (ex : $D \rightarrow O$)
  - correction des mots en contexte (ex : *flight* ***form*** *Eathrow*)
    - trouver des mots proches pour chaque mot et tester les fréquences des combinaisons (cf. « *Essayez avec cette orthographe* »)
    - utiliser les plus grands ensembles ou les requêtes les plus populaires

Normalisation linguistique : normalisation phonétique

- Ramener à une même forme des mots de même prononciation
  ex : *chebyshev* $\rightarrow$ *tchebycheff*
- Algorithme du **Soundex** :
  - chaque mot est compressé en une forme réduite de 4 caractères
  - construction d'un index de formes réduites d'équivalents
  - extrait de l'algorithme :
    - conserver la première lettre d'un mot
    - remplacer les lettres a, e, i, o, u, h, w, y par 0
    - autres corrrespondances : B, F, P, V $\rightarrow$ 1 ; C, G, J, K, Q, S, X, Z $\rightarrow$ 2
    - suppression des nombres répétés et des 0
    - on obtient un code normalisé, ex : *Herman* $\rightarrow$ *H655*

Normalisation linguistique : variantes de termes

- Regrouper des **variantes de termes** (à déterminer), ex :
  - *genetic disease* (terme de base)
  - *disease is genetic* (variante syntaxique)
  - *hereditary disease* (variante sémantique)
  - *genetically determined forms of the disease* (variante morpho-syntaxique)
  - *disease is familial* (variante syntaxico-sémantique)
  - *transmissible neurodegenerative diseases* (variante syntaxico-sémantique)
- Nécessite l'identification de termes normalisés
- Possibilité d'**indexation dynamique** en fonction des termes d'une requête pour des bases de documents de taille réduite

cf. le cours sur la détection de variantes de termes

Étapes de l'indexation (1/2)

1. Extraction des index des documents
   - extraction de chaque occurrence
   - mémorisation du document pour chaque occurrence
2. tri des index
3. regroupement des index
   - indication du nombre d'occurrences par document
   - une seule entrée par couple (index, document)
4. Séparation en **dictionnaire** et **liste de postings**
   - comptage du nombre total pour chaque index dans la collection complète
   - obtention d'un **dictionnaire** énumérant les index et leur nombre d'occurrences dans le corpus
   - obtention d'une **liste de postings**, associant un nombre d'occurrences à chaque document pour un index du dictionnaire

Étapes de l'indexation (2/2)

Term	Doc #
I	1
did	1
enact	1
julius	1
caesar	1
I	1
was	1
killed	1
i'	1
the	1
capitol	1
brutus	1
killed	1
me	1
so	2
let	2
it	2
be	2
with	2
caesar	2
the	2
noble	2
brutus	2
hath	2
told	2
you	2
caesar	2
was	2
ambitious	2

extraction des index

Term	Doc #
ambitious	2
be	2
brutus	1
brutus	2
capitol	1
caesar	1
caesar	2
caesar	2
did	1
enact	1
hath	1
I	1
I	1
i'	1
it	2
julius	1
killed	1
killed	1
let	2
me	1
noble	2
so	2
the	1
the	2
told	2
you	2
was	1
was	2
with	2

tri des index

Term	Doc #	Freq
ambitious	2	1
be	2	1
brutus	1	1
brutus	2	1
capitol	1	1
caesar	1	1
caesar	2	2
did	1	1
enact	1	1
hath	2	1
I	1	2
i'	1	1
it	2	1
julius	1	1
killed	1	2
let	2	1
me	1	1
noble	2	1
so	2	1
the	1	1
the	2	1
told	2	1
you	2	1
was	1	1
was	2	1
with	2	1

regroupement

dictionnaire

Term	N docs	Tot Freq
ambitious	1	1
be	1	1
brutus	2	2
capitol	1	1
caesar	2	3
did	1	1
enact	1	1
hath	1	1
I	1	2
i'	1	1
it	1	1
julius	1	1
killed	1	2
let	1	1
me	1	1
noble	1	1
so	1	1
the	2	2
told	1	1
you	1	1
was	2	2
with	1	1

postings

Doc #	Freq
2	1
2	1
1	1
2	1
1	1
1	1
2	2
1	1
1	1
2	1
1	2
1	1
2	1
1	1
1	2
2	1
1	1
2	1
2	1
1	1
2	1
2	1
2	1
1	1
2	1
2	1

séparation en dictionnaire et *postings*

D'après (Manning et Prabhakar, 2004)

