Partie 2
Recherche booléenne

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
CERI
Centre d'Enseignement et de Recherche en Informatique

AVIGNON UNIVERSITÉ

Plan de la séance

Section 1

Exemple et approche naïve

AVIGNON
UNIVERSITÉ

Exemple récurrent : Shakespeare

Collection : pièces de théâtre de Shakespeare

1. The Two Gentlemen of Verona
2. The Taming of the Shrew
3. Henry VI, Parts 1–3
6. Titus Andronicus
7. Richard III
8. The Comedy of Errors
9. Love's Labour's Lost
10. Richard II
11. Romeo & Juliet
12. A Midsummer Night's Dream
13. King John
14. The Merchant of Venice
15. Henry IV, Parts 1–2
17. The Merry Wives of Windsor
18. Much Ado About Nothing
19. Henry V
20. Julius Caesar
21. As You Like It
22. Hamlet
23. Twelfth Night
24. Troilus and Cressida
25. Measure for Measure
26. Othello
27. All's Well That Ends Well
28. King Lear
29. Timon of Athens
30. Macbeth
31. Antony & Cleopatra
32. Pericles, Prince of Tyre
33. Coriolanus
34. The Winter's Tale
35. Cymbeline
36. The Tempest
37. Henry VIII

Question : quelles pièces contiennent les mots Brutus et Caesar, mais pas Calpurnia ?

Méthode naïve : le grepping

- Description
  - Parcourir chaque texte mot-à-mot
  - Relever pour chaque pièce si les mots apparaissent
- Avantage :
  - Facilement automatisable
- Inconvénients :
  - Trop lent : $|\text{Shakespeare}| \ll |\text{Web}|$
  - Pas flexible : positionnement relatif des mots ?
  - Pas nuancé : pertinence relative des documents ?

Section 2

Notion d'index

AVIGNON
UNIVERSITÉ

Notion d'index
Définition générique

- Observation :
  - Parcourir les documents à la demande est lent, inefficace
- Solution :
  - Prétraiter les documents pour en extraire l'information pertinente
  - Construire une structure de données contenant cette information

Index

Structure de données représentant les informations **pertinentes** pour la RI contenues dans le corpus, et ce de façon **compacte** et **accessible**.

- Pertinence : stocker l'information nécessaire au traitement des requêtes
- Compacité : minimiser l'espace occupé
- Accessibilité : minimiser le temps d'accès

Matrice d'incidence

Matrice rectangulaire binaire $T \times D$ représentant un corpus de T termes et D documents. Chaque valeur indique si un terme est **présent** (1) ou **pas** (0) dans un document.

- Forme la plus basique d'index
- Terme ≈ mot indexé

	Antony & Cleopatra	*Julius Caesar*	*The Tempest*	*Hamlet*	*Othello*	*Mac Beth*	...
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	0	0	...
...	...	...	...	...	...	...	...

Notion d'index
Représentation sac-de-mots *simplifiée*

- Principe :
  - Chaque document est vu comme un sac-de-mots
  - Requêtes combinant termes avec opérateurs booléens

Modèle sac-de-mots *simplifié* (eng : Bag-of-words)

Soit $\mathcal{T}$ l'ensemble des termes du corpus, tel que $\mathcal{T} = \{t_1, ..., t_T\}$, où les t_j sont les T termes constituant le lexique ($1 \leq j \leq T$).
Un **document** d est représenté par l'**ensemble des termes** qu'il contient, en ignorant toute autre information, y compris leur ordre dans le texte. On a : $d = \{t_{j_1}, ..., t_{j_M}\}$, où M est le nombre de termes constituant le document ($0 \leq i \leq T$).

- Exemples : $d_1 = \{t_1, t_2, t_{99}\}$; $d_{12} = \{t_1, t_{75}, t_{126}, t_{127}\}$
- Limite : $\text{Sarkozy bat Hollande}$ et $\text{Hollande bat Sarkozy}$ représentés de la même façon

Modèle de recherche booléen

Une **requête** est exprimée sous la forme d'une **formule logique propositionnelle**. Les documents **pertinents** sont ceux dont le sac-de-mots **respecte** la formule.

Exemples :

- Documents : $d_1 = \{t_1, t_2, t_{99}\}$, $d_2 = \{t_1, t_{75}, t_{126}, t_{127}\}$
- Requête : $t_1 \wedge (t_{13} \vee \neg t_{75})$
- Résultat : $R = \{d_1\}$

Notion d'index
Modèle booléen et matrice d'incidence

- Exemple : Brutus ∧ Caesar ∧ ¬Calpurnia
  - Chaque terme est caractérisé par un vecteur binaire
  - On applique les opérations bit à bit :

```
110100 & 110111 & !010000 = 110100 & 101111 = 100100
```

  - Résultat : *Antony & Cleopatra* ; *Hamlet*

	Antony & Cleopatra	*Julius Caesar*	*The Tempest*	*Hamlet*	*Othello*	*Mac Beth*	...
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	0	0	...
...	...	...	...	...	...	...	...

Notion d'index
Limites de la matrice d'incidence

- Matrice d'incidence → problème de taille
  - Shakespeare : 37 pièces × 32 000 termes = 1 184 000 valeurs (≈150 ko)
  - Corpus standard :
    - $D \approx 10^6$ documents
    - Chacun fait 2–3 pages (≈1 000 mots)
    - $T \approx 500\,000$ termes
    - Total : 5×10^{11} valeurs (≈60 Go)
- Mais cette matrice est creuse
  - Contient $\approx 99{,}8\%$ de zéros

  → une meilleure représentation est possible

Section 3

Fichier inverse

AVIGNON
UNIVERSITÉ

Fichier inverse
Définitions préliminaires I

Lexique

Ensemble $\mathcal{T} = \{t_1, ..., t_T\}$ des **termes** présents dans le corpus $\mathcal{C}$.

Dans un fichier inverse, les termes du lexique sont classés par ordre lexicographique.

Ordre lexicographique

Généralisation de l'ordre **alphabétique** tenant compte des caractères **non-littéraux** : chiffres, ponctuation, etc.

Fichier inverse
Définitions préliminaires II

DocID

Expression permettant d'identifier un document de façon **unique** dans le corpus.

Concrètement, on utilise le numéro du document dans le corpus, i.e. i pour d_i.

Posting

Référence à un **document** (via son docID), possiblement complétée d'autres informations concernant ce document.

Fichier inverse
Définition d'un fichier inverse

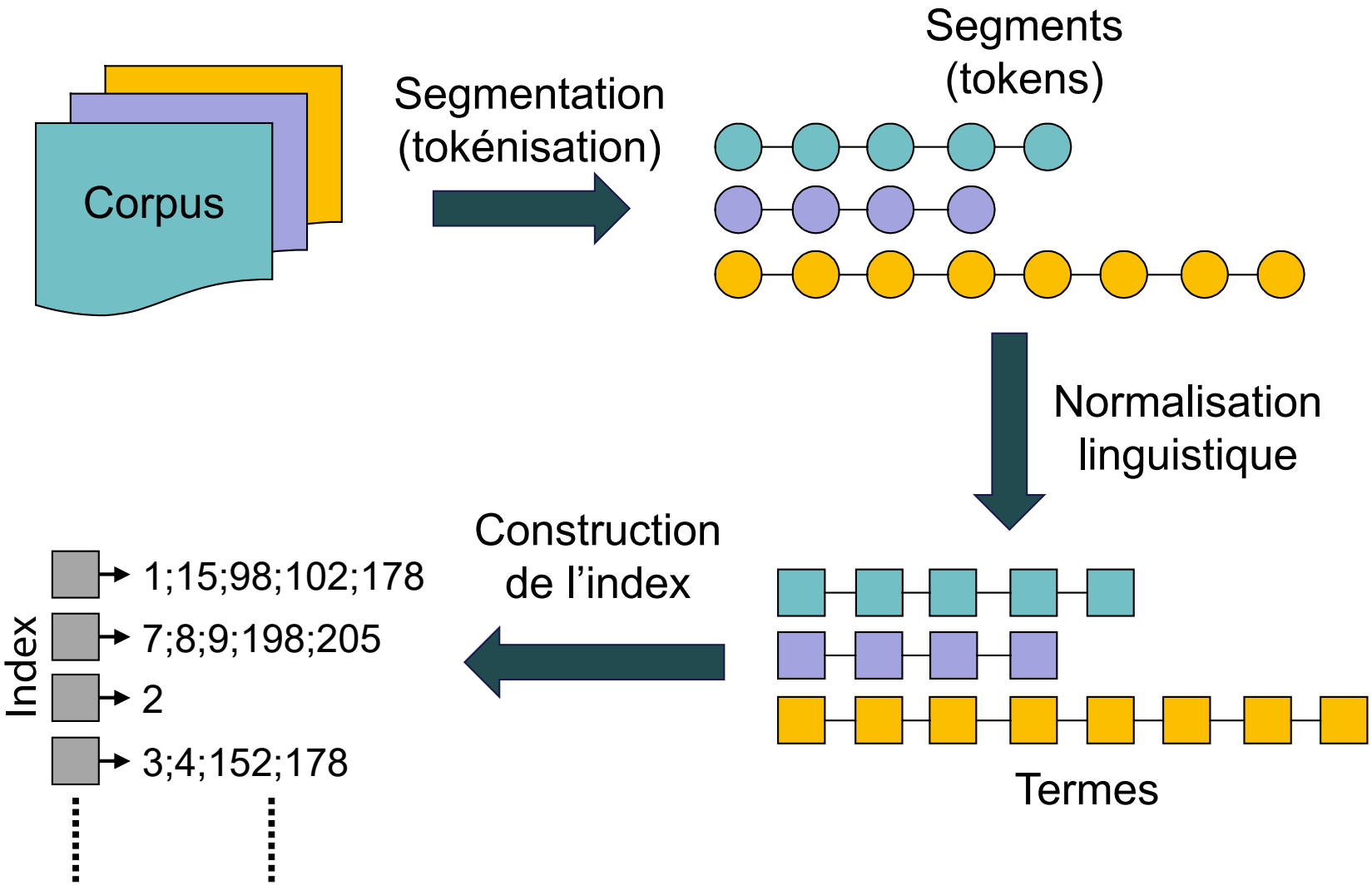
Fichier inverse (ou fichier inversé)

Associe à chaque **terme** t_j du lexique un **ensemble de postings** ℓ_j correspondant aux documents contenant le terme. Les postings sont **ordonnés** par docID croissant.

Étapes du processus d'indexation :

1. Constitution du corpus
2. Segmentation du texte (tokénisation)
3. Normalisation linguistique
4. Construction de l'index

Fichier inverse
Description du processus d'indexation



Fichier inverse
Exemple

1. Constitution du corpus

   Friends, Romans, countrymen. So let it be with Caesar...

