Lundi 16/11

Introduction

Recherche de l'information : rechercher des données dans infos non strucuturées

à ≠ échelles:

- petite échelle (*par exemple: les emails)*
- moyenne échelle (*dans une société, pour une recherche de documents*)
- grande échelle (moteur de recherche pour rechercher une information dans tous les sites web existants)

Données structurées : données classées et nommées, on peut les ranger dans une BD, et on fait la recherche d'info avec des requêtes SQL par exemple

⇒ relatif aux bases de données

Données non structurées :

- il n'y a **pas** de champs
- peut être du **texte**, vidéo, etc.
- permet des requêtes incluant des opérateurs

Recherche d'information:

- **Collection** : un ensemble de documents (*pour le début du cours, on considère que la collection est statique*)
- **Objectif** :

Évaluation de la performance d'un système de recherche:

- ev
- vefv

VP: Vrais Positifs

VN: Vrais Négatifs

FN: Faux Négatifs

FP: Faux Positifs

Formules:

$$précision = VP/(FP + VP) * 100$$

$$rappel = VP/(FN + VP) * 100$$

Recherche d'information

Illustration / Exemple:

But: on recherche dans la collection de Shakespeare, toutes les pièces qui contiennent les mots **Brutus ET Caesar mais PAS Calpurnia**

Utilisation de matrices d'incidence

	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

Si :

- 1: le mot est dans la pièce
- 0: le mot n'est pas dans la pièce

Pour Calpurnia: 010000, on inverse 101111 pour faire en sorte que le résultat correspondent à la recherche

Opération binaire:

110100 **AND**

110111 **AND**

101111

100100

⇒ Mais qd la collection devient trop grande:

- la matrice n'est pas applicable car la matrice sera énorme
- majoritairement composée de 0 et quelques 1
- Plus de 99% des valeurs seront = 0 ⇒ **grnde perte d'info**

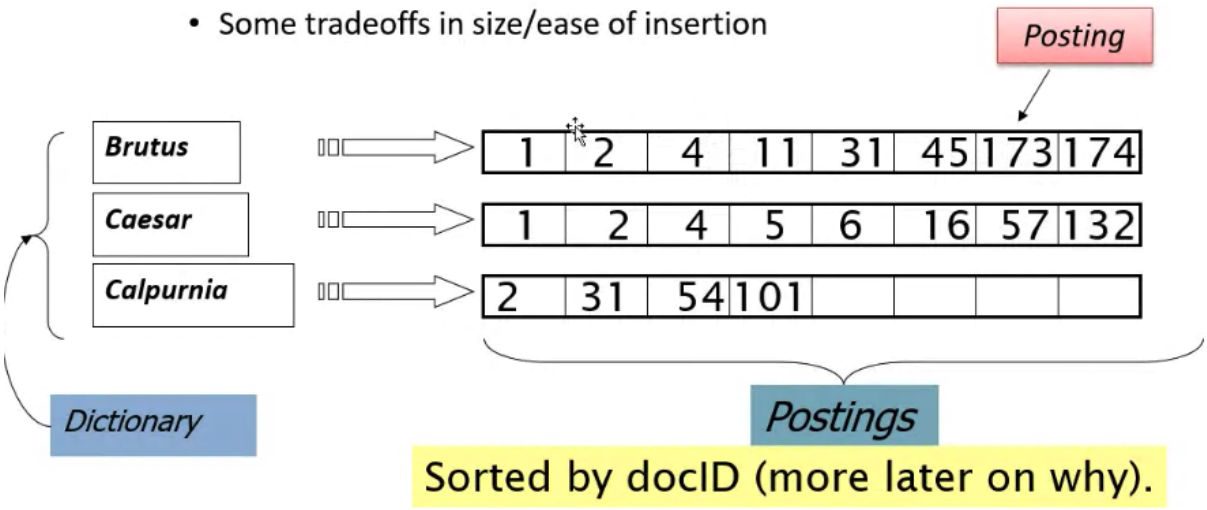
⇒ Méthode non utilisée car la majorité des infos sont à 0

Modèle d'index inversé

⇒ Système où on code uniquement la présence des mots (1)

→ on définit un dictionnaire (liste de tous les mots du dictionnaire), pour chaque mot, il y a un pointeur vers la liste, identifié par un **docID**

→ Posting list: liste de mots classés dans l'ordre croissant



Étapes

1. Récupérer tous les mots de chaque document dans une colonne d'un tableau, avec leur docID
2. On trie tous les **mots** par ordre alphabétique
3. On crée l'index inversé à partir de la dernière table
    - On ajoute le mot dans le dictionnaire (et on mets à jour en fonction du nb de fois où il apparait)

