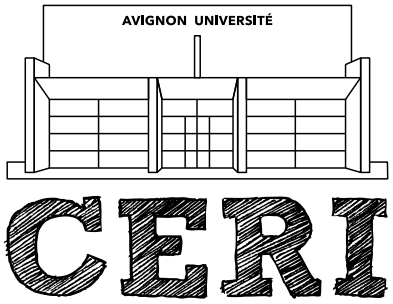


Centre d'Enseignement et de Recherche en Informatique

Partie 4
Recherche positionnelle et approchée

Vincent Labatut

Laboratoire Informatique d'Avignon – LIA EA 4128
vincent.labatut@univ-avignon.fr

2019/20

M2 ILSEN
UE Ingénierie du document et de l'information
UCE3 Indexation & Recherche d'information

AVIGNON UNIVERSITÉ

Plan de la séance

1. Traitement de groupes de mots
   - Groupes de mots
   - Index positionnel
2. Recherche approchée par jokers
   - Recherche approchée
   - Index Permuterm
   - Index de k-grammes
3. Recherche approchée par correction
   - Comparaison de chaînes de caractères
   - Correction phonétique & Soundex

Section 1

Traitement de groupes de mots

AVIGNON
UNIVERSITÉ

Traitement de groupes de mots
Problématique

- Limitation :
  - De nombreux concepts sont décrits par des groupes de mots
    - Ex. : Stanford University
  - Traitement sous forme de conjonction : sensible à The inventor **Stanford** Ovshinsky never went to **university**
  - Problème connexe : proximité des tokens
- Solution :
  - Requête d'expression exacte (phrase query)

  → index actuel insuffisant
- Deux modifications proposées :
  - Manipuler des paires de mots
  - Utiliser un index positionnel

Traitement de groupes de mots
Notion de k-gramme

k-gramme

Sous-séquence de k éléments dans une séquence plus longue.

En NLP, il s'agit :

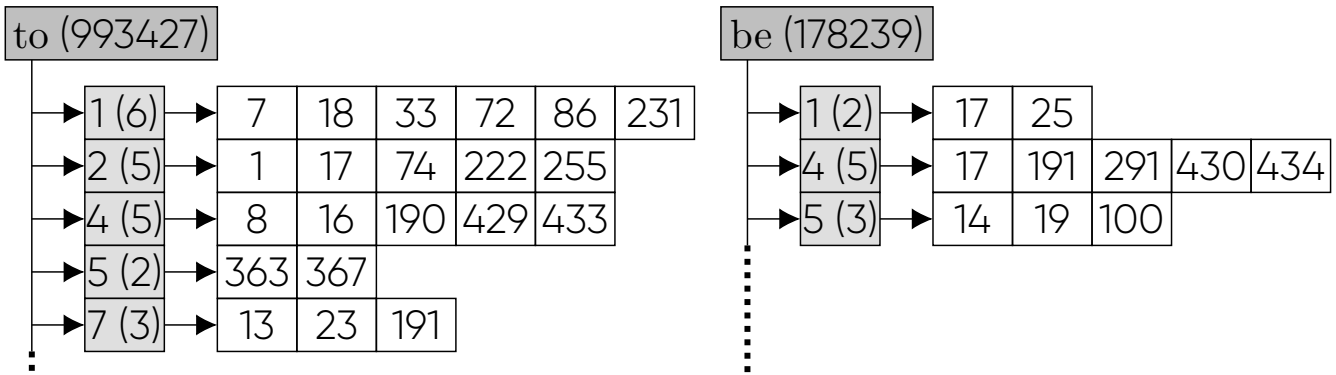
- Soit de k-grammes de mots, par ex. :
  - Séquence : il pleuvait hier il fera beau demain
  - Bigrammes : il pleuvait ; pleuvait hier ; hier il ; il fera ; fera beau...
  - Trigrammes : il pleuvait hier ; pleuvait hier il ; hier il fera...
  - Tetragrammes : il pleuvait hier il ; pleuvait hier il fera...
- Soit de k-grammes de caractères, par ex. :
  - Séquence : sapristi
  - Bigrammes : sa ; ap ; pr ; ri ; st ; ti
  - Trigrammes : sap ; apr ; pri ; ris ; ist ; sti
  - Tetragrammes : sapr ; apri ; pris ; rist ; isti
  - Pentagrammes : sapri ; apris ; prist ; risti

Traitement de groupes de mots
Paires de mots

- Principe :
  - Tokénisation par bigrammes de mots
    - Pour corpus et requêtes
    - Ex. : Friends, Romans, Countrymen
    - → friends romans ; romans countrymen
  - → Entrées de l'index = couples de mots
- Limitations :
  - Faux positifs
    - Ex. : Stanford University Palo Alto
    - → stanford university ; university palo ; palo alto
    - Contenir les 3 couples ≠ contenir la séquence recherchée
  - Index beaucoup plus grand

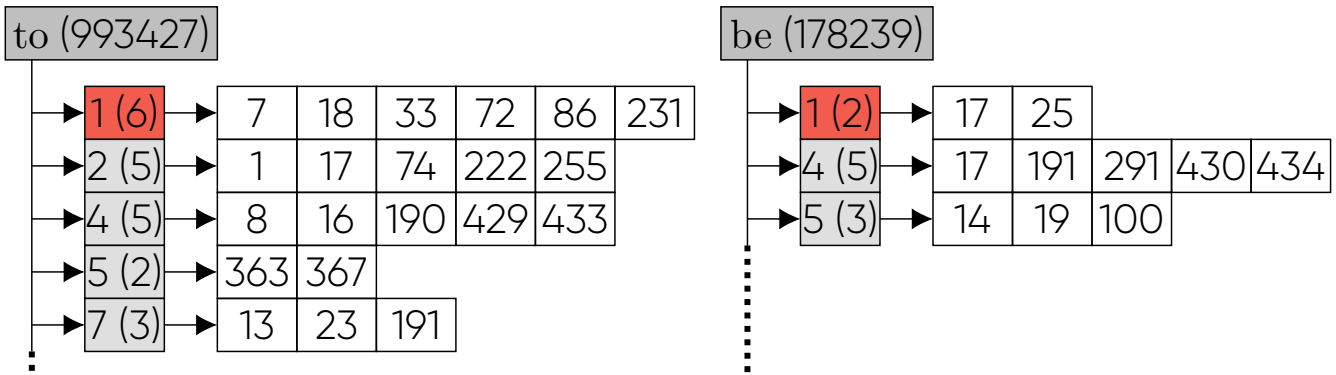
Traitement de groupes de mots
Notion d'index positionnel

- Non-positionnel (jusqu'à présent) : postings = docID
- Positionnel : docID + liste de positions
  - Position exprimée en termes de numéro de token dans le texte
- Traitement des requêtes :
  - Similaire à l'approche classique...
  - ...mais en considérant en plus les positions relatives des mots
- Exemple : $\mathbf{to}_1\ \mathbf{be}_2\ \mathrm{or}_3\ \mathrm{not}_4\ \mathbf{to}_5\ \mathbf{be}_6$