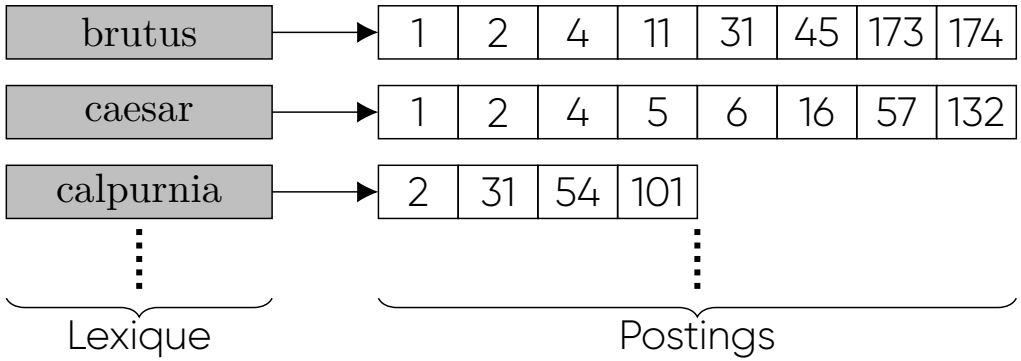
2. Segmentation du texte

   Friends | Romans | countrymen | So | let | it | be | with | Caesar ...

3. Normalisation linguistique

   friend | roman | countryman | so | let | it | be | with | caesar ...

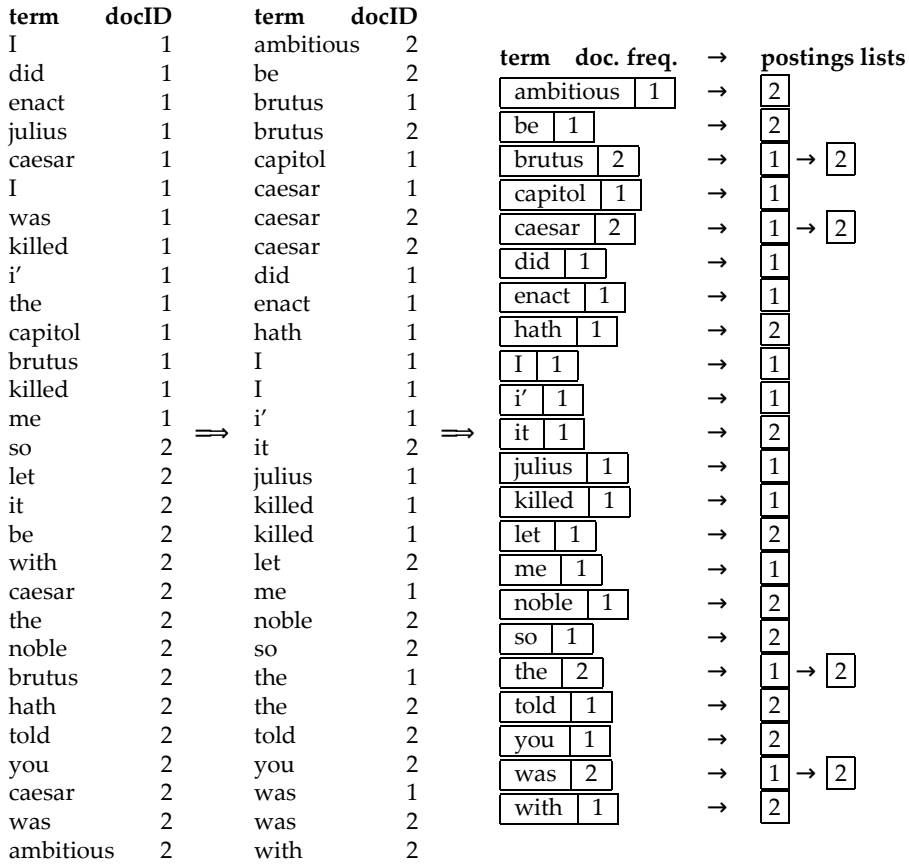
4. Construction de l'index



Fichier inverse
Étapes de construction de l'index

Indexation simple :

1. Parcourir chaque fichier : mettre à jour une liste de paires (terme,docID) (*gauche*)
2. Trier la liste par terme puis docID (*centre*)
3. Fusionner les occurrences multiples dans le *même* document
4. Grouper les occurrences multiples dans des documents *différents* pour obtenir l'index (*droite*)



Fichier inverse
Observations sur la construction de l'index

- Importance de l'ordonnancement
- Fréquence de document des termes $df(t)$ encodée dans le lexique
- Représentation des postings (implémentation)

Fréquence de document d'un terme

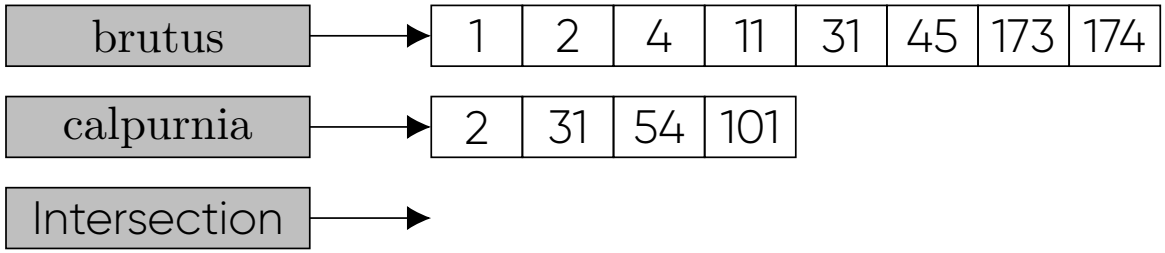
La fréquence de document d'un terme $t \in \mathcal{T}$ est le **nombre** de **documents** du corpus $\mathcal{C}$ contenant ce terme, noté $df(t)$.
Formellement :

$$df(t) = |\{d \in \mathcal{C} : t \in d\}|,$$

où $|...|$ dénote la cardinalité d'un ensemble.

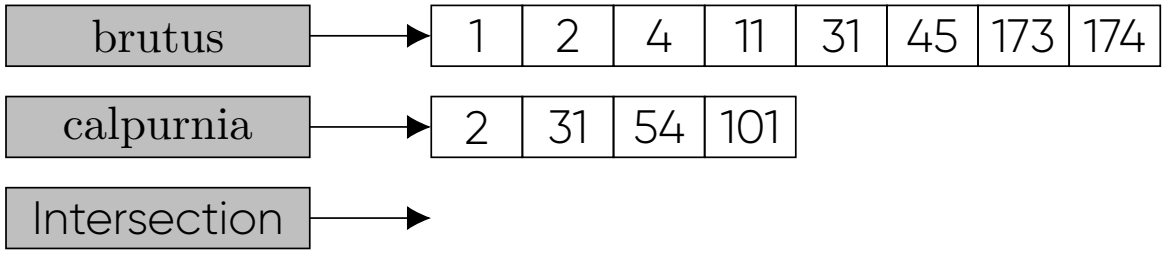
Traitement des requêtes
Conjonctions

- Requête simple conjonctive : opérateur ET
  - Ex. : $\text{brutus} \wedge \text{calpurnia}$
- Principe :
  1. Trouver brutus dans le lexique et récupérer ses postings
  2. Trouver calpurnia dans le lexique et récupérer ses postings
  3. Calculer l'intersection de ces deux listes



Traitement des requêtes
Calcul de l'intersection

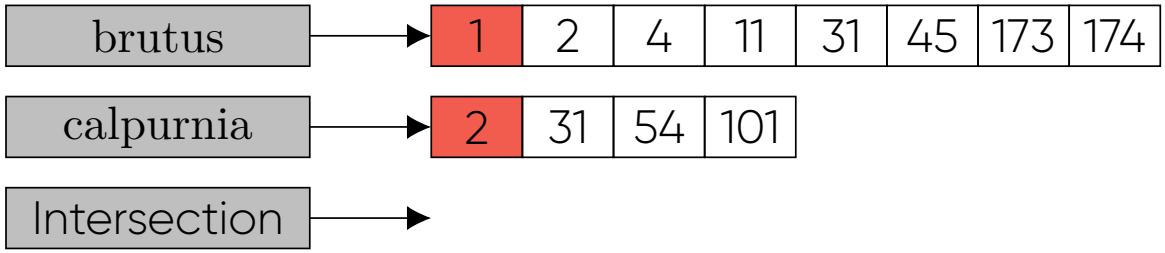
- Calcul de l'intersection :
  - On parcourt les deux listes de postings à la fois
    - Si égalité de docID : on ajoute dans le résultat et on avance les 2 listes
    - Sinon, on avance sur la liste de plus petit docID
  - Arrêt quand on arrive à la fin d'au moins une des 2 listes



- Propriétés de cette méthode :
  - Fonctionne seulement si les postings sont ordonnés
  - Complexité linéaire par rapport au nombre de postings

Traitement des requêtes
Calcul de l'intersection

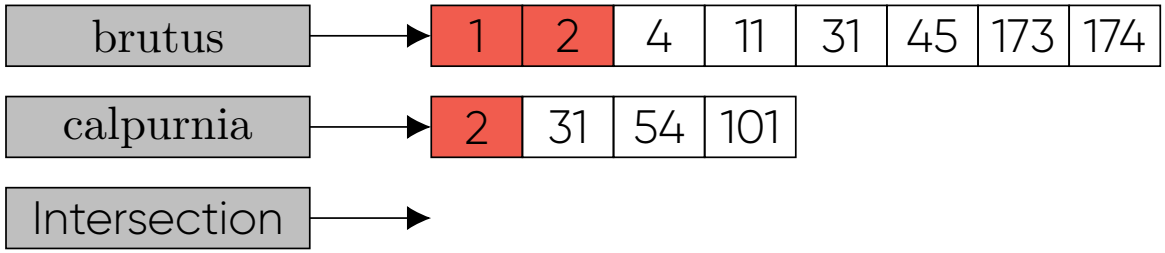
- Calcul de l'intersection :
  - On parcourt les deux listes de postings à la fois
    - Si égalité de docID : on ajoute dans le résultat et on avance les 2 listes
    - Sinon, on avance sur la liste de plus petit docID
  - Arrêt quand on arrive à la fin d'au moins une des 2 listes



- Propriétés de cette méthode :
  - Fonctionne seulement si les postings sont ordonnés
  - Complexité linéaire par rapport au nombre de postings

Traitement des requêtes
Calcul de l'intersection

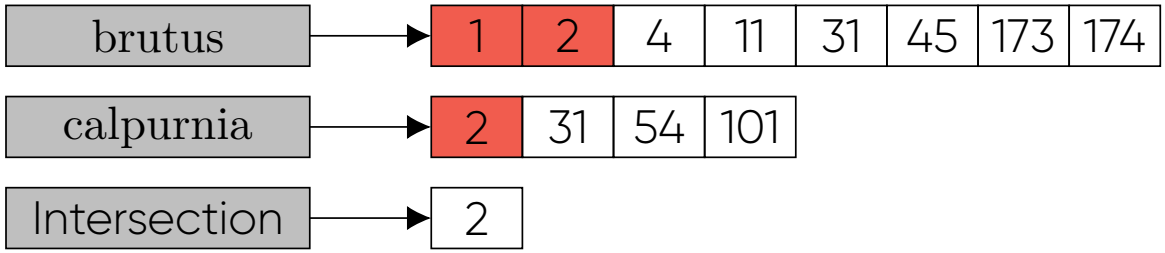
- Calcul de l'intersection :
  - On parcourt les deux listes de postings à la fois
    - Si égalité de docID : on ajoute dans le résultat et on avance les 2 listes
    - Sinon, on avance sur la liste de plus petit docID
  - Arrêt quand on arrive à la fin d'au moins une des 2 listes