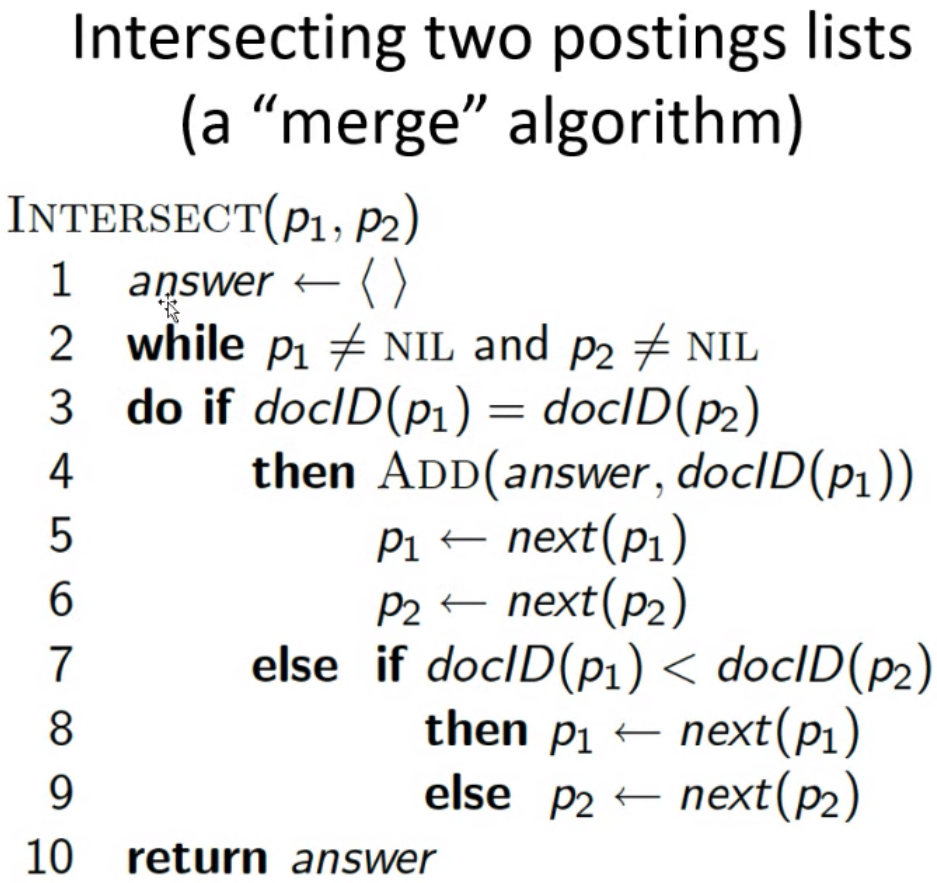
- Dans la posting list, on met le **docID** correspondant au mot (s'il apparait au moins 2 fois, on met le docID de sa dernière apparition)

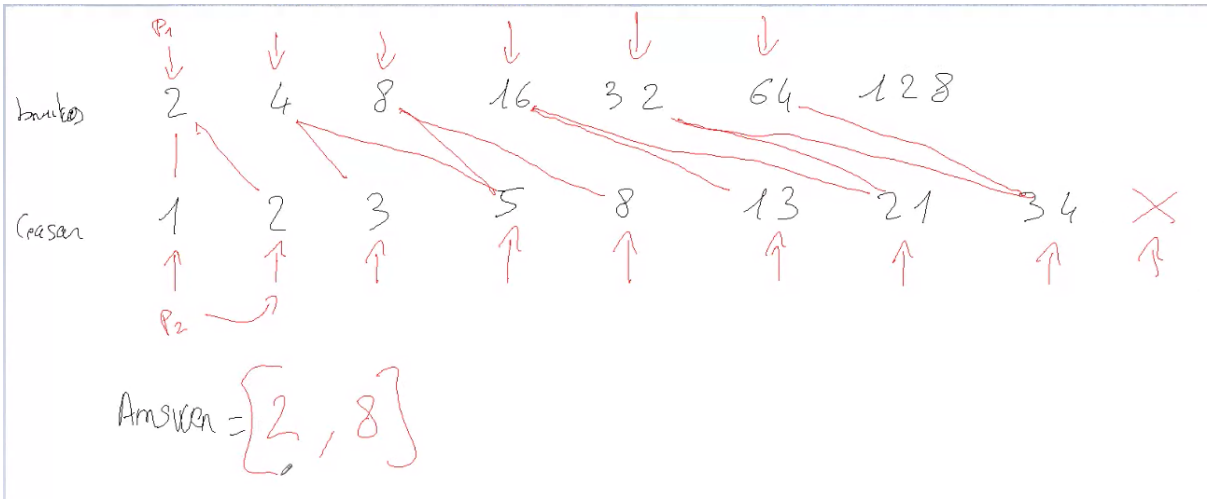
PS: On stocke le dictionnaire dans la RAM de l'ordinateur (car il doit être disponible tout le tps), et la posting list sur le disque dur

Utilisation

Dans le cas de notre exemple de départ:

1. On cherche "Brutus" dans le dictionnaire et on prend sa posting liste
2. On cherche "Caesar" dans le dictionnaire et on prend sa posting liste
3. On fusionne les deux listes ensemble le plus rapidement possible





Dès qu'on arrive à la fin d'une liste, on s'arrête car on sait qu'il y a plus d'autre occurence dans la 2e liste (**grâce à l'algo**)

Phrases de requêtes

→ Utilisation de **bi-mots (biwords)** : paires de mots consécutifs