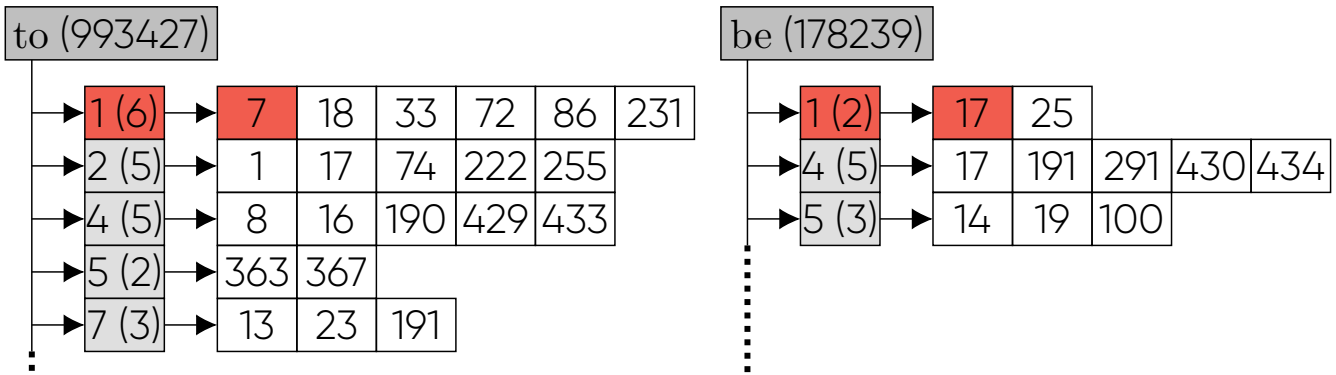
Traitement de groupes de mots
Notion d'index positionnel

- Non-positionnel (jusqu'à présent) : postings = docID
- Positionnel : docID + liste de positions
  - Position exprimée en termes de numéro de token dans le texte
- Traitement des requêtes :
  - Similaire à l'approche classique...
  - ...mais en considérant en plus les positions relatives des mots
- Exemple : $\mathbf{to}_1\ \mathbf{be}_2\ \mathrm{or}_3\ \mathrm{not}_4\ \mathbf{to}_5\ \mathbf{be}_6$



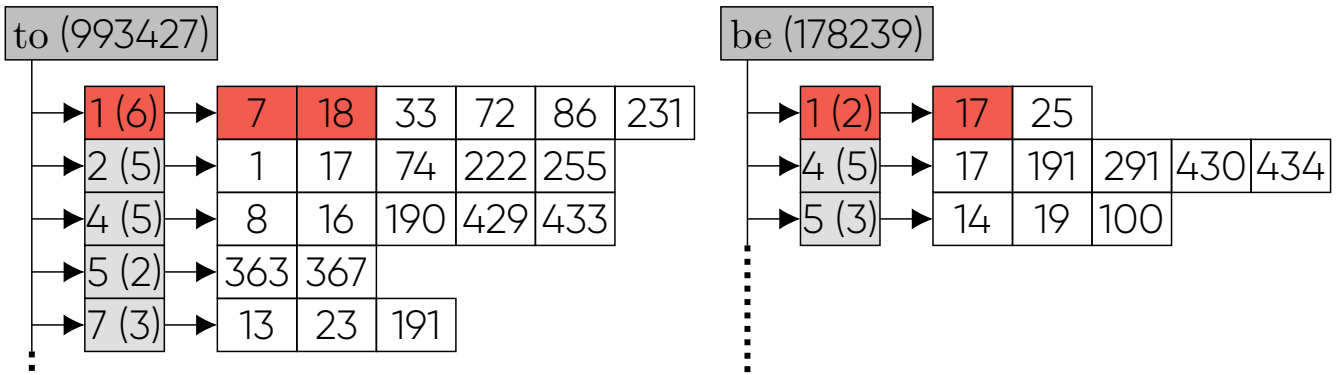
Traitement de groupes de mots
Notion d'index positionnel

- Non-positionnel (jusqu'à présent) : postings = docID
- Positionnel : docID + liste de positions
  - Position exprimée en termes de numéro de token dans le texte
- Traitement des requêtes :
  - Similaire à l'approche classique...
  - ...mais en considérant en plus les positions relatives des mots
- Exemple : $\mathbf{to}_1\ \mathbf{be}_2\ \mathrm{or}_3\ \mathrm{not}_4\ \mathbf{to}_5\ \mathbf{be}_6$



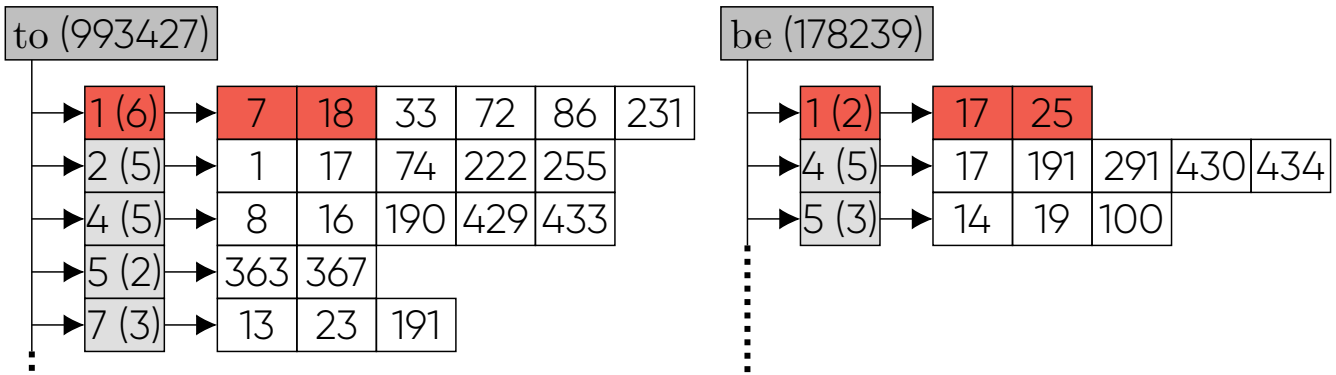
Traitement de groupes de mots
Notion d'index positionnel

- Non-positionnel (jusqu'à présent) : postings = docID
- Positionnel : docID + liste de positions
  - Position exprimée en termes de numéro de token dans le texte
- Traitement des requêtes :
  - Similaire à l'approche classique...
  - ...mais en considérant en plus les positions relatives des mots
- Exemple : $\mathbf{to}_1\ \mathbf{be}_2\ \mathrm{or}_3\ \mathrm{not}_4\ \mathbf{to}_5\ \mathbf{be}_6$