- Propriétés de cette méthode :
  - Fonctionne seulement si les postings sont ordonnés
  - Complexité linéaire par rapport au nombre de postings

Traitement des requêtes
Calcul de l'intersection

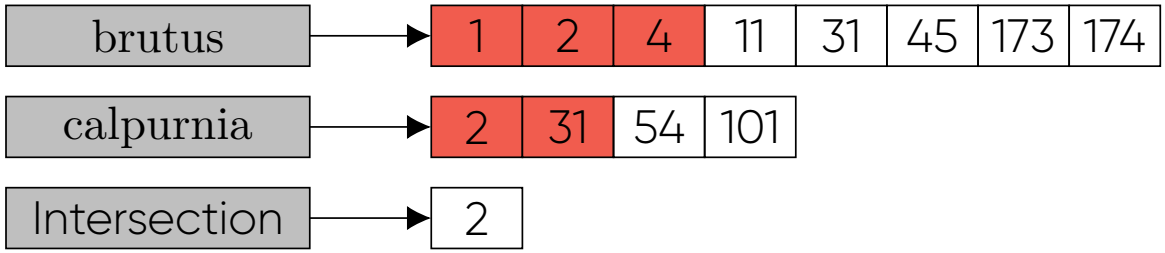
- Calcul de l'intersection :
  - On parcourt les deux listes de postings à la fois
    - Si égalité de docID : on ajoute dans le résultat et on avance les 2 listes
    - Sinon, on avance sur la liste de plus petit docID
  - Arrêt quand on arrive à la fin d'au moins une des 2 listes



- Propriétés de cette méthode :
  - Fonctionne seulement si les postings sont ordonnés
  - Complexité linéaire par rapport au nombre de postings

Traitement des requêtes
Calcul de l'intersection

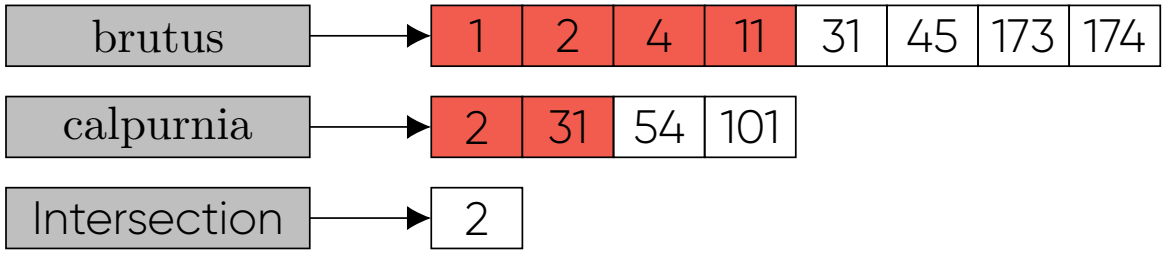
- Calcul de l'intersection :
  - On parcourt les deux listes de postings à la fois
    - Si égalité de docID : on ajoute dans le résultat et on avance les 2 listes
    - Sinon, on avance sur la liste de plus petit docID
  - Arrêt quand on arrive à la fin d'au moins une des 2 listes



- Propriétés de cette méthode :
  - Fonctionne seulement si les postings sont ordonnés
  - Complexité linéaire par rapport au nombre de postings

Traitement des requêtes
Calcul de l'intersection

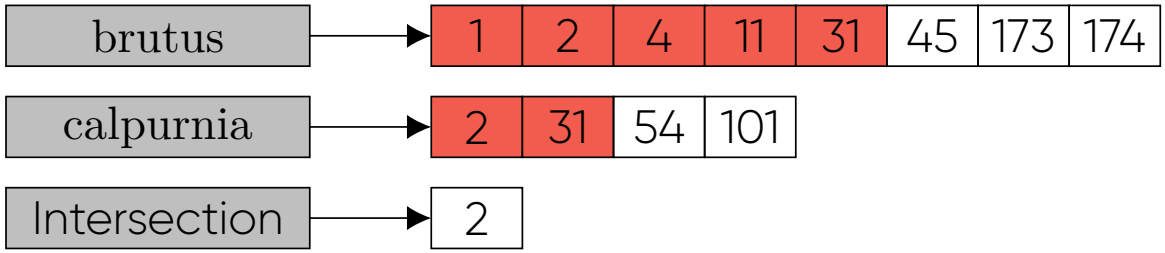
- Calcul de l'intersection :
  - On parcourt les deux listes de postings à la fois
    - Si égalité de docID : on ajoute dans le résultat et on avance les 2 listes
    - Sinon, on avance sur la liste de plus petit docID
  - Arrêt quand on arrive à la fin d'au moins une des 2 listes



- Propriétés de cette méthode :
  - Fonctionne seulement si les postings sont ordonnés
  - Complexité linéaire par rapport au nombre de postings

Traitement des requêtes
Calcul de l'intersection

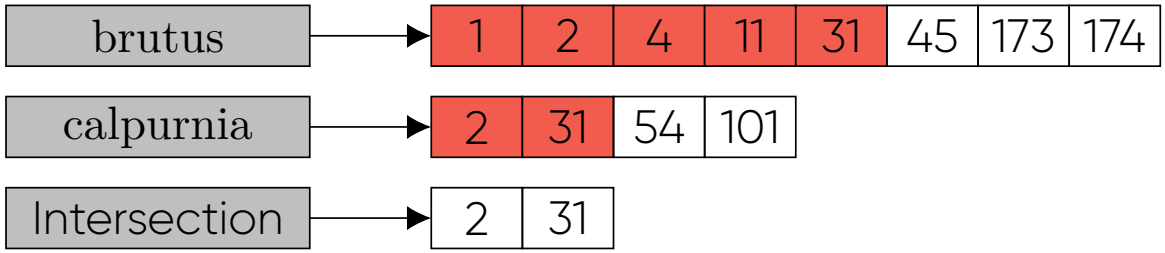
- Calcul de l'intersection :
  - On parcourt les deux listes de postings à la fois
    - Si égalité de docID : on ajoute dans le résultat et on avance les 2 listes
    - Sinon, on avance sur la liste de plus petit docID
  - Arrêt quand on arrive à la fin d'au moins une des 2 listes



- Propriétés de cette méthode :
  - Fonctionne seulement si les postings sont ordonnés
  - Complexité linéaire par rapport au nombre de postings

Traitement des requêtes
Calcul de l'intersection

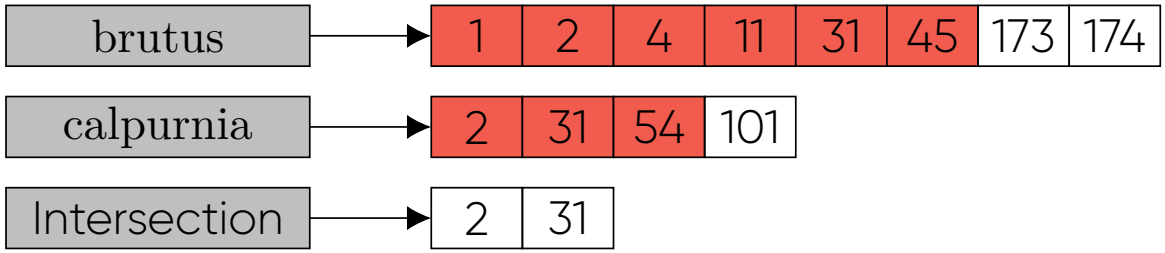
- Calcul de l'intersection :
  - On parcourt les deux listes de postings à la fois
    - Si égalité de docID : on ajoute dans le résultat et on avance les 2 listes
    - Sinon, on avance sur la liste de plus petit docID
  - Arrêt quand on arrive à la fin d'au moins une des 2 listes



- Propriétés de cette méthode :
  - Fonctionne seulement si les postings sont ordonnés
  - Complexité linéaire par rapport au nombre de postings

Traitement des requêtes
Calcul de l'intersection

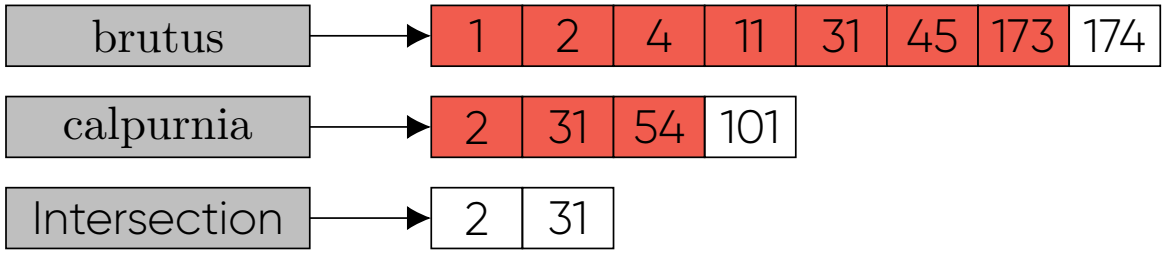
- Calcul de l'intersection :
  - On parcourt les deux listes de postings à la fois
    - Si égalité de docID : on ajoute dans le résultat et on avance les 2 listes
    - Sinon, on avance sur la liste de plus petit docID
  - Arrêt quand on arrive à la fin d'au moins une des 2 listes



- Propriétés de cette méthode :
  - Fonctionne seulement si les postings sont ordonnés
  - Complexité linéaire par rapport au nombre de postings

Traitement des requêtes
Calcul de l'intersection

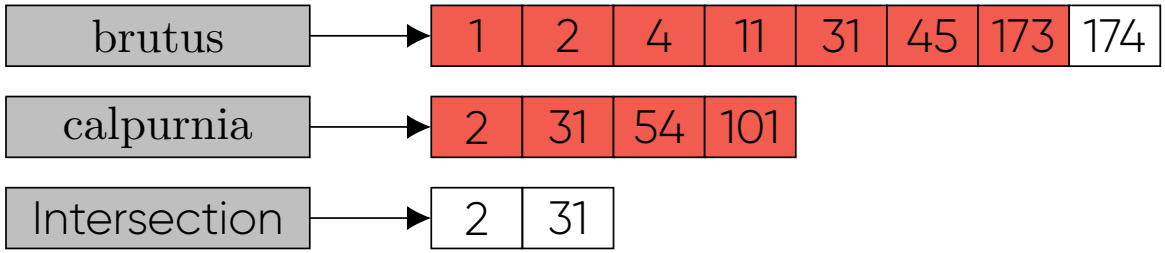
- Calcul de l'intersection :
  - On parcourt les deux listes de postings à la fois
    - Si égalité de docID : on ajoute dans le résultat et on avance les 2 listes
    - Sinon, on avance sur la liste de plus petit docID
  - Arrêt quand on arrive à la fin d'au moins une des 2 listes



