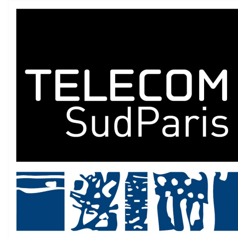


Recherche d'informations

Ou comment construire un moteur de recherche

Julien Romero

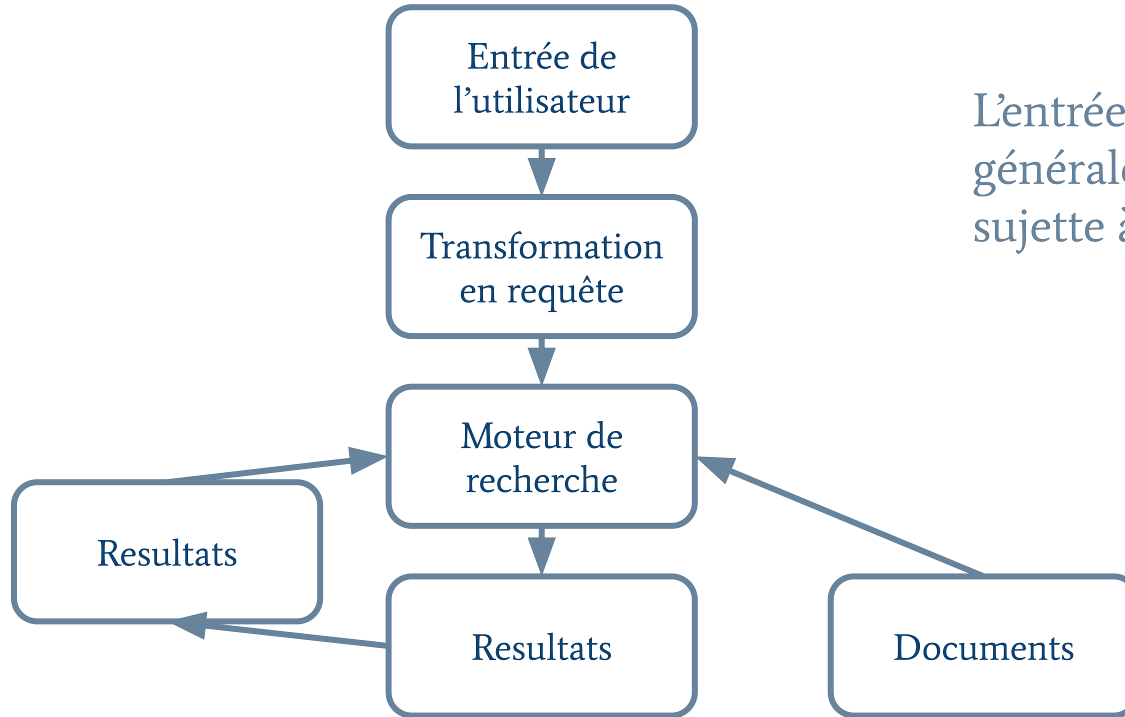
Recherche d’informations

La **recherche d’information** (ou information retrieval en anglais, IR) est une discipline qui consiste à trouver un **document non structuré** (généralement textuel) qui contient une **information donnée** dans une **large collection de documents**.

Applications :

- Recherche web
- Recherche dans les mails
- Recherche sur votre ordinateur
- Recherche dans les données d’une entreprise

Architecture classique



L'entrée de l'utilisateur est en langage généralement en langage naturel, donc sujette à ambiguités.

La matrice d’incidence terme-document

Exemple - Recherche dans des CVs

- Vous êtes une entreprise de recrutement et vous souhaitez pouvoir rapidement accéder aux meilleurs CV contenant l'information que vous recherchez
- Par exemple, trouver toutes les personnes sachant programmer en Python et Java, mais n'ayant pas fait d'école d'ingénieur.
  - Ici, notre requête serait Python AND Java AND NOT Ingénieur
  - (on suppose que l'on peut uniquement recherche par mot dans le CV)
- Solution naive : Une boucle for
  - Beaucoup trop long pour de larges collections de documents !

Première solution : la matrice d'incidence terme-document

- Matrice M de dimension Nombre de termes (mots) * Nombre de documents telle que M[i][j] vaut 1 si le terme i est dans le document j, 0 sinon.