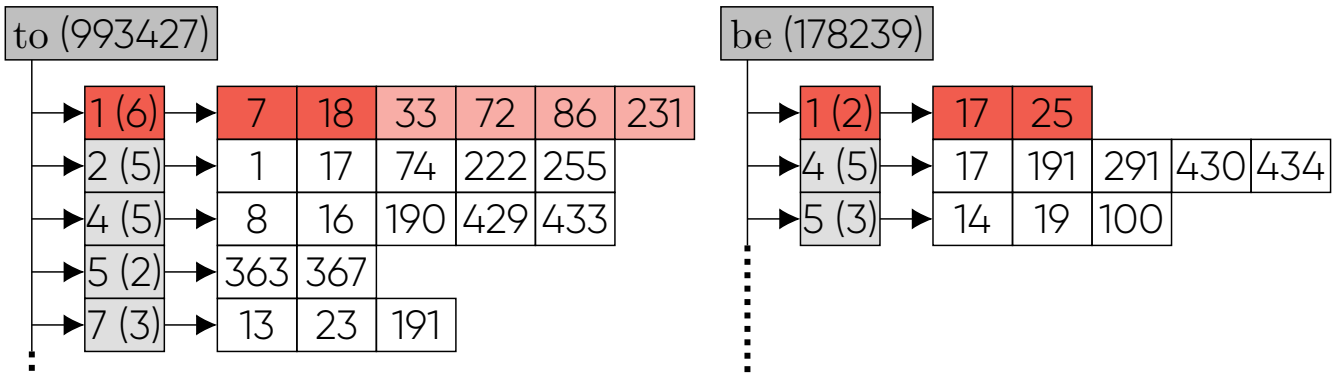
Traitement de groupes de mots
Notion d'index positionnel

- Non-positionnel (jusqu'à présent) : postings = docID
- Positionnel : docID + liste de positions
  - Position exprimée en termes de numéro de token dans le texte
- Traitement des requêtes :
  - Similaire à l'approche classique...
  - ...mais en considérant en plus les positions relatives des mots
- Exemple : $\mathbf{to}_1\ \mathbf{be}_2\ \mathrm{or}_3\ \mathrm{not}_4\ \mathbf{to}_5\ \mathbf{be}_6$



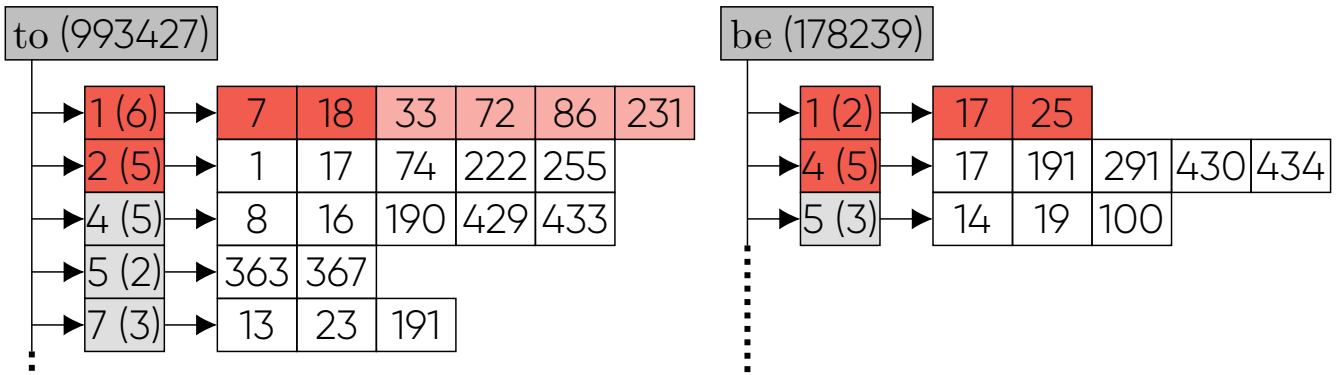
Traitement de groupes de mots
Notion d'index positionnel

- Non-positionnel (jusqu'à présent) : postings = docID
- Positionnel : docID + liste de positions
  - Position exprimée en termes de numéro de token dans le texte
- Traitement des requêtes :
  - Similaire à l'approche classique...
  - ...mais en considérant en plus les positions relatives des mots
- Exemple : $\mathbf{to}_1\ \mathbf{be}_2\ \mathrm{or}_3\ \mathrm{not}_4\ \mathbf{to}_5\ \mathbf{be}_6$



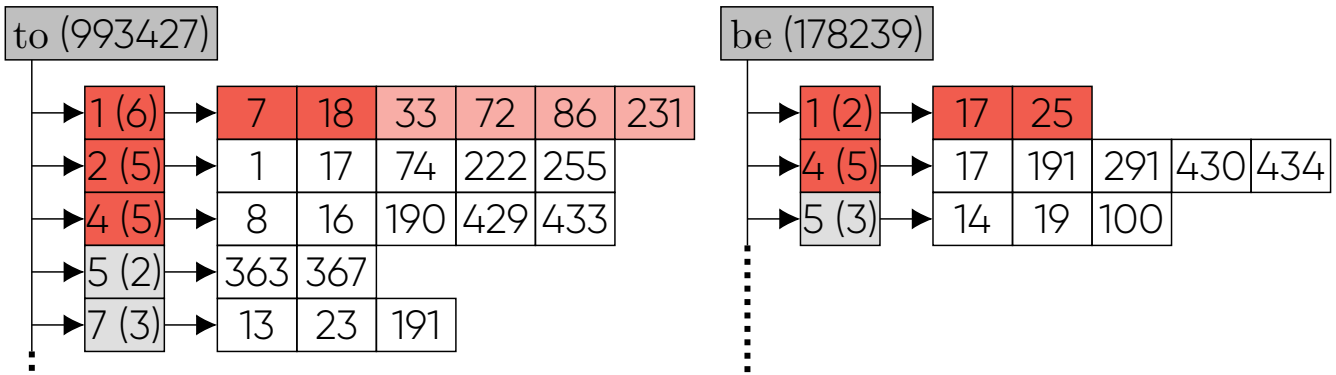
Traitement de groupes de mots
Notion d'index positionnel

- Non-positionnel (jusqu'à présent) : postings = docID
- Positionnel : docID + liste de positions
  - Position exprimée en termes de numéro de token dans le texte
- Traitement des requêtes :
  - Similaire à l'approche classique...
  - ...mais en considérant en plus les positions relatives des mots
- Exemple : $\mathbf{to}_1\ \mathbf{be}_2\ \mathrm{or}_3\ \mathrm{not}_4\ \mathbf{to}_5\ \mathbf{be}_6$