- Propriétés de cette méthode :
  - Fonctionne seulement si les postings sont ordonnés
  - Complexité linéaire par rapport au nombre de postings

Traitement des requêtes
Calcul de l'intersection

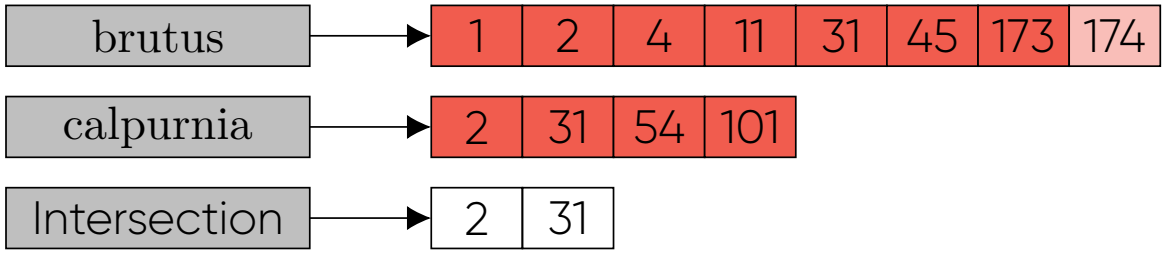
- Calcul de l'intersection :
  - On parcourt les deux listes de postings à la fois
    - Si égalité de docID : on ajoute dans le résultat et on avance les 2 listes
    - Sinon, on avance sur la liste de plus petit docID
  - Arrêt quand on arrive à la fin d'au moins une des 2 listes



- Propriétés de cette méthode :
  - Fonctionne seulement si les postings sont ordonnés
  - Complexité linéaire par rapport au nombre de postings

Traitement des requêtes
Calcul de l'intersection

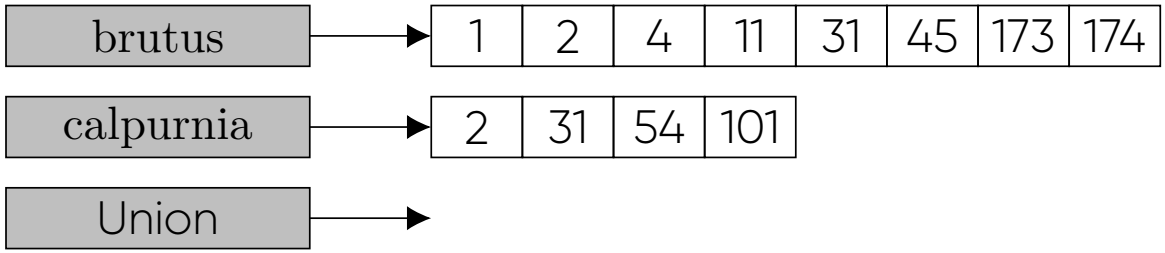
- Calcul de l'intersection :
  - On parcourt les deux listes de postings à la fois
    - Si égalité de docID : on ajoute dans le résultat et on avance les 2 listes
    - Sinon, on avance sur la liste de plus petit docID
  - Arrêt quand on arrive à la fin d'au moins une des 2 listes



- Propriétés de cette méthode :
  - Fonctionne seulement si les postings sont ordonnés
  - Complexité linéaire par rapport au nombre de postings

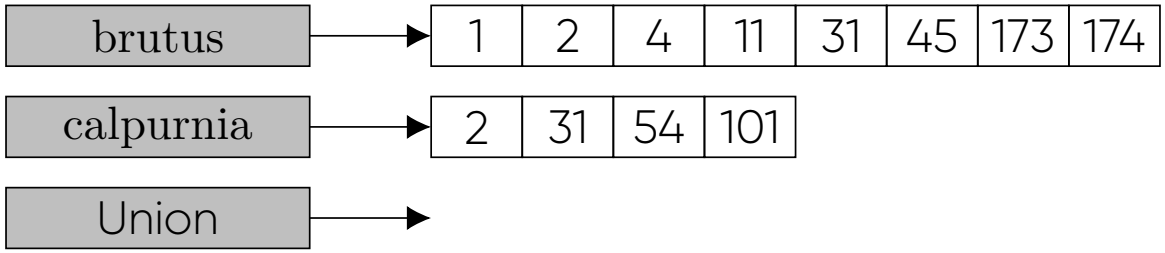
Traitement des requêtes
Disjonctions

- Requête simple disjonctive : opérateur OU
  - Ex. : $\text{brutus} \vee \text{calpurnia}$
- Principe :
  - Comme pour l'intersection, mais en prenant l'union au lieu de l'intersection



Traitement des requêtes
Calcul de l'union

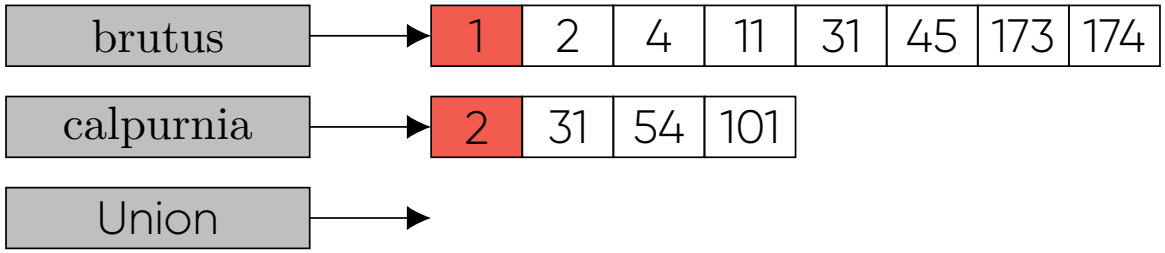
- Calcul de l'union :
  1. On parcourt les deux listes de postings à la fois
     1. Si égalité de docID : on ajoute dans le résultat et on avance les 2 listes
     2. Sinon, on ajoute le plus petit docID dans le résultat et on avance sur sa liste
  2. Quand on termine une liste, on rajoute au résultat ce qu'il reste dans l'autre



- Propriétés de cette méthode :
  - Même contrainte qu'intersection
  - Même complexité algorithmique

Traitement des requêtes
Calcul de l'union

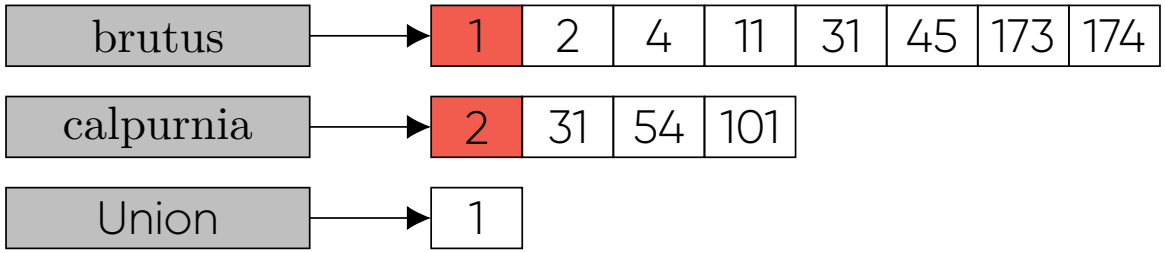
- Calcul de l'union :
  1. On parcourt les deux listes de postings à la fois
     1. Si égalité de docID : on ajoute dans le résultat et on avance les 2 listes
     2. Sinon, on ajoute le plus petit docID dans le résultat et on avance sur sa liste
  2. Quand on termine une liste, on rajoute au résultat ce qu'il reste dans l'autre



- Propriétés de cette méthode :
  - Même contrainte qu'intersection
  - Même complexité algorithmique

Traitement des requêtes
Calcul de l'union

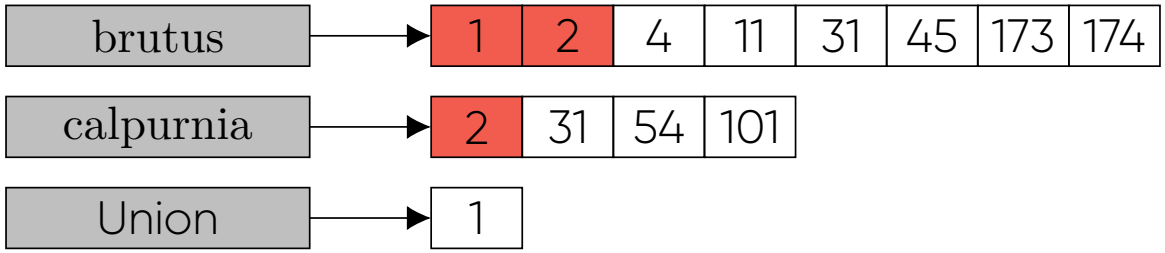
- Calcul de l'union :
  1. On parcourt les deux listes de postings à la fois
     1. Si égalité de docID : on ajoute dans le résultat et on avance les 2 listes
     2. Sinon, on ajoute le plus petit docID dans le résultat et on avance sur sa liste
  2. Quand on termine une liste, on rajoute au résultat ce qu'il reste dans l'autre



- Propriétés de cette méthode :
  - Même contrainte qu'intersection
  - Même complexité algorithmique

Traitement des requêtes
Calcul de l'union

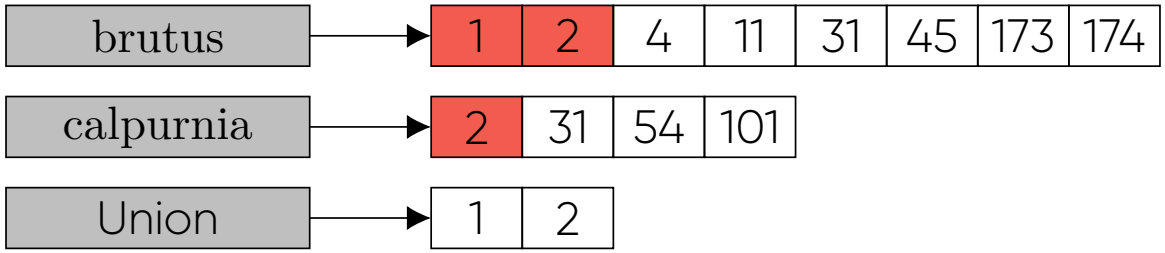
- Calcul de l'union :
  1. On parcourt les deux listes de postings à la fois
     1. Si égalité de docID : on ajoute dans le résultat et on avance les 2 listes
     2. Sinon, on ajoute le plus petit docID dans le résultat et on avance sur sa liste
  2. Quand on termine une liste, on rajoute au résultat ce qu'il reste dans l'autre



- Propriétés de cette méthode :
  - Même contrainte qu'intersection
  - Même complexité algorithmique