Espace mémoire et temps de traitement

- Les dictionnaires sont stockés en mémoire de travail
- Les postings sont (en général) en mémoire de stockage
- Compromis entre **techniques de compression** de l'information et **vitesse d'exécution** des requêtes, ex :
  - utilisation de la racinisation (stemming)
    - nombre de termes réduit d'environ 40%
    - nombre de pointeurs réduit de 10-20%
    - espace total réduit d'environ 30%
  - utilisation de listes de mots vides (stop lists)
    - règle des 30 : environ 30 mots représentent environ 30% des occurrences de termes dans des textes écrits
    - éliminer les 150 termes les plus fréquents réduit l'espace d'environ 25%

Structures de données

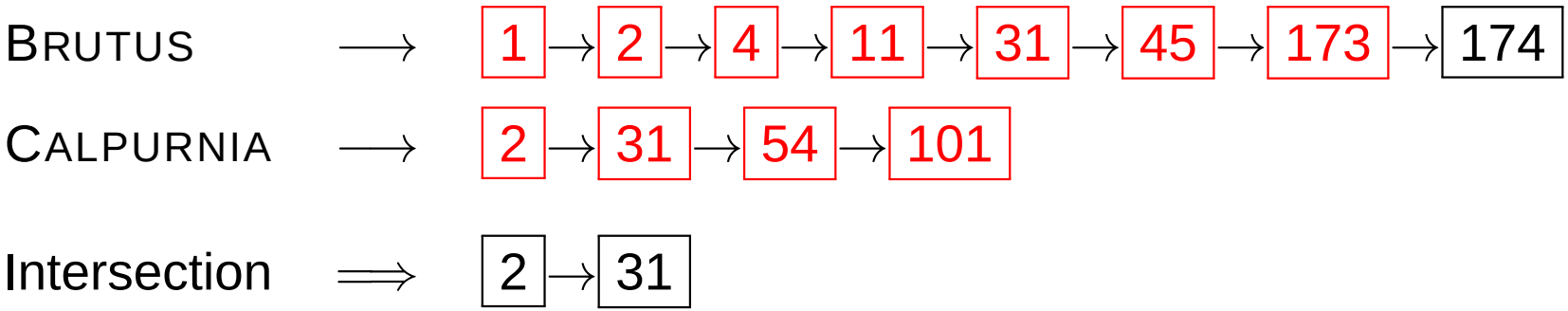
- Critères
  - nombre fixe ou variable d'éléments ? nombre important ?
  - fréquences d'accès aux différentes valeurs ?
- Utilisation de **tables de hachage**
  - consultation très rapide
  - reconstruction périodique si le vocabulaire évolue
  - impossible d'effectuer une recherche par préfixe (ex : *automat**)
- Utilisation d'**arbres**
  - reconstruction en cas de modification du vocabulaire
  - utilisation de B-arbres (nombre de fils compris entre 2 bornes)
  - recherche par préfixe possible

Requêtes avec jockers

- Recherche de préfixe avec un B-arbre facile, ex :
  pour *univers**, trouver les entrées entre *univers* et *univert* (exclu)
- Recherche de suffixe à l'aide d'un arbre pour les mots inversés :
  pour **tion*, trouver les entrées entre *noit* et *nois* (exclu)
- Recherche simultanée de préfixe et suffixe, ex :
  - pour *dé*tion*, faire l'intersection des résultats de *dé** et **tion* est très coûteux
  - utilisation d'index de permutation (***permuterm***) : rotation du terme pour que le jocker apparaisse à la fin
  - ex : pour *test*, on ajoute à l'arbre *test$*, *est$t*, *st$te* et *t$tes*
  - requêtes (d'après (Schutze))
    - pour *X*, chercher *X$*
    - pour *X**, chercher *X*$*
    - pour **X*, chercher *X$**
    - pour **X**, chercher *X**
    - pour *X*Y*, chercher *Y$X**
  - la taille d'un B-arbre peut quadrupler
  - possibilité d'utiliser des index de *k*-grams