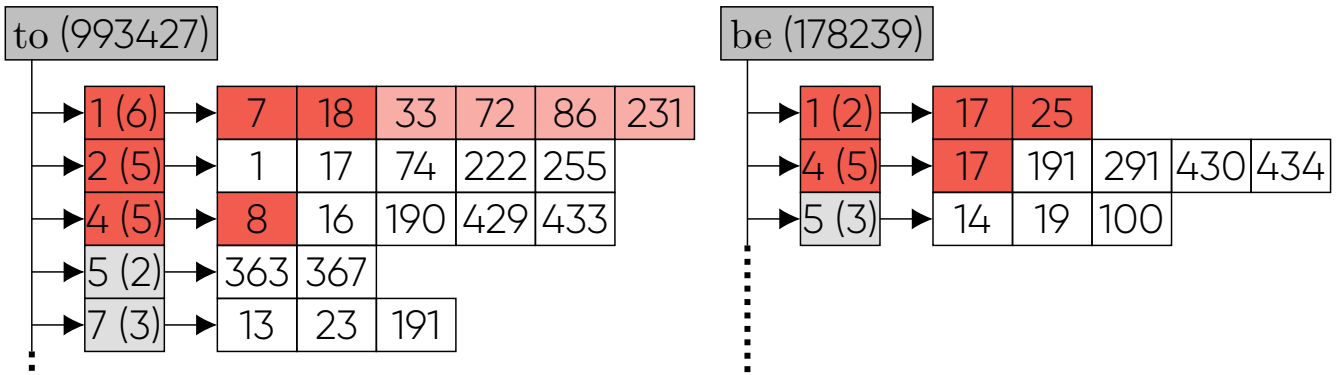
Traitement de groupes de mots
Notion d'index positionnel

- Non-positionnel (jusqu'à présent) : postings = docID
- Positionnel : docID + liste de positions
  - Position exprimée en termes de numéro de token dans le texte
- Traitement des requêtes :
  - Similaire à l'approche classique...
  - ...mais en considérant en plus les positions relatives des mots
- Exemple : $\mathbf{to}_1\ \mathbf{be}_2\ \mathrm{or}_3\ \mathrm{not}_4\ \mathbf{to}_5\ \mathbf{be}_6$



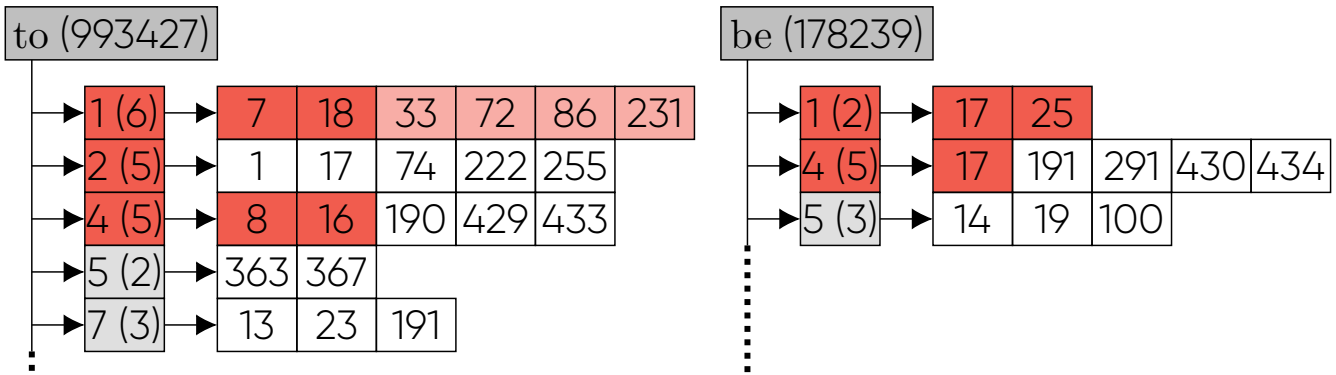
Traitement de groupes de mots
Notion d'index positionnel

- Non-positionnel (jusqu'à présent) : postings = docID
- Positionnel : docID + liste de positions
  - Position exprimée en termes de numéro de token dans le texte
- Traitement des requêtes :
  - Similaire à l'approche classique...
  - ...mais en considérant en plus les positions relatives des mots
- Exemple : $\mathbf{to}_1\ \mathbf{be}_2\ \mathrm{or}_3\ \mathrm{not}_4\ \mathbf{to}_5\ \mathbf{be}_6$



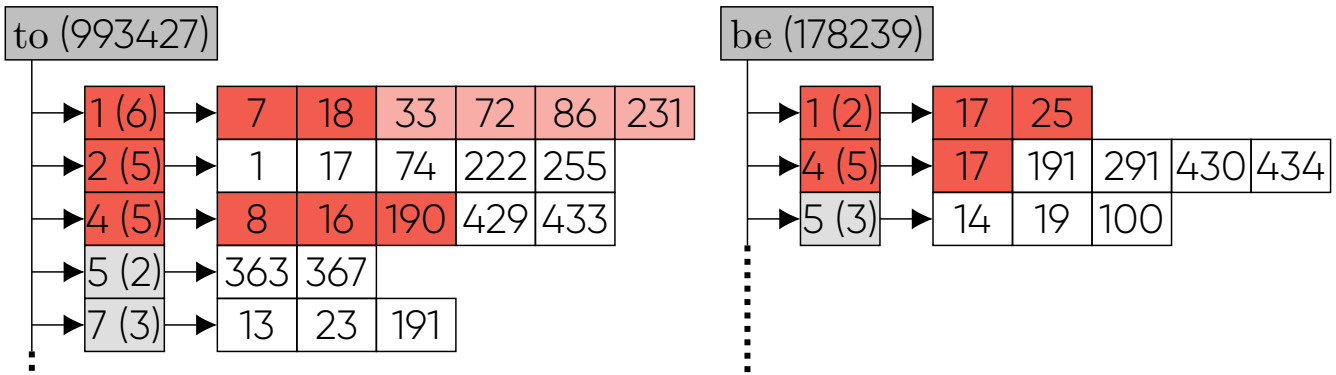
Traitement de groupes de mots
Notion d'index positionnel

- Non-positionnel (jusqu'à présent) : postings = docID
- Positionnel : docID + liste de positions
  - Position exprimée en termes de numéro de token dans le texte
- Traitement des requêtes :
  - Similaire à l'approche classique...
  - ...mais en considérant en plus les positions relatives des mots
- Exemple : $\mathbf{to}_1\ \mathbf{be}_2\ \mathrm{or}_3\ \mathrm{not}_4\ \mathbf{to}_5\ \mathbf{be}_6$