Traitement des requêtes
Calcul de l'union

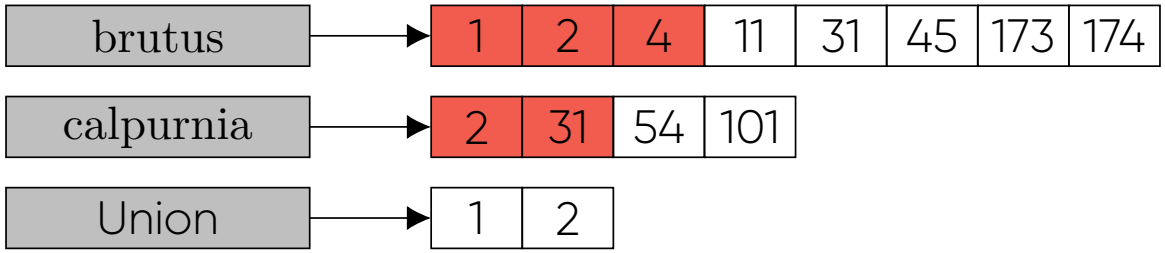
- Calcul de l'union :
  1. On parcourt les deux listes de postings à la fois
     1. Si égalité de docID : on ajoute dans le résultat et on avance les 2 listes
     2. Sinon, on ajoute le plus petit docID dans le résultat et on avance sur sa liste
  2. Quand on termine une liste, on rajoute au résultat ce qu'il reste dans l'autre



- Propriétés de cette méthode :
  - Même contrainte qu'intersection
  - Même complexité algorithmique

Traitement des requêtes
Calcul de l'union

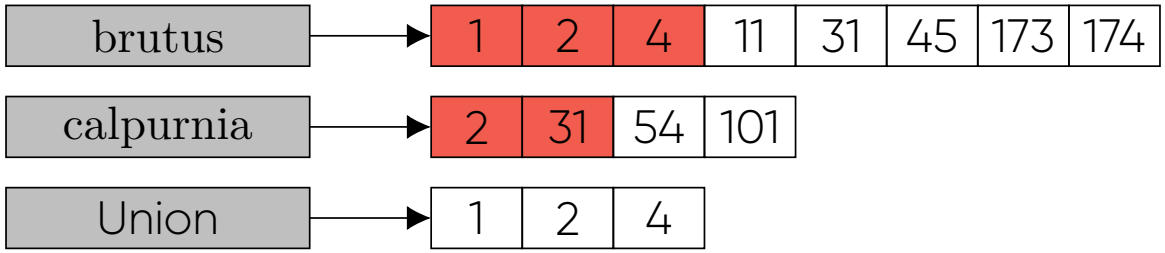
- Calcul de l'union :
  1. On parcourt les deux listes de postings à la fois
     1. Si égalité de docID : on ajoute dans le résultat et on avance les 2 listes
     2. Sinon, on ajoute le plus petit docID dans le résultat et on avance sur sa liste
  2. Quand on termine une liste, on rajoute au résultat ce qu'il reste dans l'autre



- Propriétés de cette méthode :
  - Même contrainte qu'intersection
  - Même complexité algorithmique

Traitement des requêtes
Calcul de l'union

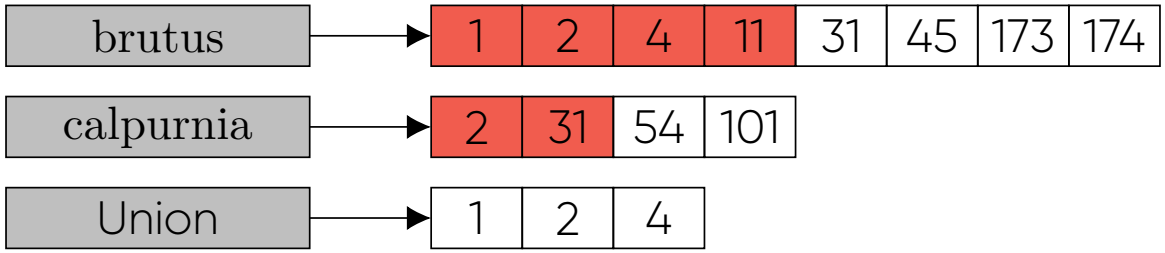
- Calcul de l'union :
  1. On parcourt les deux listes de postings à la fois
     1. Si égalité de docID : on ajoute dans le résultat et on avance les 2 listes
     2. Sinon, on ajoute le plus petit docID dans le résultat et on avance sur sa liste
  2. Quand on termine une liste, on rajoute au résultat ce qu'il reste dans l'autre



- Propriétés de cette méthode :
  - Même contrainte qu'intersection
  - Même complexité algorithmique

Traitement des requêtes
Calcul de l'union

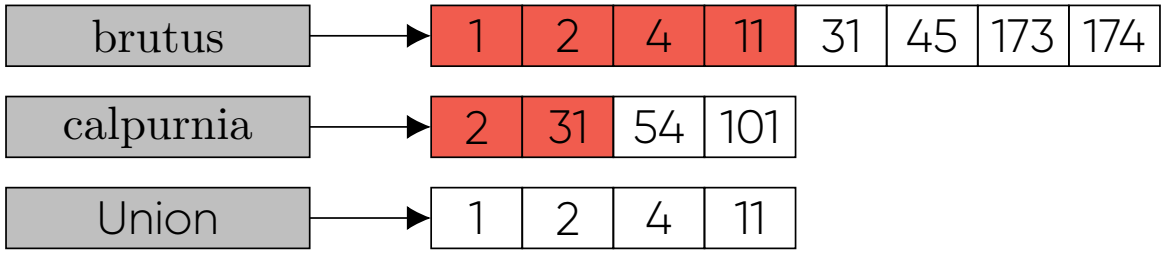
- Calcul de l'union :
  1. On parcourt les deux listes de postings à la fois
     1. Si égalité de docID : on ajoute dans le résultat et on avance les 2 listes
     2. Sinon, on ajoute le plus petit docID dans le résultat et on avance sur sa liste
  2. Quand on termine une liste, on rajoute au résultat ce qu'il reste dans l'autre



- Propriétés de cette méthode :
  - Même contrainte qu'intersection
  - Même complexité algorithmique

Traitement des requêtes
Calcul de l'union

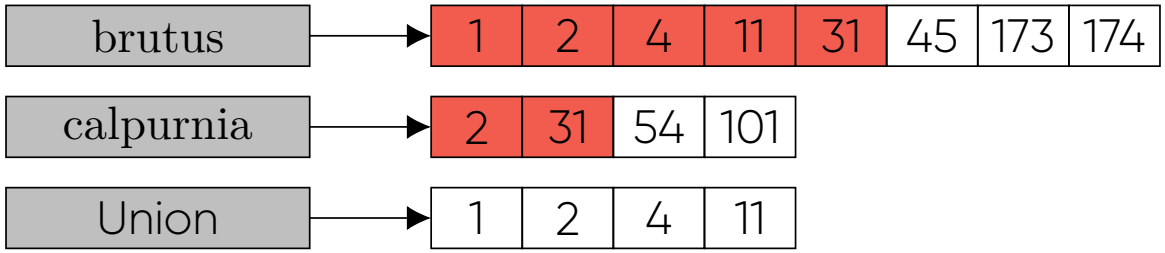
- Calcul de l'union :
  1. On parcourt les deux listes de postings à la fois
     1. Si égalité de docID : on ajoute dans le résultat et on avance les 2 listes
     2. Sinon, on ajoute le plus petit docID dans le résultat et on avance sur sa liste
  2. Quand on termine une liste, on rajoute au résultat ce qu'il reste dans l'autre



- Propriétés de cette méthode :
  - Même contrainte qu'intersection
  - Même complexité algorithmique

Traitement des requêtes
Calcul de l'union

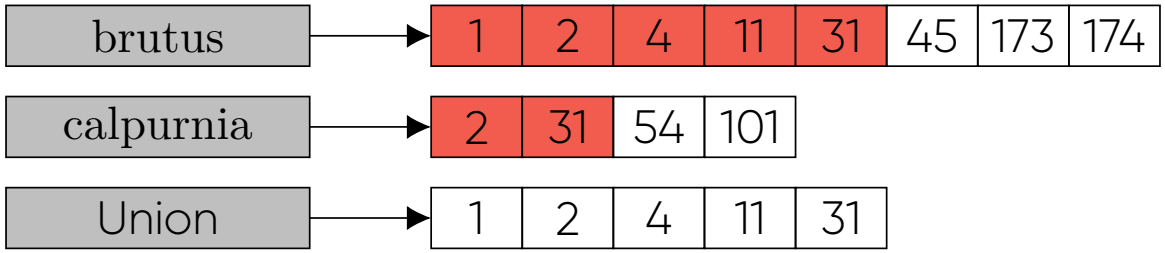
- Calcul de l'union :
  1. On parcourt les deux listes de postings à la fois
     1. Si égalité de docID : on ajoute dans le résultat et on avance les 2 listes
     2. Sinon, on ajoute le plus petit docID dans le résultat et on avance sur sa liste
  2. Quand on termine une liste, on rajoute au résultat ce qu'il reste dans l'autre



- Propriétés de cette méthode :
  - Même contrainte qu'intersection
  - Même complexité algorithmique

Traitement des requêtes
Calcul de l'union

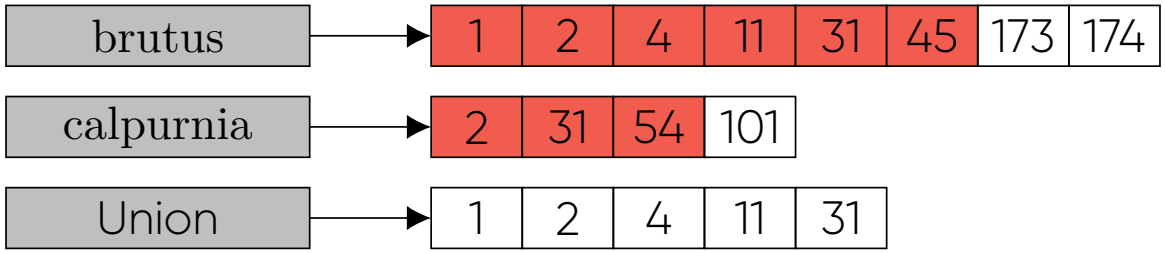
- Calcul de l'union :
  1. On parcourt les deux listes de postings à la fois
     1. Si égalité de docID : on ajoute dans le résultat et on avance les 2 listes
     2. Sinon, on ajoute le plus petit docID dans le résultat et on avance sur sa liste
  2. Quand on termine une liste, on rajoute au résultat ce qu'il reste dans l'autre



- Propriétés de cette méthode :
  - Même contrainte qu'intersection
  - Même complexité algorithmique

Traitement des requêtes
Calcul de l'union

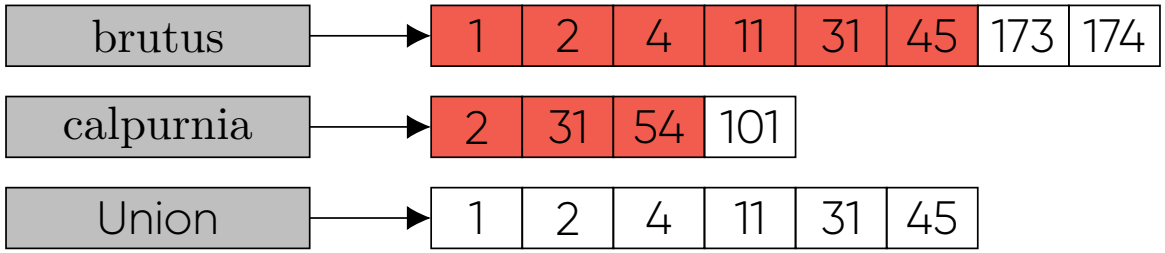
- Calcul de l'union :
  1. On parcourt les deux listes de postings à la fois
     1. Si égalité de docID : on ajoute dans le résultat et on avance les 2 listes
     2. Sinon, on ajoute le plus petit docID dans le résultat et on avance sur sa liste
  2. Quand on termine une liste, on rajoute au résultat ce qu'il reste dans l'autre