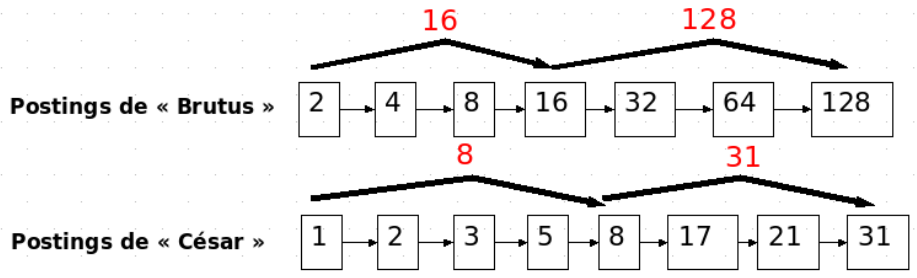
Utilisation des index pour des requêtes booléennes

- La liste des postings d'un index est la liste ordonnée de tous les documents qui le contiennent
- On trouve l'ensemble de documents contenant deux index (ex : `Brutus AND Calpurnia`) en calculant l'intersection de leurs listes de postings (temps linéaire sur le nombre d'entrées)



Utilisation des index pour des requêtes booléennes

- Les requêtes booléennes peuvent être optimisées pour limiter la taille des ensembles manipulés
- Possibilité d'ajout de pointeurs de saut (*skip pointers*) aux listes de postings pour sauter les entrées qui ne sont pas concernées par la recherche (accélérer les intersections), ex :



- Problème : comment répartir et maintenir ces pointeurs ?
  - beaucoup : un pointeur ne permet de sauter que quelques documents, mais fréquemment utilisé
  - peu : un pointeur permet de sauter de nombreux documents, mais peu utilisé
- Heuristique simple :
  - répartir de façon homogène $\sqrt{P}$ (P : taille de la liste de postings) pointeurs de saut
  - gestion plus difficile sur un index dynamique

Recherche de segments contigus

- Permettre la recherche de segments (ex : *"Université Paris Sud"*, environ 10% des requêtes) ou de proximités (ex : `moteur NEAR recherche`)
- Possibilité d'utiliser des index constitués de groupes de mots consécutifs, ex :
  "Université Paris Sud" $\rightarrow$ *"Université Paris"*, *"Paris Sud"*
- De longs segments peuvent être décomposés, ex :
  `"Organisation des Nations Unies"` $\rightarrow$ `"Organisation des" AND "Nations Unies"`
- Nécessite une étape de post-filtrage sur les documents pour trouver ceux qui contiennent effectivement le segment complet
- Inconvénients des index de groupes de mots
  - peuvent retourner des faux positifs
  - taille conséquente

Mémorisation de la position des index

- Mémorisation de la position (*offset*) des occurrences d'index dans les documents
- Augmentation significative de la taille de l'index (x2 à x4)
  - jusqu'à 50% de la taille du texte indexé
- Exemple : *"to be or not to be"*
  - extraction des entrées d'index pour chaque terme (*to, be, or, not*)
  - regroupement des positions dans les documents et recherche des segments :

    TO, 993427 :
    ⟨ 1, 6 : ⟨7, 18, 33, 72, 86, 231⟩ ;
    2, 5 : ⟨1, 17, 74, 222, 255⟩ ;
    4, 5 : ⟨8, 16, 190, 429, 433⟩ ;
    5, 2 : ⟨363, 367⟩ ;
    7, 3 : ⟨13, 23, 191⟩ ; . . . ⟩