	Jean	Paul	Jacques	Daniel	Didier	Henri
Python	1	1	0	0	1	0
Java	0	1	1	1	1	0
C	0	0	0	0	0	1
Ingénieur	1	0	1	1	1	0
Licence	1	1	1	1	0	1

Calcul des correspondances

- Chaque terme peut être représenté par un vecteur binaire
  - Python = 110010
  - Java = 011110
  - Ingénieur = 101110
- Une requête logique peut être transcrite par des opérations bit-à-bit
  - Python AND Java AND NOT Ingénieur
    = 110010 AND 011110 AND NOT 101110
    = 010010 AND 010001
    = 010000 => Paul

Désavantages

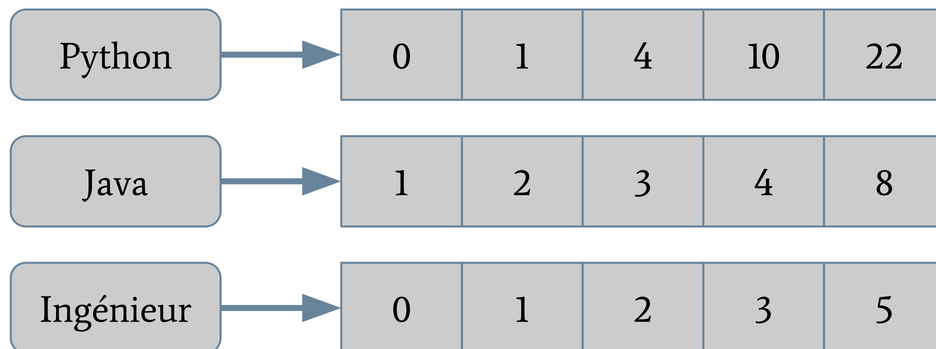
Matrice rapidement très grosse :

- 1 million de documents, 1000 mots par documents, 5 lettres par mots
    - $5*10^9$ lettres = 5 GB de données
- S'il y a 500k mots différents, cela fait une matrice avec 10^{14} cases
    - Matrice très sparse
    - Il faut adapter les représentations

L’index inversé

L'index inversé

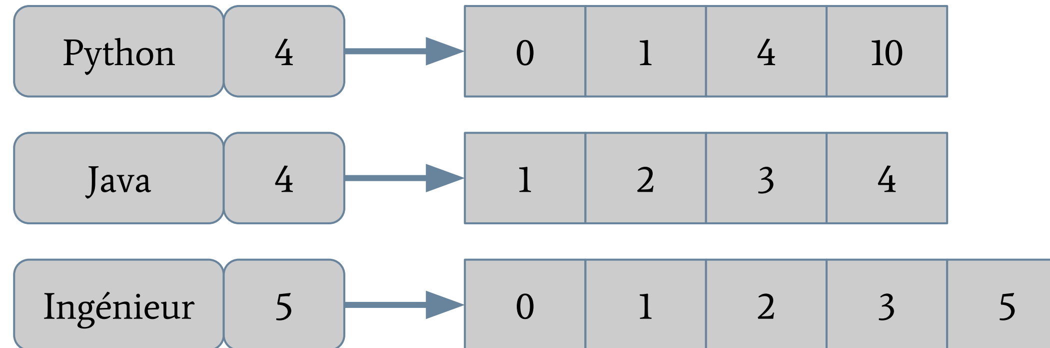
Idée principale : Pour chaque term t, nous stockons la liste (triée) de tous les documents contenant t (leur ID, plus précisément).



En pratique : Un dictionnaire dont les valeurs sont un tableau extensible ou une liste chaînée

L'index inversé - Fréquence de documents

Il est souvent utile de stocker le nombres de documents contenant le terme.



D'où viennent les termes ?

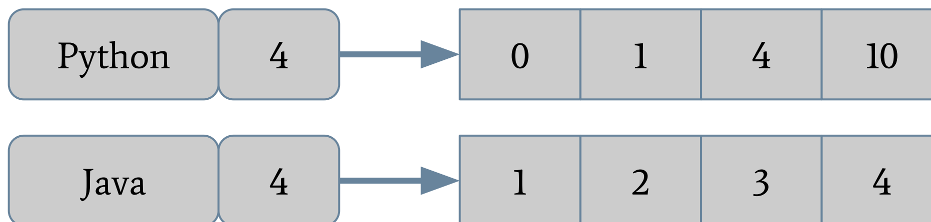
Voir le cours **introduction au traitement du langage naturel** !