- Propriétés de cette méthode :
  - Même contrainte qu'intersection
  - Même complexité algorithmique

Traitement des requêtes
Calcul de l'union

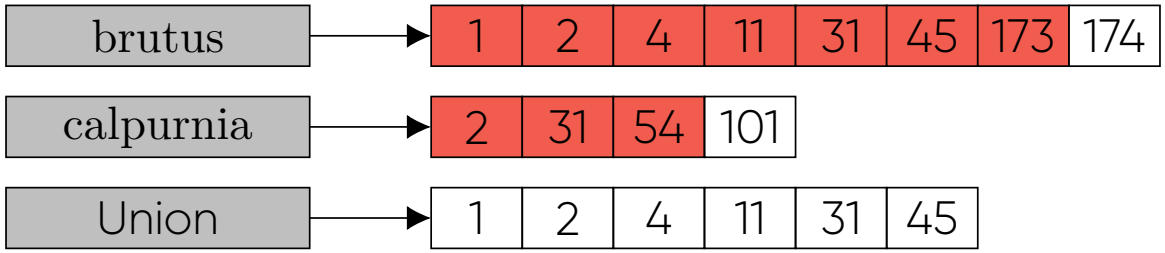
- Calcul de l'union :
  1. On parcourt les deux listes de postings à la fois
     1. Si égalité de docID : on ajoute dans le résultat et on avance les 2 listes
     2. Sinon, on ajoute le plus petit docID dans le résultat et on avance sur sa liste
  2. Quand on termine une liste, on rajoute au résultat ce qu'il reste dans l'autre



- Propriétés de cette méthode :
  - Même contrainte qu'intersection
  - Même complexité algorithmique

Traitement des requêtes
Calcul de l'union

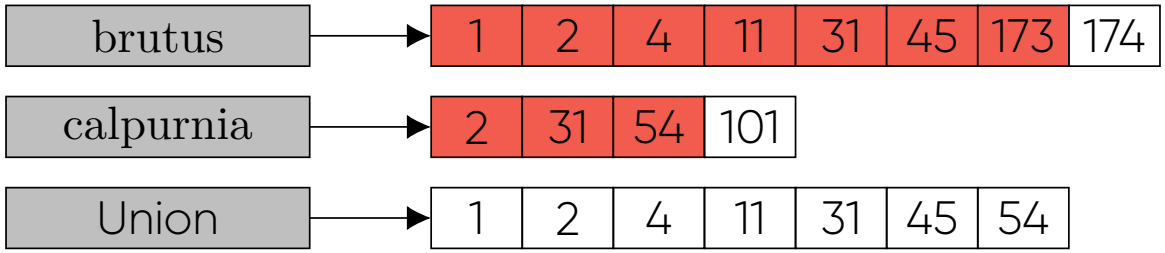
- Calcul de l'union :
  1. On parcourt les deux listes de postings à la fois
     1. Si égalité de docID : on ajoute dans le résultat et on avance les 2 listes
     2. Sinon, on ajoute le plus petit docID dans le résultat et on avance sur sa liste
  2. Quand on termine une liste, on rajoute au résultat ce qu'il reste dans l'autre



- Propriétés de cette méthode :
  - Même contrainte qu'intersection
  - Même complexité algorithmique

Traitement des requêtes
Calcul de l'union

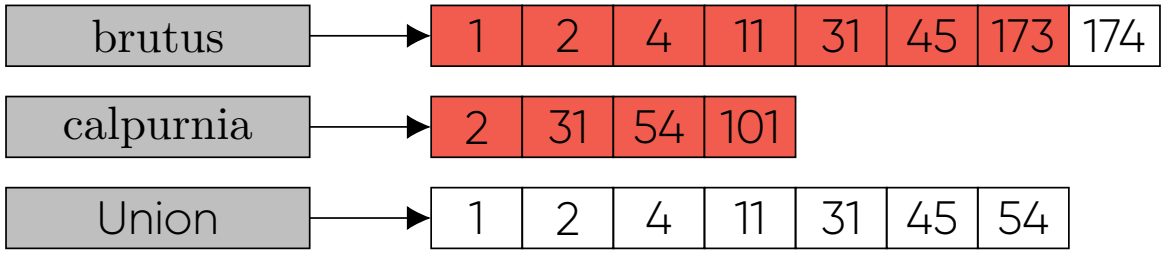
- Calcul de l'union :
  1. On parcourt les deux listes de postings à la fois
     1. Si égalité de docID : on ajoute dans le résultat et on avance les 2 listes
     2. Sinon, on ajoute le plus petit docID dans le résultat et on avance sur sa liste
  2. Quand on termine une liste, on rajoute au résultat ce qu'il reste dans l'autre



- Propriétés de cette méthode :
  - Même contrainte qu'intersection
  - Même complexité algorithmique

Traitement des requêtes
Calcul de l'union

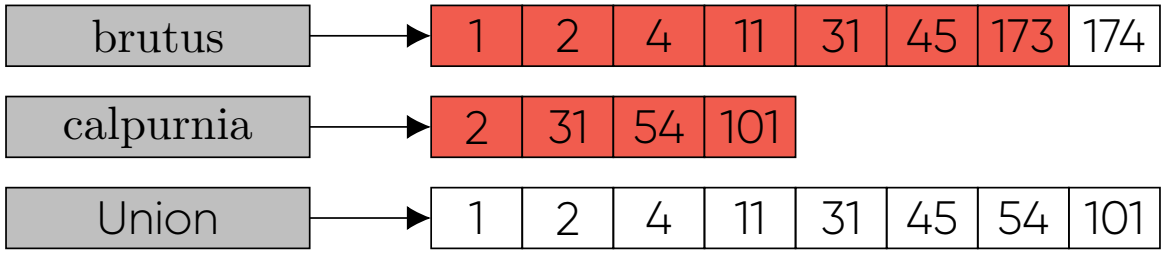
- Calcul de l'union :
  1. On parcourt les deux listes de postings à la fois
     1. Si égalité de docID : on ajoute dans le résultat et on avance les 2 listes
     2. Sinon, on ajoute le plus petit docID dans le résultat et on avance sur sa liste
  2. Quand on termine une liste, on rajoute au résultat ce qu'il reste dans l'autre



- Propriétés de cette méthode :
  - Même contrainte qu'intersection
  - Même complexité algorithmique

Traitement des requêtes
Calcul de l'union

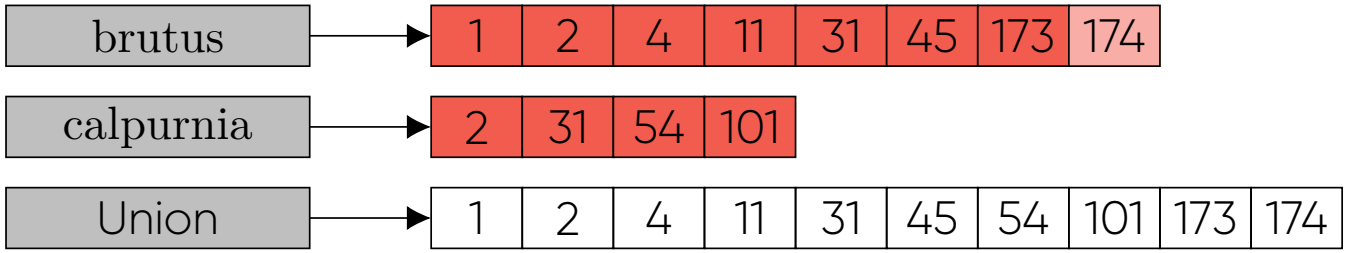
- Calcul de l'union :
  1. On parcourt les deux listes de postings à la fois
     1. Si égalité de docID : on ajoute dans le résultat et on avance les 2 listes
     2. Sinon, on ajoute le plus petit docID dans le résultat et on avance sur sa liste
  2. Quand on termine une liste, on rajoute au résultat ce qu'il reste dans l'autre



- Propriétés de cette méthode :
  - Même contrainte qu'intersection
  - Même complexité algorithmique

Traitement des requêtes
Calcul de l'union

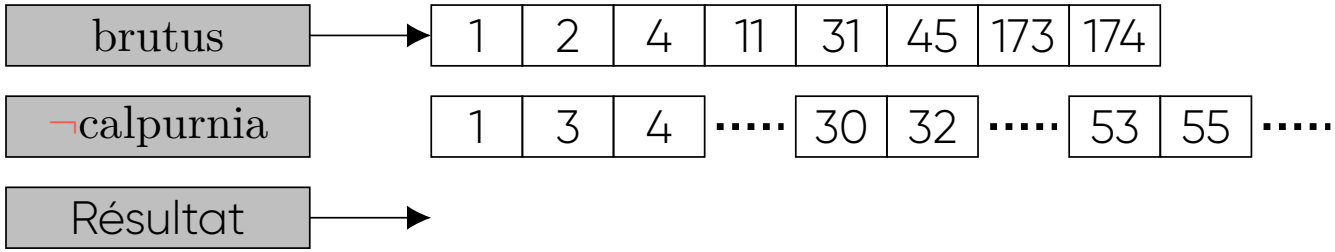
- Calcul de l'union :
  1. On parcourt les deux listes de postings à la fois
     1. Si égalité de docID : on ajoute dans le résultat et on avance les 2 listes
     2. Sinon, on ajoute le plus petit docID dans le résultat et on avance sur sa liste
  2. Quand on termine une liste, on rajoute au résultat ce qu'il reste dans l'autre



- Propriétés de cette méthode :
  - Même contrainte qu'intersection
  - Même complexité algorithmique

Traitement des requêtes
Négation

- Opérateur négation : NON
  - Ex. : $\text{brutus} \wedge \neg\text{calpurnia}$
- Approche naïve : traiter l'opérateur indépendamment
  - Utiliser le complémentaire des postings de calpurnia
  - Calculer le ET comme précédemment sur les deux listes



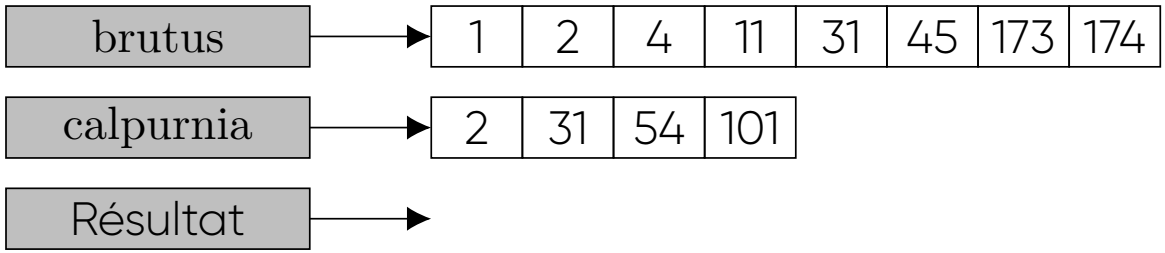
- Inconvénient :
  - Le complémentaire a une longueur élevée