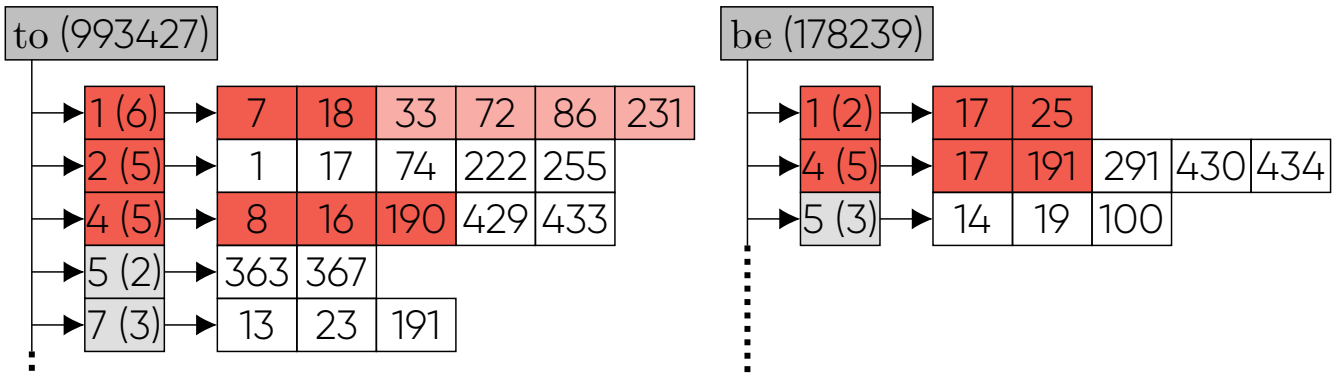
Traitement de groupes de mots
Notion d'index positionnel

- Non-positionnel (jusqu'à présent) : postings = docID
- Positionnel : docID + liste de positions
  - Position exprimée en termes de numéro de token dans le texte
- Traitement des requêtes :
  - Similaire à l'approche classique...
  - ...mais en considérant en plus les positions relatives des mots
- Exemple : $\mathbf{to}_1\ \mathbf{be}_2\ \mathrm{or}_3\ \mathrm{not}_4\ \mathbf{to}_5\ \mathbf{be}_6$



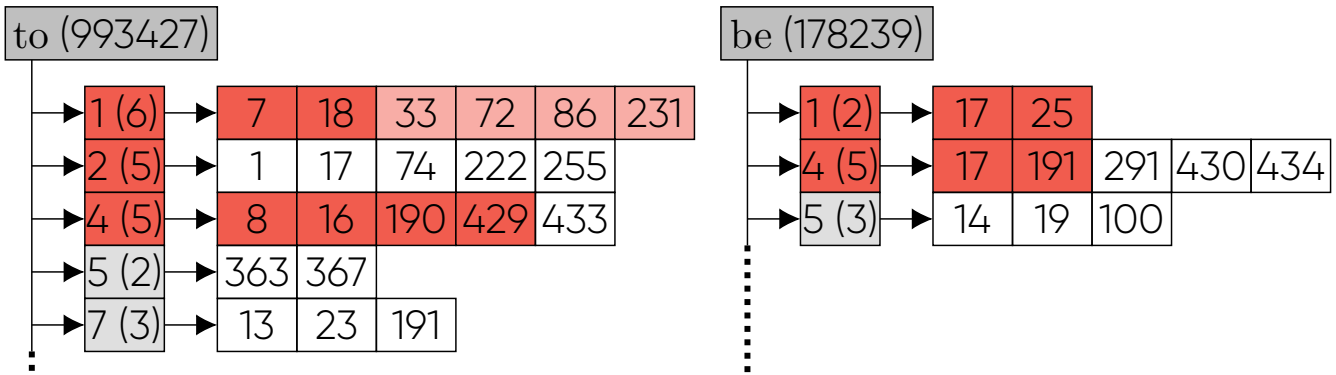
Traitement de groupes de mots
Notion d'index positionnel

- Non-positionnel (jusqu'à présent) : postings = docID
- Positionnel : docID + liste de positions
  - Position exprimée en termes de numéro de token dans le texte
- Traitement des requêtes :
  - Similaire à l'approche classique...
  - ...mais en considérant en plus les positions relatives des mots
- Exemple : $\mathbf{to}_1\ \mathbf{be}_2\ \mathrm{or}_3\ \mathrm{not}_4\ \mathbf{to}_5\ \mathbf{be}_6$



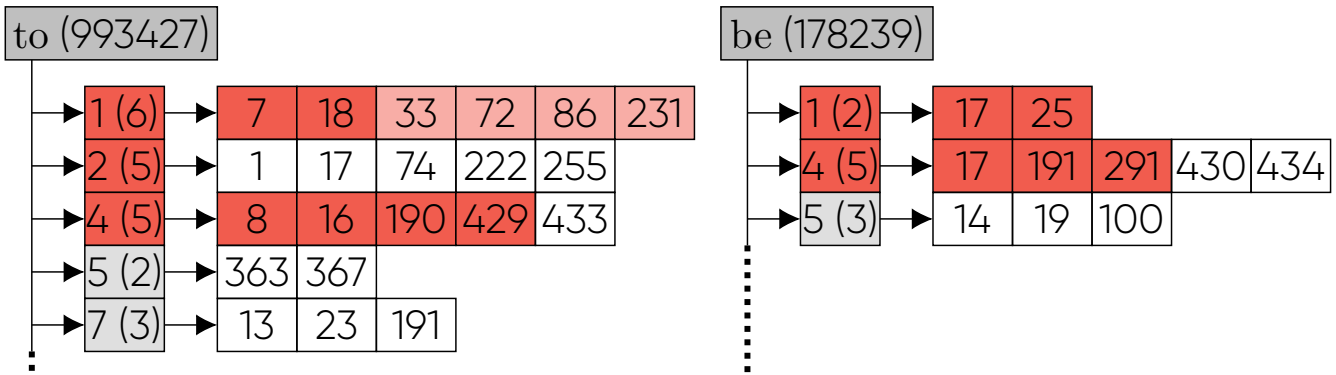
Traitement de groupes de mots
Notion d'index positionnel

- Non-positionnel (jusqu'à présent) : postings = docID
- Positionnel : docID + liste de positions
  - Position exprimée en termes de numéro de token dans le texte
- Traitement des requêtes :
  - Similaire à l'approche classique...
  - ...mais en considérant en plus les positions relatives des mots
- Exemple : $\mathbf{to}_1\ \mathbf{be}_2\ \mathrm{or}_3\ \mathrm{not}_4\ \mathbf{to}_5\ \mathbf{be}_6$



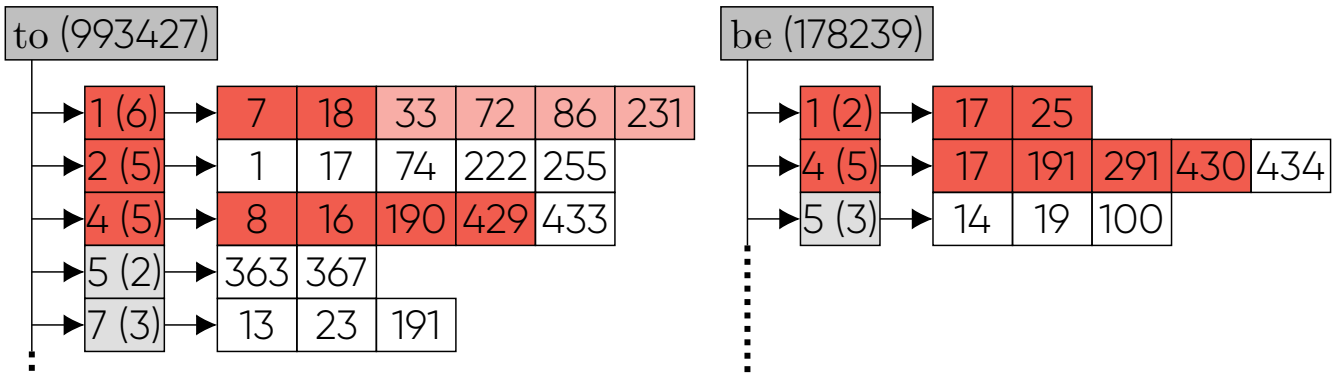
Traitement de groupes de mots
Notion d'index positionnel