- Tokenisation
- Normalisation

Traitement d'une requête

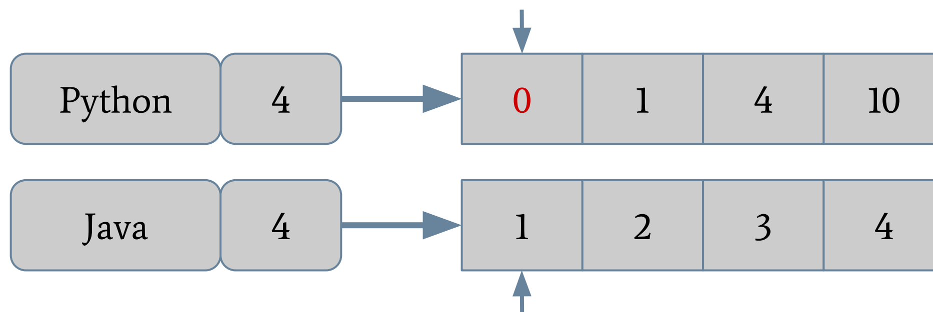
La requête AND

- Une requête q1 AND q2 retourne tous les documents répondant à la requête q1 et q2.
  - Dans le cas simple, on veut que deux termes apparaissent dans le document
  - Ex. : Python AND Java
- Comment faire avec un index inversé ?
  - Il faut faire l'intersection de deux listes triées.



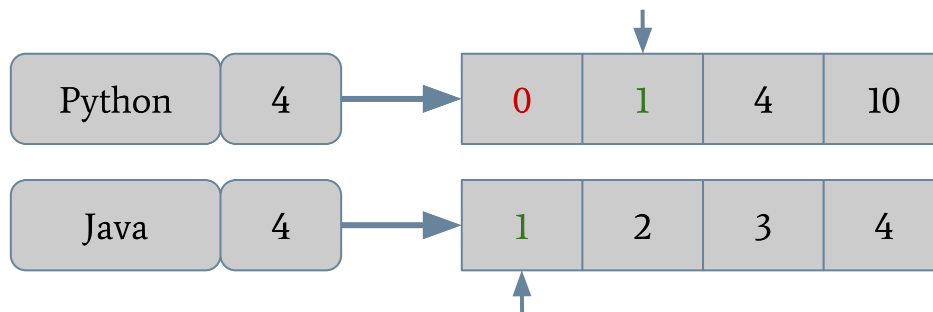
La requête AND

- Une requête q1 AND q2 retourne tous les documents répondant à la requête q1 et q2.
  - Dans le cas simple, on veut que deux termes apparaissent dans le document
  - Ex. : Python AND Java
- Comment faire avec un index inversé ?
  - Il faut faire l'intersection de deux listes triées.



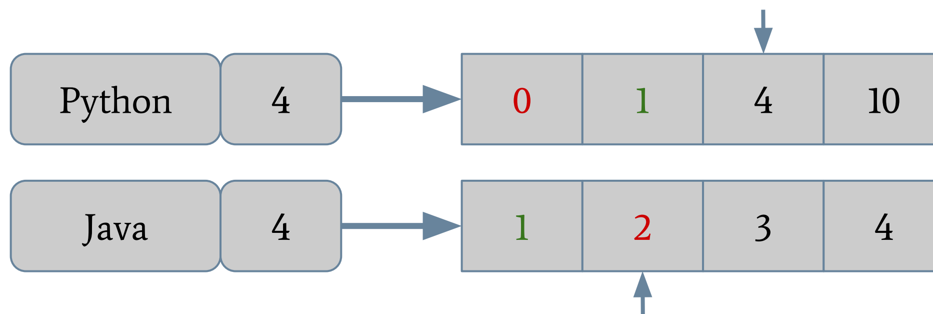
La requête AND

- Une requête q1 AND q2 retourne tous les documents répondant à la requête q1 et q2.
  - Dans le cas simple, on veut que deux termes apparaissent dans le document
  - Ex. : Python AND Java
- Comment faire avec un index inversé ?
  - Il faut faire l'intersection de deux listes triées.