  → traitement inefficace car très long en pratique
  - (Même si la complexité de l'algorithme reste la même)

Traitement des requêtes
Calcul de la négation

- Approche plus rapide : traiter ET et NON en même temps

1. On parcourt les deux listes de postings à la fois
   1. Si égalité de docID : on avance dans les 2 listes
   2. Sinon : on avance sur la liste du docID le plus petit, et si c'est la 1ère liste, on rajoute ce docID au résultat.
2. Si on termine la 2ème liste, on rajoute au résultat ce qu'il reste dans la 1ère

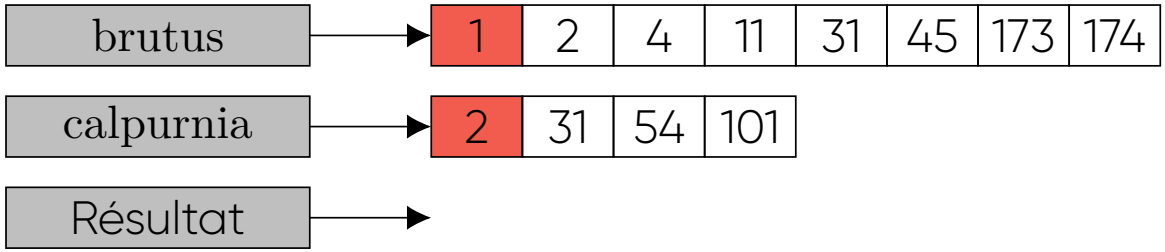


- On applique une approche similaire pour NON OU

Traitement des requêtes
Calcul de la négation

- Approche plus rapide : traiter ET et NON en même temps

1. On parcourt les deux listes de postings à la fois
   1. Si égalité de docID : on avance dans les 2 listes
   2. Sinon : on avance sur la liste du docID le plus petit, et si c'est la 1ère liste, on rajoute ce docID au résultat.
2. Si on termine la 2ème liste, on rajoute au résultat ce qu'il reste dans la 1ère

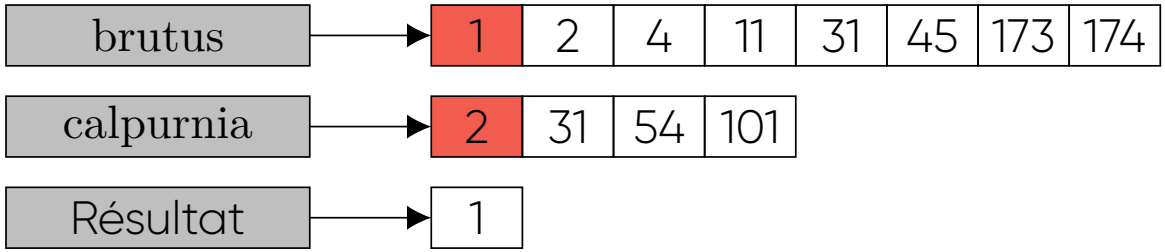


- On applique une approche similaire pour NON OU

Traitement des requêtes
Calcul de la négation

- Approche plus rapide : traiter ET et NON en même temps

1. On parcourt les deux listes de postings à la fois
   1. Si égalité de docID : on avance dans les 2 listes
   2. Sinon : on avance sur la liste du docID le plus petit, et si c'est la 1ère liste, on rajoute ce docID au résultat.
2. Si on termine la 2ème liste, on rajoute au résultat ce qu'il reste dans la 1ère

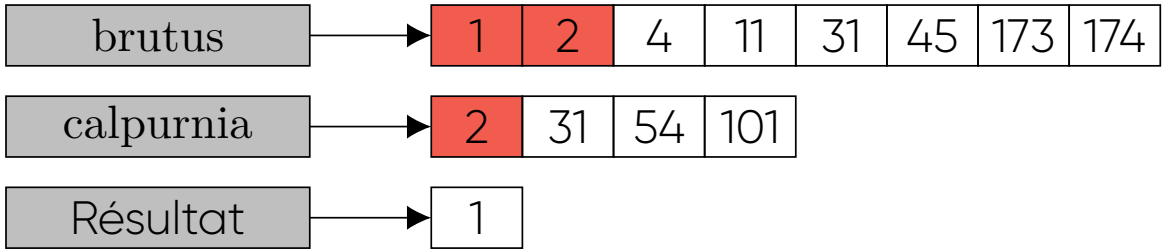


- On applique une approche similaire pour NON OU

Traitement des requêtes
Calcul de la négation

- Approche plus rapide : traiter ET et NON en même temps

1. On parcourt les deux listes de postings à la fois
   1. Si égalité de docID : on avance dans les 2 listes
   2. Sinon : on avance sur la liste du docID le plus petit, et si c'est la 1ère liste, on rajoute ce docID au résultat.
2. Si on termine la 2ème liste, on rajoute au résultat ce qu'il reste dans la 1ère

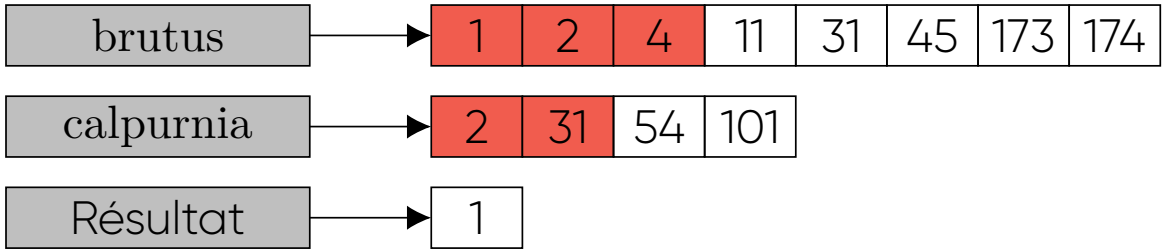


- On applique une approche similaire pour NON OU

Traitement des requêtes
Calcul de la négation

- Approche plus rapide : traiter ET et NON en même temps

1. On parcourt les deux listes de postings à la fois
   1. Si égalité de docID : on avance dans les 2 listes
   2. Sinon : on avance sur la liste du docID le plus petit, et si c'est la 1ère liste, on rajoute ce docID au résultat.
2. Si on termine la 2ème liste, on rajoute au résultat ce qu'il reste dans la 1ère

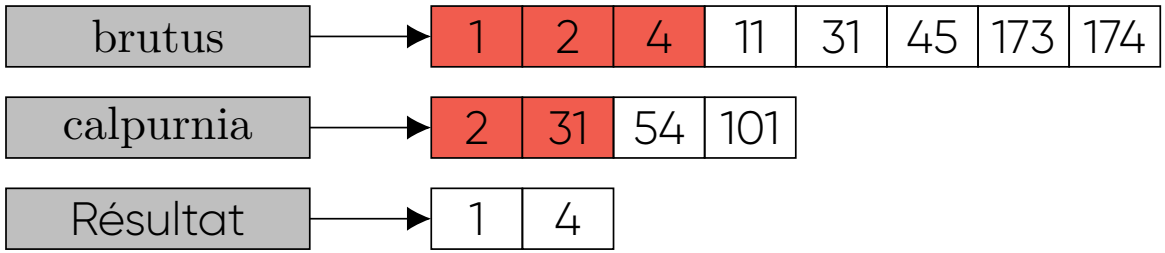


- On applique une approche similaire pour NON OU

Traitement des requêtes
Calcul de la négation

- Approche plus rapide : traiter ET et NON en même temps

1. On parcourt les deux listes de postings à la fois
   1. Si égalité de docID : on avance dans les 2 listes
   2. Sinon : on avance sur la liste du docID le plus petit, et si c'est la 1ère liste, on rajoute ce docID au résultat.
2. Si on termine la 2ème liste, on rajoute au résultat ce qu'il reste dans la 1ère

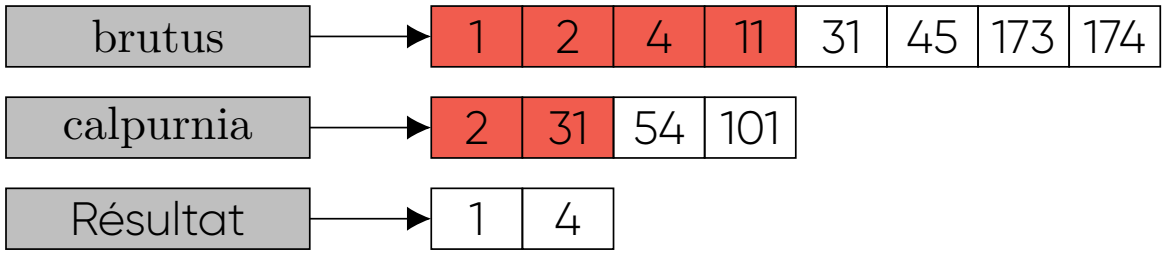


- On applique une approche similaire pour NON OU

Traitement des requêtes
Calcul de la négation

- Approche plus rapide : traiter ET et NON en même temps

1. On parcourt les deux listes de postings à la fois
   1. Si égalité de docID : on avance dans les 2 listes
   2. Sinon : on avance sur la liste du docID le plus petit, et si c'est la 1ère liste, on rajoute ce docID au résultat.
2. Si on termine la 2ème liste, on rajoute au résultat ce qu'il reste dans la 1ère

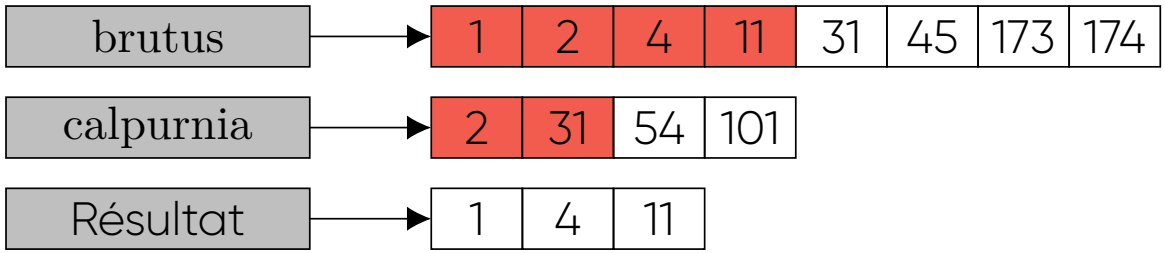


- On applique une approche similaire pour NON OU

Traitement des requêtes
Calcul de la négation

- Approche plus rapide : traiter ET et NON en même temps

1. On parcourt les deux listes de postings à la fois
   1. Si égalité de docID : on avance dans les 2 listes
   2. Sinon : on avance sur la liste du docID le plus petit, et si c'est la 1ère liste, on rajoute ce docID au résultat.
2. Si on termine la 2ème liste, on rajoute au résultat ce qu'il reste dans la 1ère