- Non-positionnel (jusqu'à présent) : postings = docID
- Positionnel : docID + liste de positions
  - Position exprimée en termes de numéro de token dans le texte
- Traitement des requêtes :
  - Similaire à l'approche classique...
  - ...mais en considérant en plus les positions relatives des mots
- Exemple : $\mathbf{to}_1\ \mathbf{be}_2\ \mathrm{or}_3\ \mathrm{not}_4\ \mathbf{to}_5\ \mathbf{be}_6$



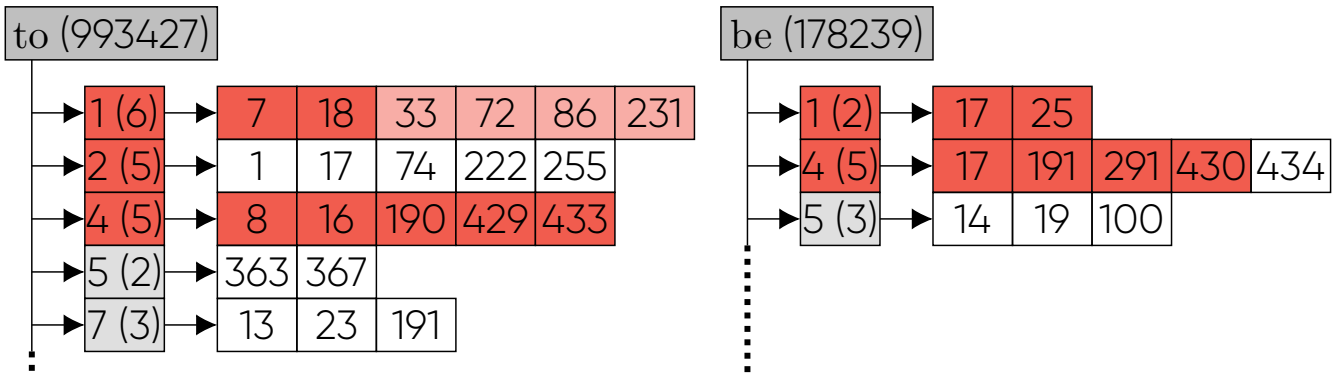
Traitement de groupes de mots
Notion d'index positionnel

- Non-positionnel (jusqu'à présent) : postings = docID
- Positionnel : docID + liste de positions
  - Position exprimée en termes de numéro de token dans le texte
- Traitement des requêtes :
  - Similaire à l'approche classique...
  - ...mais en considérant en plus les positions relatives des mots
- Exemple : $\mathbf{to}_1\ \mathbf{be}_2\ \mathrm{or}_3\ \mathrm{not}_4\ \mathbf{to}_5\ \mathbf{be}_6$



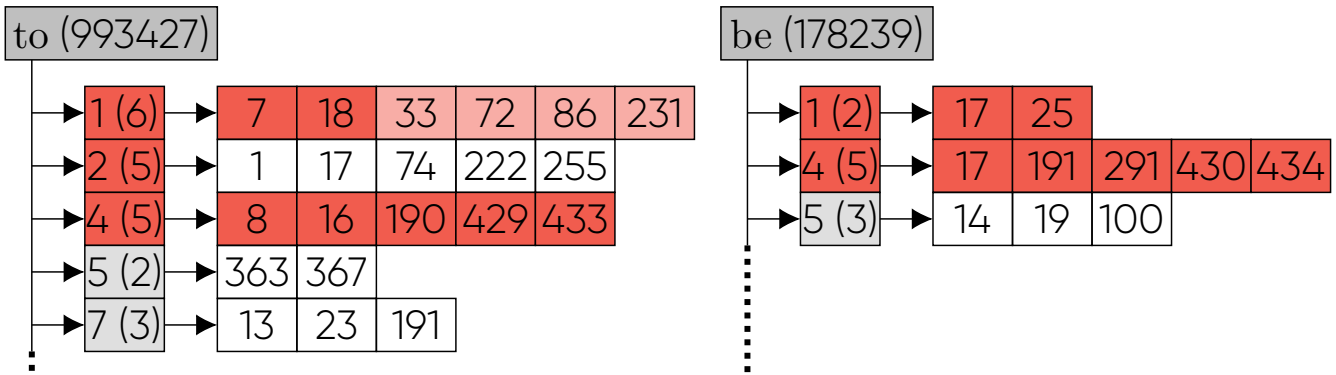
Traitement de groupes de mots
Notion d'index positionnel

- Non-positionnel (jusqu'à présent) : postings = docID
- Positionnel : docID + liste de positions
  - Position exprimée en termes de numéro de token dans le texte
- Traitement des requêtes :
  - Similaire à l'approche classique...
  - ...mais en considérant en plus les positions relatives des mots
- Exemple : $\mathbf{to}_1\ \mathbf{be}_2\ \mathrm{or}_3\ \mathrm{not}_4\ \mathbf{to}_5\ \mathbf{be}_6$



Traitement de groupes de mots
Notion d'index positionnel

- Non-positionnel (jusqu'à présent) : postings = docID
- Positionnel : docID + liste de positions
  - Position exprimée en termes de numéro de token dans le texte
- Traitement des requêtes :
  - Similaire à l'approche classique...
  - ...mais en considérant en plus les positions relatives des mots
- Exemple : $\mathbf{to}_1\ \mathbf{be}_2\ \mathrm{or}_3\ \mathrm{not}_4\ \mathbf{to}_5\ \mathbf{be}_6$



Traitement de groupes de mots
Propriétés des index positionnels

- Plus précis :
  - Renvoie une liste de positions dans le texte
  - (par opposition à une liste de documents)
- Opérateur de proximité : même approche
  - Ex. : $\mathrm{employment}\ /3\ \mathrm{place}$
  - I.e. : place situé 3 mots (ou moins) après $\mathrm{employment}$
- Propriétés de l'index positionnel
  - Occupe plus d'espace mémoire
    - 2 ordres de grandeurs
  - Accès plus lent :
    - Linéaire relativement au nombre de tokens
    - (et non plus au nombre de documents)