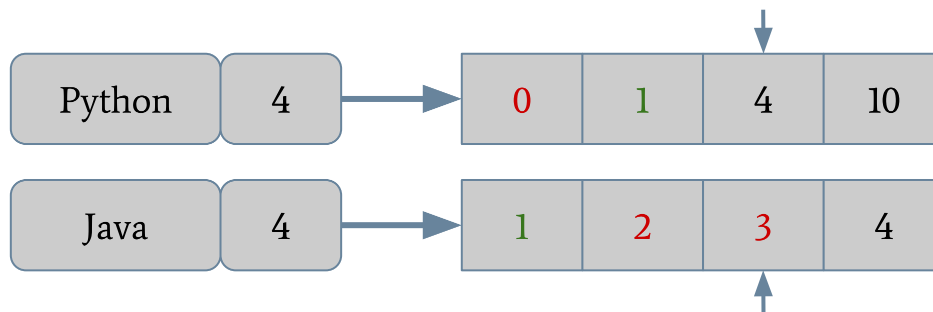
La requête AND

- Une requête q1 AND q2 retourne tous les documents répondant à la requête q1 et q2.
  - Dans le cas simple, on veut que deux termes apparaissent dans le document
  - Ex. : Python AND Java
- Comment faire avec un index inversé ?
  - Il faut faire l'intersection de deux listes triées.



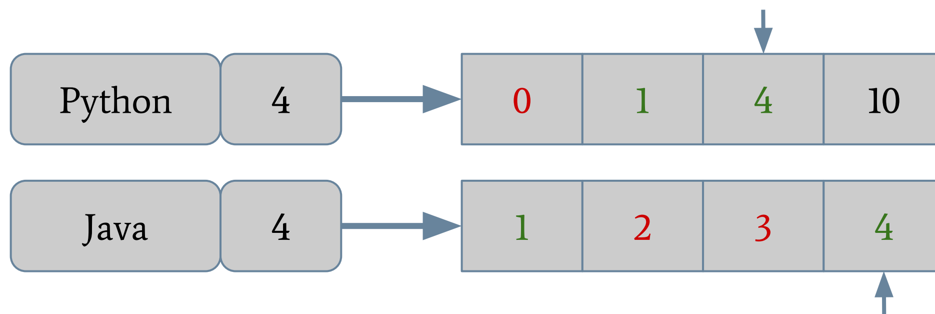
La requête AND

- Une requête q1 AND q2 retourne tous les documents répondant à la requête q1 et q2.
  - Dans le cas simple, on veut que deux termes apparaissent dans le document
  - Ex. : Python AND Java
- Comment faire avec un index inversé ?
  - Il faut faire l'intersection de deux listes triées.



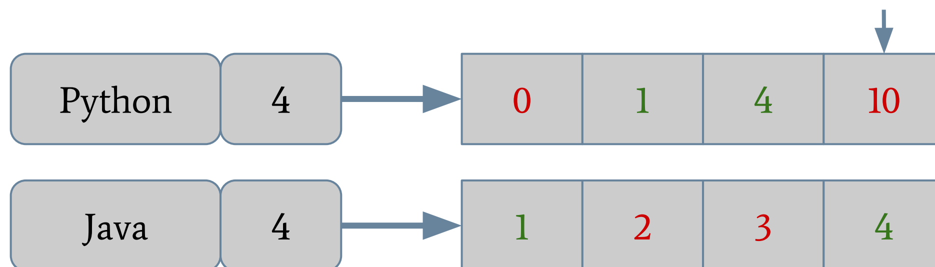
La requête AND

- Une requête q1 AND q2 retourne tous les documents répondant à la requête q1 et q2.
  - Dans le cas simple, on veut que deux termes apparaissent dans le document
  - Ex. : Python AND Java
- Comment faire avec un index inversé ?
  - Il faut faire l'intersection de deux listes triées.



La requête AND

- Une requête q1 AND q2 retourne tous les documents répondant à la requête q1 et q2.
  - Dans le cas simple, on veut que deux termes apparaissent dans le document
  - Ex. : Python AND Java
- Comment faire avec un index inversé ?
  - Il faut faire l'intersection de deux listes triées.