    BE, 178239 :
    ⟨ 1, 2 : ⟨17, 25⟩ ;
    4, 5 : ⟨17, 191, 291, 430, 434⟩ ;
    5, 3 : ⟨14, 19, 101⟩ ; . . . ⟩

Recherche par proximité

- Possibilité d'utiliser les index de position pour faire des recherches de proximité (ex : *moteur NEAR/3 recherche*)
- Trouver tous les documents qui contiennent les deux termes est peu efficace (en particulier pour les mots fréquents)
- Possibilité d'utiliser les index de position
  - on peut obtenir les positions dans les documents (par ex. mise en valeur des mots dans les résultats)
- Possibilité de combiner
  - utiliser des index de segments pour les groupes de mots très fréquents (ex : *New York*, *John Lenon*)
  - ces segments peuvent simplement apparaître dans le vocabulaire de l'index de mots

Indexation dynamique

- Les documents changent avec le temps :
  - apparition de nouveaux index / documents
  - nouvelles occurrences d'index
  - disparition d'index / documents
- Les mises à jour dynamiques sur l'index sont plus compliquées à mettre en œuvre
- Compromis entre l'accès et la mise à jour : stockage en mémoire de travail ou sur mémoire de masse
- Approche simple :
  - maintenir un **index principal** et un **index auxilaire**
  - recherche dans les deux index et regroupement des résultats
  - utilisation d'un **vecteur d'invalidation** pour les documents disparus
  - fusion périodique des deux index

Importance des index

- Tous les index dans les documents n'ont pas la même importance
  - utilisation de **listes de mots vides** (*stop lists*), qui sont éliminés lors de l'indexation et des requêtes
  - **pondération** des différents index
- Modèles **sacs de mots** (*bag-of-words*)
  - nombre d'occurrences d'un terme dans chaque document
  - on parle de la **fréquence** du terme dans un document
  - ne tiennent pas compte de l'ordre des mots
- On peut rendre compte de l'importance d'un terme dans un document relativement au corpus de documents
  - ex : pondération par ***tf.idf***

Pondération par *tf.idf*

- Calcul du poids d'un terme dans un document :

$$w_{i,d} = tf_{i,d} * idf_i \quad (1)$$

$tf_{i,d}$: fréquence du terme i dans le document d
idf_i : importance du terme i dans la collection (*inverse document frequency*)

  - mesure simple : inverse du nombre de documents de la collection contenant le terme

$$w_{i,d} = tf_{i,d} * \frac{1}{df_i} \quad (2)$$

  - mesure fréquemment utilisée : log du quotient du nombre de documents dans la collection par le nombre de documents contenant le terme

$$w_{i,d} = tf_{i,d} * \log(\frac{N}{df_i}) \quad (3)$$

- Le poids d'un terme augmente :
  - avec la fréquence du terme dans un document
  - avec la rareté du terme dans les documents du corpus

Pondération par *tf.idf* : exemple

	Antony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth
Antony	157	73	0	0	0	0
Brutus	4	157	0	1	0	0
Caesar	232	227	0	2	1	1
Calpurnia	0	10	0	0	0	0
Cleopatra	57	0	0	0	0	0
mercy	2	0	3	5	5	1
worser	2	0	1	1	1	0

	Antony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth
Antony	13.1	11.4	0.0	0.0	0.0	0.0
Brutus	3.0	8.3	0.0	1.0	0.0	0.0
Caesar	2.3	2.3	0.0	0.5	0.3	0.3
Calpurnia	0.0	11.2	0.0	0.0	0.0	0.0
Cleopatra	17.7	0.0	0.0	0.0	0.0	0.0
mercy	0.5	0.0	0.7	0.9	0.9	0.3
worser	1.2	0.0	0.6	0.6	0.6	0.0

D'après (Manning et Prabhakar, 2004)

Score de recouvrement (*overlap*)

- Chaque document peut être vu comme un vecteur, chaque dimension correspondant à un terme ayant pour valeur son poids
- Mesure simple de similarité entre une requête *req* et un document *doc* :

$$Score(req, doc) = \sum_{terme \in req} tf_{terme,doc} \tag{4}$$

- Utilisation du poids plutôt que de la fréquence :

$$Score(req, doc) = \sum_{terme \in req} tf\_idf_{terme,doc} \tag{5}$$

(suite dans le cours sur la recherche)