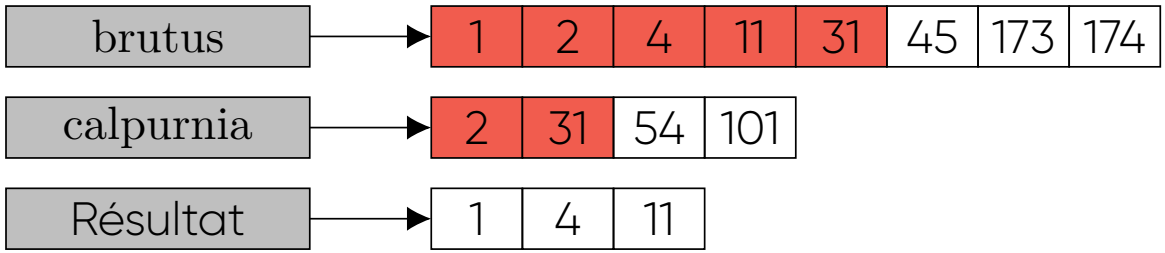


- On applique une approche similaire pour NON OU

Traitement des requêtes
Calcul de la négation

- Approche plus rapide : traiter ET et NON en même temps

1. On parcourt les deux listes de postings à la fois
   1. Si égalité de docID : on avance dans les 2 listes
   2. Sinon : on avance sur la liste du docID le plus petit, et si c'est la 1ère liste, on rajoute ce docID au résultat.
2. Si on termine la 2ème liste, on rajoute au résultat ce qu'il reste dans la 1ère

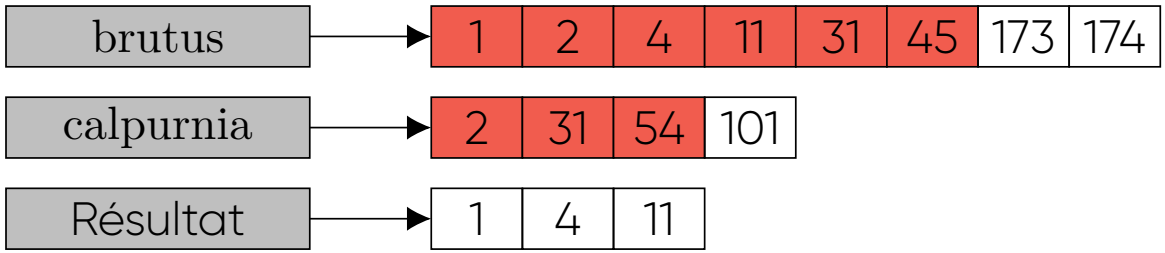


- On applique une approche similaire pour NON OU

Traitement des requêtes
Calcul de la négation

- Approche plus rapide : traiter ET et NON en même temps

1. On parcourt les deux listes de postings à la fois
   1. Si égalité de docID : on avance dans les 2 listes
   2. Sinon : on avance sur la liste du docID le plus petit, et si c'est la 1ère liste, on rajoute ce docID au résultat.
2. Si on termine la 2ème liste, on rajoute au résultat ce qu'il reste dans la 1ère

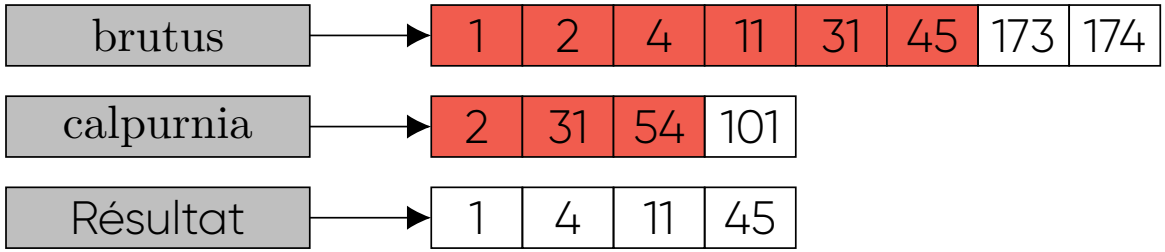


- On applique une approche similaire pour NON OU

Traitement des requêtes
Calcul de la négation

- Approche plus rapide : traiter ET et NON en même temps

1. On parcourt les deux listes de postings à la fois
   1. Si égalité de docID : on avance dans les 2 listes
   2. Sinon : on avance sur la liste du docID le plus petit, et si c'est la 1ère liste, on rajoute ce docID au résultat.
2. Si on termine la 2ème liste, on rajoute au résultat ce qu'il reste dans la 1ère

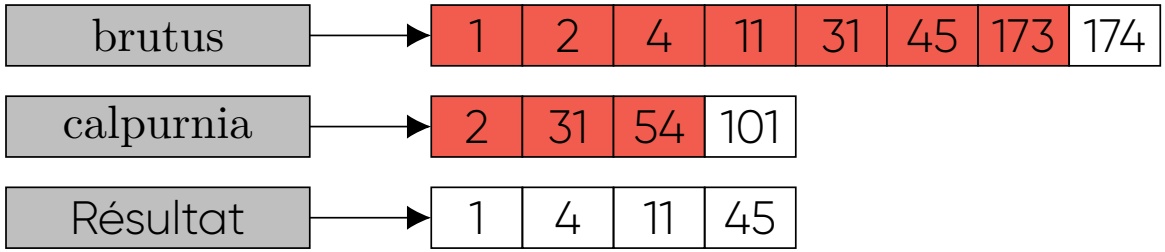


- On applique une approche similaire pour NON OU

Traitement des requêtes
Calcul de la négation

- Approche plus rapide : traiter ET et NON en même temps

1. On parcourt les deux listes de postings à la fois
   1. Si égalité de docID : on avance dans les 2 listes
   2. Sinon : on avance sur la liste du docID le plus petit, et si c'est la 1ère liste, on rajoute ce docID au résultat.
2. Si on termine la 2ème liste, on rajoute au résultat ce qu'il reste dans la 1ère

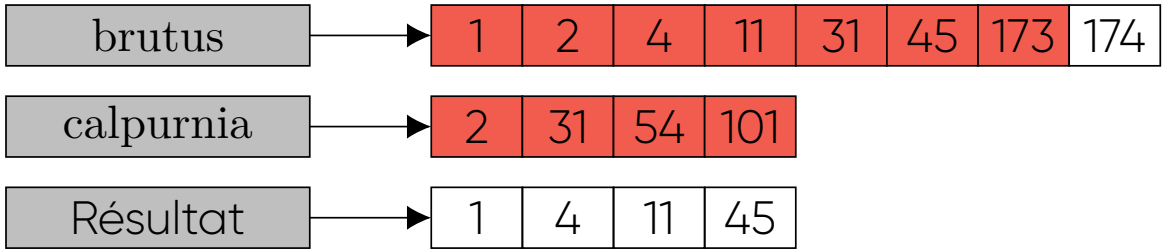


- On applique une approche similaire pour NON OU

Traitement des requêtes
Calcul de la négation

- Approche plus rapide : traiter ET et NON en même temps

1. On parcourt les deux listes de postings à la fois
   1. Si égalité de docID : on avance dans les 2 listes
   2. Sinon : on avance sur la liste du docID le plus petit, et si c'est la 1ère liste, on rajoute ce docID au résultat.
2. Si on termine la 2ème liste, on rajoute au résultat ce qu'il reste dans la 1ère

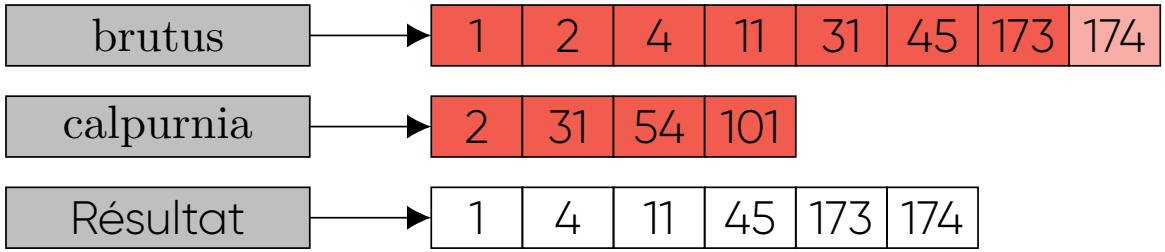


- On applique une approche similaire pour NON OU

Traitement des requêtes
Calcul de la négation

- Approche plus rapide : traiter ET et NON en même temps

1. On parcourt les deux listes de postings à la fois
   1. Si égalité de docID : on avance dans les 2 listes
   2. Sinon : on avance sur la liste du docID le plus petit, et si c'est la 1ère liste, on rajoute ce docID au résultat.
2. Si on termine la 2ème liste, on rajoute au résultat ce qu'il reste dans la 1ère



- On applique une approche similaire pour NON OU

Traitement des requêtes
Optimisation de requêtes

- Méthode permettant de :
  - Décider comment évaluer une requête contenant de multiples opérateurs
  - De façon à minimiser les calculs effectués
- Approche standard :
  - Traiter en priorité les termes de plus petite fréquence
- Ex. : $\text{brutus} \wedge \text{caesar} \wedge \text{calpurnia}$
  - Fréquences : 8, 8 et 4
  - $\rightarrow$ On calcule $(\text{calpurnia} \wedge \text{brutus}) \wedge \text{caesar}$
- Requêtes quelconques :
  - Nécessité de considérer la fréquence de résultats partiels
  - Ex. : $(\text{madding} \vee \text{crowd}) \wedge (\text{ignoble} \vee \text{strife}) \wedge (\text{killed} \vee \text{slain})$

Limites du modèle de recherche booléen

- Limites
  - Opérateurs booléens
  - Résultats pas ordonnés
  - Uniquement comparaisons exactes
- Extensions du modèle booléen
  - Opérateur de proximité
    - Mots devant être au plus à une certaine distance (exprimée en mot)
    - Ou dans la même unité structurelle (phrase, paragraphe...)
  - Utilisation de jokers
    - Ex. : plate-forme pour plateforme, plate-forme et plate forme
  - Fréquence tf des termes dans un document

  → Nécessité de modifier l'index
- Requêtes en texte libre

Section 4

Conclusion

AVIGNON
UNIVERSITÉ

Concepts abordés dans cette partie

- Grepping
- Matrice d'incidence
- Sac-de-mots simplifié
- Fichier inversé
- Modèle booléen
- Fréquence de document
- Posting
- docID
- Ordre lexicographique
- Tokénisation
- Normalisation
- Lexique

Lectures recommandées

[MRS08] *Introduction to Information Retrieval*, chapitre 1.

[BCC10] *Information Retrieval : Implementing and Evaluating Search Engines*, chapitre 1.

[BR11] *Modern Information Retrieval : The Concepts and Technology behind Search*, chapitre 1.

[AG13] *Recherche d'information - Applications, modèles et algorithmes*, chapitres 1 & 2.

[CMS15] *Search Engines : Information Retrieval in Practice*, chapitre 4.

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

[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/.