Linéaire en la taille des deux listes (nombre de documents dans le pire des cas) !

La requête OR et NOT

- Une requête q1 OR q2 retourne tous les documents répondant à la requête q1 ou q2.
- Une requête NOT retourne tous les documents ne répondant pas à la requête q.
- On peut généraliser l'algorithme du AND à OR, NOT, OR NOT, et AND NOT.

Requête booléenne

- Une requête booléenne est une requête construite à partir de termes et des opérations AND, OR, et NOT.
  - Ex.: Python AND Java AND NOT Ingénieur

Optimisation de requête

- But : Réduire le nombre de calculs pour répondre à une requête
- Dans le cas où nous avons n AND, dans quel ordre faut-il les traiter ?
  - (Java AND Python) AND Ingénieur ?
  - Java AND (Python AND Ingénieur) ?
  - (Java AND Ingénieur) AND Python ?
- Exemple : Commencer par les sets les plus petits, et ensuite de plus en plus grand
  - La taille d'un AND ne dépassera jamais la taille d'un des membres
  - C'est une des raisons pour laquelle nous gardons la fréquence des documents

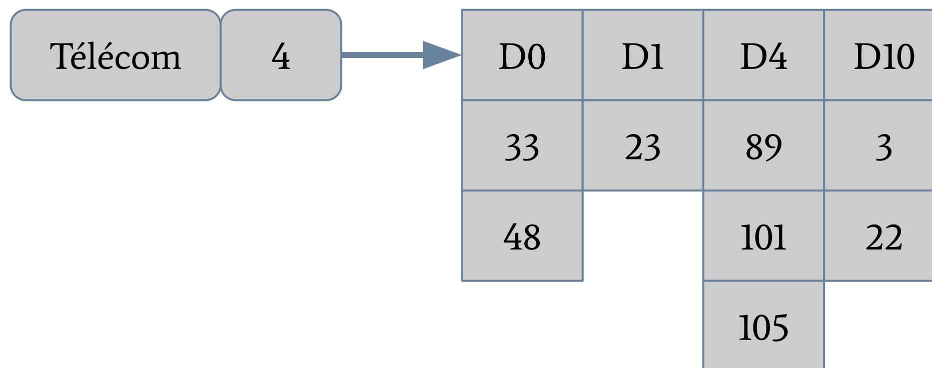
Recherche de segments de phrases et index positionnel

Recherche de segments de phrases

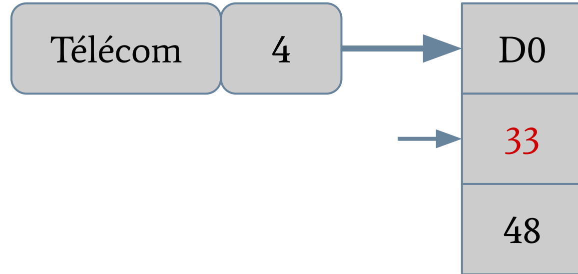
- Comment faire pour chercher une suite de termes ?
  - Très courant. Ex. : New York, Télécom SudParis, Institut Polytechnique de Paris
- On ne peut plus stocker uniquement les documents associés avec un terme !
- Première solution : Stocker des bigrams (suite de deux termes)
  - Prend beaucoup plus de place
  - Ne se généralise pas aux N-grams (prendrait beaucoup trop de place)

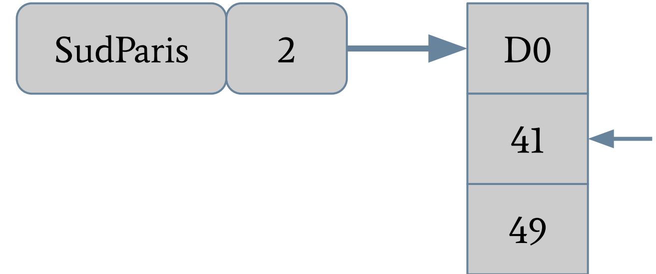
Les index positionnels

- Pour chaque terme, on stocke maintenant tous les documents où il apparaît ET la position du terme dans le dictionnaire.