Section 2

Recherche approchée par jokers

AVIGNON
UNIVERSITÉ

Recherche approchée par jokers
Limitations de la recherche exacte

- Limitation : rigidité orthographique
  - Requête exacte : pas de faute d'orthographe
  - Requête complète : mots décrits en entier
  - Correspondance exacte : même orthographe dans l'index
- Solutions :
  - Corriger les erreurs dans les requêtes
  - Utiliser des jokers (*, ?, etc.)
  - Comparer phonétiquement

  → nécessité d'une structure de données appropriée pour l'index

Recherche approchée par jokers
Structure du lexique

- Lexique : 2 alternatives
  - Table de hachage
    - Fonction de hachage : termes $\rightarrow$ entiers
    - Recherche en temps constant
    - Impossible de trouver facilement des variantes lexicales du terme
    - Pas de recherche par préfixe
    - Fonction peut être insuffisante si le lexique grandit
  - Arbres de recherche
    - Label = préfixe (mot pour les feuilles)
    - Recherche en temps logarithmique
    - Possible de trouver des variantes
    - Possible de chercher par préfixe
    - Mais insertion d'élément coûteuse ($\rightarrow$ B-trees)
    - Occupation mémoire supérieure à hachage

Recherche approchée par jokers
Notion de joker

Métacaractère, ou joker (eng : wild card)

Caractère spécial se substituant à une **sous-séquence** de caractères dans une **requête**.

La syntaxe exacte dépend du moteur de recherche, mais le caractère * est souvent utilisé pour représenter n'importe quelle sous-séquence (y compris ∅)
Les jokers sont utiles dans les cas suivants :

- Utilisateur incertain de l'orthographe
  - Ex. : silhouette vs. sihlouette
- Utilisateur sait qu'il existe plusieurs orthographes
  - Ex. : clé vs. clef
- Utilisateur vise plusieurs mots proches
  - Ex. : algorithmique et algorithme

Recherche approchée par jokers
Exploitation de l'arbre de recherche

- Recherche de préfixe sur arbre : $\mathrm{part}*$
  - Naviguer jusqu'au nœud correspondant au préfixe
  - Parcourir en partant de ce nœud, jusqu'à rencontrer un nœud ne correspondant plus au préfixe
  - Ex. : pour $\mathrm{mon}*$: aller à mon- et parcourir jusqu'à moo-
- Recherche de suffixe : $*\mathrm{lier}$
  - Même chose sur un arbre construit à l'envers (ordre alphabétique ascendant)
- Recherche d'infixe : $\mathrm{part}*\mathrm{lier}$
  - Calculer l'ensemble des termes pour $\mathrm{part}*$
  - Calculer l'ensemble des termes pour $*\mathrm{lier}$
  - Prendre leur intersection

  $\rightarrow$ Approche coûteuse

Recherche approchée par jokers
Notion de Permuterm

- Alternative : index Permuterm
  - Principe : transformer la requête de manière à placer le * à la fin
  - Caractère $ marque la fin du terme
  - Lexique contient toutes les rotations d'un terme
    - Ex. : $\text{racine} \rightarrow \text{racine\$}, \text{\$racine}, \text{e\$racin}, \text{ne\$raci}, \text{ine\$rac}, \text{cine\$ra}, \text{acine\$r}$
- Transformation de la requête :
  - Rotation amenant l'éventuel * à la fin
  - $\text{racine} \rightarrow \text{racine\$}$
  - $\text{*ine} \rightarrow \text{ine\$*}$; $\text{rac*} \rightarrow \text{\$rac*}$
  - $\text{r*e} \rightarrow \text{e\$r*}$; $\text{*ci*} \rightarrow \text{ci*}$
- Bilan :
  - Plus rapide qu'approche précédente
  - Mais lexique beaucoup plus grand
    - Ex. : $\times 10$ pour un corpus en anglais

Recherche approchée par jokers
Index de k-grammes de caractères

- Alternative : index de k-gramme de caractères
  - On utilise \$ pour noter le début ou la fin d'un terme.
  - Ex. pour $k = 3$: $\mathrm{chat} \rightarrow \mathrm{\$ch}, \mathrm{cha}, \mathrm{hat}, \mathrm{at\$}$
  - Index supplémentaire :
    - Lexique = tous les k-grammes des termes originaux
    - Chacun pointe vers la liste des termes originaux le contenant
    - Ex. : ntr pointe vers $\mathrm{a}\mathbf{ntr}\mathrm{e}$, $\mathrm{e}\mathbf{ntr}\mathrm{e}$, $\mathrm{c}\mathbf{ntr}\mathrm{e}$...
- Utilisation :
  - $\mathrm{ra}* \rightarrow$ on cherche le trigramme $\mathrm{\$ra}$
  - $*\mathrm{ne} \rightarrow$ on cherche le trigramme $\mathrm{ne\$}$
  - $\mathrm{ra}*\mathrm{ne} \rightarrow$ intersection des résultats de $\mathrm{\$ra}$ et $\mathrm{ne\$}$
- Complication pour $\mathrm{red}*$:
  - $\mathrm{\$re}$ et red vont produire (entre autres) $\mathbf{retired}$

  $\rightarrow$ besoin d'un filtrage supplémentaire (simple comparaison)

Section 3

Recherche approchée par correction

AVIGNON
UNIVERSITÉ

Recherche approchée par correction
Distance de Levenshtein

Distance édit

Permet de comparer deux **séquences** en déterminant le nombre **minimal** d'**opérations** nécessaires pour passer de l'une à l'autre.

Distance de Levenshtein

La distance de Levenshtein [Lev66] est une distance **édit** autorisant les opérations **insérer**, **supprimer** et **remplacer**.

Utilisée en NLP pour comparer les chaînes de caractères :

- dog vs. $\mathrm{do} \rightarrow 1$
- cat vs. $\mathrm{cart} \rightarrow 1$
- cat vs. $\mathrm{act} \rightarrow 2$
- niche vs. $\mathrm{chiens} \rightarrow 5$

Recherche approchée par correction
Notations

- Calcul de la distance de Levenshtein :
  - Méthode récursive naïve [Lev66]
  - Algorithme de Wagner–Fischer [WF74]
- Notations :
  - $|s|$: longueur de la chaîne s, ex. $|\text{"truc"}| = 4$
  - $\texttt{init}(s)$: chaîne privée de son dernier caractère
    - Ex. $\texttt{init}(\text{"truc"}) = \text{"tru"}$
  - $\texttt{last}(s)$: dernier caractère
    - Ex. $\texttt{last}(\text{"truc"}) = \text{"c"}$
  - $[P]$: crochets d'Iverson
    - Ex. $[1 > 4] = 0$, mais $[1 < 4] = 1$

Recherche approchée par correction
Méthode naïve pour calculer Levenshtein

```
1  Function lev(s1,s2) :
       Input : s1,s2 : String
       Output : d : Integer
2      if s1 = ∅ then
3          d ← |s2|
4      else if s2 = ∅ then
5          d ← |s1|
6      else
7          d ← min (lev (init (s1), s2) + 1 ;
8                   lev (s1, init (s2)) + 1 ;
9                   lev (init (s1), init (s2)) + [last (s1) ≠last (s2)])
10     end if
11     return d
12 return
```

Algorithme 1 – Méthode naïve pour calculer Levenshtein [Lev66].