Exemple d'indexation : Google'98 (Brin et Page, 1998)

- Critère essentiel : faible temps de calcul sur de grands ensembles de données
- Conversion des documents en *hits* (occurrences de mots) comprennant :
  - le mot, converti en wordID à l'aide d'une table de hashage (vocabulaire)
  - la position dans le document
  - la taille relative de la fonte
  - la casse du mot
- L'indexeur répartit les hits dans des *barriques* (*barrels*) qui constituent un index partiel
  - le tri requiert peu d'espace temporaire
  - mise à jour régulière et indépendante
- Les barriques sont ensuite triées régulièrement par wordID pour constituer deux index :
  - pour les mots apparaissant dans les titres et les ancres
  - pour le texte intégral

Google'98 : indexation (1/2)

- Un module de résolution d'URLs (`URLResolver`) convertit les URLs en docID
- L'indexeur analyse les liens dans les pages et alimente la base de liens (utilisée pour le calcul du PageRank) avec des paires de docID : origine et cible, et texte des liens
- Les structures de données utilisées sont très compactes, afin de minimiser les accès disque
  - fichiers virtuels (adressables sur 64 bits) pouvant être répartis sur plusieurs systèmes de fichiers (BigFiles)
  - compression des pages mémorisées par zlib (facteur 3) (favorisation du temps d'accès)
  - l'index est composé d'entrées de taille fixe triées par docID (accès séquentiel)
  - les entrées contiennent notamment un pointeur dans l'archive (repository) qui permet un accès en un positionnement sur le disque
  - fichiers d'associations entre checksums d'URL et docID

Google'98 : indexation (2/2)

- Lexique : en mémoire vive (14.10^6 mots : moins de 256Mo)
- Listes de hits (occurrence d'un mot particulier dans un document particulier) :
  - représentent la majeure partie de la taille des index
  - deux types de hits (encodage compact sur 2 octets) : *fancy hits* (mots dans une URL, un titre, un texte d'ancre ou une valeur de métadonnée) et *plain hits*
  - L'index (*forward index*) réparti est en 64 barriques (barrels) :
    - chaque barrique correspond à une plage d'identifiants de mots (wordID)
    - le docID d'un document qui contient des mots correspondant à une barrique donnée est ajouté à la barrique avec la liste des wordIDs correspondant et leur liste de hits
  - Index inversé : mêmes barriques que l'index mais triées
    - le lexique contient pour chaque wordID un pointeur vers la barrique qui le contient
    - ce pointeur pointe sur une liste de docID et leur liste de hits
    - les docID sont triés : permet des recherches rapides pour des requêtes comprenant plusieurs mots
    - deux ensembles d'index inversés : *fancy hits* et *plain hits*

Bibliographie du cours

Brin, Sergey et Lawrence Page (1998) *The anatomy of a large-scale hypertextuel web search engine*, in Computer Networks and ISDN Systems, Vol. 30, Num. 1

Fluhr, Christian (2000) *Indexation et recherche d'information textuelle*, in Ingénierie des Langues, Jean-Marie Pierrel éditeur, Hermès

Jacquemin, Christian (2004) *Indexation et Recherche d'Information*, Cours de DESS II et SCHM, Université Paris-Sud 11

Lefèvre, Philippe (2000) *La recherche d'informations*, Hermès Sciences, Paris

Manning, Christopher et Prabhakar Raghavan (2004) *Text retrieval and mining*, CS276A, Cours, Université Stanford

Manning, Christopher, Prabhakar Raghavan et Hinrich Schütze (2008) *Introduction to Information Retrieval*, Cambridge University Press

Moreau, Fabienne, Vincent Claveau et Pascale Sébillot (2007) *Intégrer plus de connaissances linguisitiques en recherche d'information peut-il augmenter les performances des systèmes ?*, Actes de CORIA'07, Saint-Etienne, France

Schutze, Hinrich (2007) *Introduction to Information Retrieval*, Cours, Université de Stuttgart