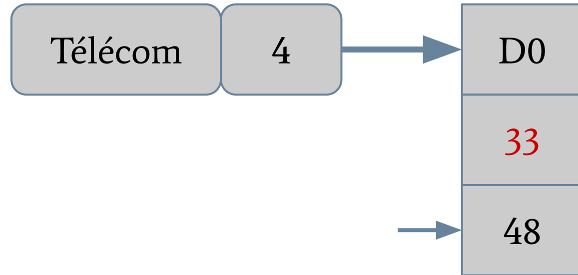


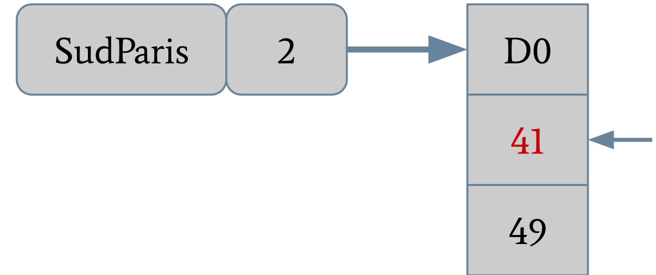
Algorithme de fusion avec des positions



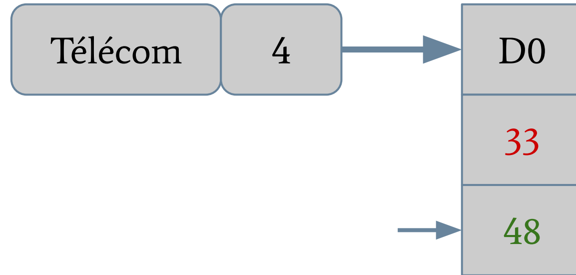


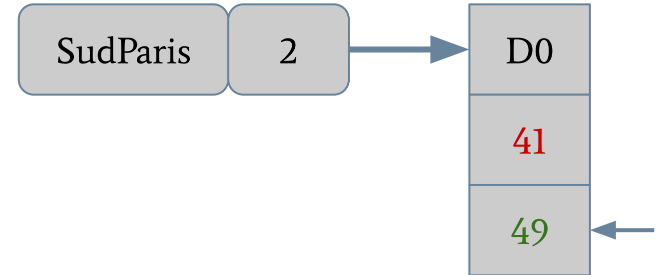
Algorithme de fusion avec des positions





Algorithme de fusion avec des positions





On peut généraliser à des requêtes permettant d'avoir K mots entre deux mots

- Ex: Institut \2 Paris pour Institut Polytechnique de Paris et Institut Océanique de Paris

Désavantage d’un index positionnel

Un index positionnel prend plus de place !

- Il faut maintenant stocker toutes les positions en plus de l’identifiant du document
- Environ 2 à 4 fois plus large qu’un index normal (pour les langues ressemblant au français)

Recherche ordonnée

Désavantages de la recherche booléenne

- Résultats très binaires : soit un document correspond, soit il ne correspond pas
  - “Télécom SudParis” : 175k résultats sur Google
  - “Télécom SudParis gagne la coupe” : 0 résultats
- Souvent pas de résultats ou beaucoup trop de résultats pour les traiter
  - Surtout pour les recherches web !
- Plus difficile à écrire pour des non-initiés
  - Mais très efficaces dans la main d’experts

Recherche ordonnée

- Le système doit maintenant retourner une **liste ordonnée** de résultats. Les "meilleurs" documents apparaissent avant les moins bons.
  - On va donner un score entre 0 et 1 à chaque document pour une requête donnée
  - Il nous faut une métrique valide pour générer ce score
- En pratique, les requêtes sont en **langage naturel**

Score d'un document - Jaccard

On prend le set de terme Q dans la requête et un document D donné et l'on calcule :

$$\frac{|Q \cap D|}{|Q \cup D|}$$

Exemple : Q = "Télécom SudParis", D1 = "Télécom Paris gagne la coupe", D2 = "Télécom SudParis court un marathon"