Recherche approchée par correction
Limites de la méthode naïve

- Complexité temporelle : $O(3^{\min(|s_1|;|s_2|)})$
  - Cause de la complexité élevée : opérations redondantes
  - (de nombreux calculs sont effectués plusieurs fois)
- Amélioration : Algorithme de Wagner-Fischer
  - Méthode à base de programmation dynamique
- Principe :
  - Utiliser une matrice entière $\mathbf{D}$ de taille $|s_1| \times |s_2|$ pour stocker les résultats des calculs intermédiaires
    - Initialisée avec des valeurs négatives
  - Chaque élément $\mathbf{D}_{ij}$ contient la distance entre :
    - La sous-chaîne contenant les i premiers caractères de s_1
    - Et celle contenant les j premiers caractères de s_2
  - Si on a besoin d'une distance déjà calculée, on la récupère dans la matrice au lieu de la recalculer

Recherche approchée par correction
Algorithme de Wagner-Fisher

1 **Function** `lev`$(s_1, s_2, \mathbf{D})$ **:**
  Input : s_1,s_2 : String, $\mathbf{D}$: Integer Matrix
  Output : d : Integer
2 **if** $s_1 = \varnothing$ **then**
3 $d \leftarrow |s_2|$
4 **else if** $s_2 = \varnothing$ **then**
5 $d \leftarrow |s_1|$
6 **else if** $\mathbf{D}_{|s_1|,|s_2|} \geq 0$ **then**
7 $d \leftarrow \mathbf{D}_{|s_1|,|s_2|}$
8 **else**
9 $d \leftarrow \min\big($`lev` $(`$`init` $(s_1), s_2) + 1$;
10 `lev` $(s_1,$ `init` $(s_2)) + 1$;
11 `lev` $(`$`init` $(s_1),$ `init` $(s_2)) + [$`last` $(s_1) \neq$`last` $(s_2)]\big)$
12 $\mathbf{D}_{|s_1|,|s_2|} \leftarrow d$
13 **end if**
14 **return** d
15 **return**

Algorithme 2 – Algorithme de Wagner-Fischer [WF74].

Recherche approchée par correction
Utilisation de Levenshtein pour la recherche approchée

- Correction :
  - Remplacer par le terme le plus proche dans le lexique
- Recherche exhaustive = trop coûteuse

→ Approche heuristique

- Restriction aux mots de même initiale
- Restriction aux termes commençant par une rotation du mot
- Restriction aux termes ayant un certain nombre de k-grammes en commun
  - Déterminer tous les k-grammes du mot requête
  - Obtenir tous les termes ayant des k-grammes communs (via l'index k-gramme des jokers)
  - Filtrer ceux qui ont n k-grammes communs (par intersection des ensembles obtenus)
  - Autre possibilité : coefficient de Jaccard $\cap/\cup$

Recherche approchée par correction
Correction phonétique

- Correction phonétique :
  - Requête tapée "à l'oreille"
    - Ex. d'utilisation : recherche de noms propres
  - Principe :
    - Chaque mot est remplacé par un code représentant sa prononciation
    - Deux homophones (mots de même prononciation) ont le même code
    - On compare le code du mot requête à ceux des termes
- Exemple : code Soundex [Rep02]
  - Composé d'1 lettre et 3 chiffres : `A123`

Recherche approchée par correction
Algorithme du Soundex

- Algorithme :
  1. Garder la première lettre du mot
  2. Remplacer toutes les voyelles et muettes (H, W) par zéro
  3. Remplacer toutes les consonnes par leur numéro de classe :
     - B, F, P, V = 1
     - C, G, J, K, Q, S, X, Z = 2
     - D, T = 3
     - L = 4
     - N, M = 5
     - R = 6
  4. Remplacer les valeurs consécutives égales par une seule occurrence
  5. Supprimer les zéros
  6. Renvoyer les 4 premiers caractères
  7. Compléter par des zéros, si nécessaire
- Exemple : Hermann $\rightarrow$ `H655`

Section 4

Conclusion

AVIGNON
UNIVERSITÉ

Concepts abordés dans cette partie

- Index positionnel
- Opérateur de proximité
- Distance édit
- Distance de Levenshtein
- Algorithme de Wagner-Fisher
- Joker, métacaractère
- Permuterm
- k-gramme de mots
- k-gramme de caractères
- Soundex

Lectures recommandées

[MRS08] *Introduction to Information Retrieval*, chapitre 3.

[BCC10] *Information Retrieval : Implementing and Evaluating Search Engines*, chapitre 2.

[BR11] *Modern Information Retrieval : The Concepts and Technology behind Search*, chapitre 9.

[AG13] *Recherche d'information - Applications, modèles et algorithmes*, chapitre 3.

[CMS15] *Search Engines : Information Retrieval in Practice*, chapitre 5.

Références bibliographiques I

[AG13] M.-R. Amini et É. Gaussier. *Recherche d'information - Applications, modèles et algorithmes.* Paris, FR : Eyrolles, 2013. url : https://www.eyrolles.com/Informatique/Livre/recherche-d-information-9782212673760/.

[BR11] R. Baeza-Yates et B. Ribeiro-Neto. *Modern Information Retrieval : The Concepts and Technology behind Search.* 2nd Edition. Boston, USA : Addison Wesley Longman, 2011. url : http://people.ischool.berkeley.edu/~hearst/irbook/.

[BCC10] S. Büttcher, C. L. A. Clarke et G. V. Cormack. *Information Retrieval : Implementing and Evaluating Search Engines.* Cambridge, USA : MIT Press, 2010. url : http://www.ir.uwaterloo.ca/book/.

[CMS15] W. B. Croft, D. Metzler et T. Strohman. *Search Engines : Information Retrieval in Practice.* Pearson, 2015. url : http://www.search-engines-book.com/.

Références bibliographiques II

[Lev66] V. I. Levenshtein. « Binary codes capable of correcting deletions, insertions, and reversals ». In : *Soviet Physics Doklady* 10.8 (1966), p. 707–710. url : https://nymity.ch/sybilhunting/pdf/Levenshtein1966a.pdf.

[MRS08] C. D. Manning, P. Raghavan et H. Schütze. *Introduction to Information Retrieval*. New York, USA : Cambridge University Press, 2008. url : http://www-nlp.stanford.edu/IR-book/.

[Rep02] D. J. Repici. *Understanding Classic SoundEx Algorithms*. Creativyst. 2002. url : http://www.creativyst.com/Doc/Articles/SoundEx1/SoundEx1.htm.

[WF74] R. A. Wagner et M. J. Fischer. « The String-to-String Correction Problem ». In : *Journal of the Association for Computing Machinery* 21.1 (1974), p. 168–173. doi : 10.1145/321796.321811.