Jaccard(D1, Q) = 1 / 6 < Jaccard(D2, Q) = 2 / 5

Désavantages de Jaccard

- On ne prend pas en compte la fréquence d'un terme dans un document
  - Plus il est présent, plus il est important
- Tous les termes n'ont pas le même poids
  - Les termes rares sont plus important
  - Ex: Q = "un fieffé menteur" -> "un" n'est pas important, "fieffé" est très important, "menteur" est moyennement important
- Pas de normalisation de longueur
  - Les documents plus longs auront tendance à avoir un score plus bas

Considérer la fréquence des termes

- Le retour de Bag-of-Words (c.f. Cours introduction au TAL)
  - Chaque document d est représenté par un vecteur ayant la taille du vocabulaire. L'index d'un terme t correspond à sa fréquence d'apparition tf(t, d) dans le document.
  - Souvent, on rajoute un log pour limiter l'impact de la fréquence (la pertinence n'augmente pas linéairement avec la fréquence)
- On peut définir le score pour une requête q et un document d par :

$$score(q, d) = \sum_{t \in q \cap d} log(1 + tf_{t,d})$$

On a toujours besoin de notre index !

Poids IDF

- Avec l'approche précédent, nous ne prenons pas en compte l'importance des mots
- On peut définir l'importance des mots par la fréquence d'apparition dans les documents :
    - Un mot qui apparaît dans tous les documents est peu utile
- On définit donc l'IDF (inverse document fréquency) par la formule suivante (avec N le nombre total de documents, et df(t) la fréquence en document du terme t) :

$$idf(t) = log(\frac{N}{df_t})$$

Poids TF-IDF

- On définit le poids TF-IDF comme le mélange d'un terme relié à la fréquence d'un terme dans un document avec l'IDF du terme

$$tfidf(t,d) = log(1 + tf_{t,d}) * log(\frac{N}{df_t})$$

- TF-IDF est utilisé partout dans les moteurs de recherche
- On peut définir un score pour une requête par :

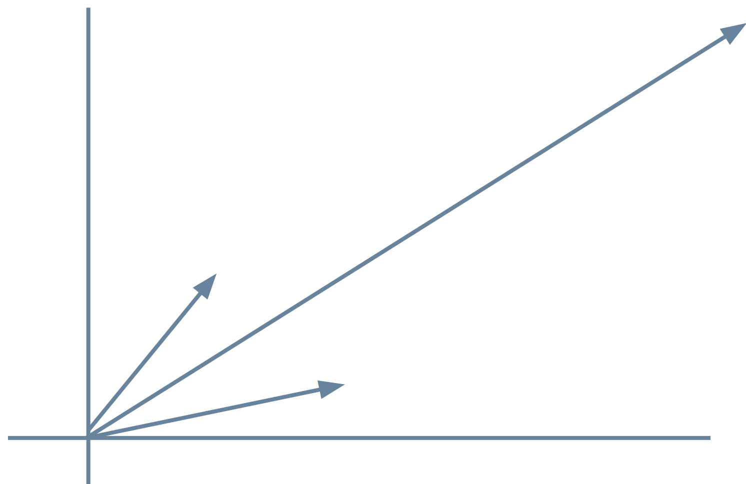
$$score(q,d) = \sum_{t \in q \cap d} tfidf(t,d)$$

Les requêtes comme des vecteurs

- Nous avons toujours des problèmes de normalisation
  - Un grand document aura des fréquences plus élevées qu'un court document
- Les mots dans la requête ont tous la même importance
- Idée :
  - On traite la requête comme un document et on obtient son vecteur TF-IDF
  - Le score d'un document est donné par la distance entre le vecteur de la requête et le vecteur d'un document
- Similarité utilisée : la similarité cosinus

La similarité cosinus

- Idée : les angles sont plus important que la distance euclidienne
  - Comparer un document d avec le document d.d (concaténation du même document)



$$cosine(\vec{u}, \vec{v}) = \frac{\vec{u}\vec{v}}{|\vec{u}||\vec{v}|}$$

Résumé

- Les index sont au cœur de moteur de recherche
- On peut écrire des requêtes booléenne
  - Demande d'étendre les index avec des positions pour chercher des segments de phrases
- En pratique, on veut retourner un résultat ordonner
  - TF-IDF est la solution la plus utilisée
- Il existe des modèles probabilistes plus avancés pour pondérer les résultats d'une recherche
  - Un des plus connu s'appelle **Okapi BM25**

En route vers le TP