Terms and Postings

adapted from Schütze, Center for Information and Language Processing,
University of Munich

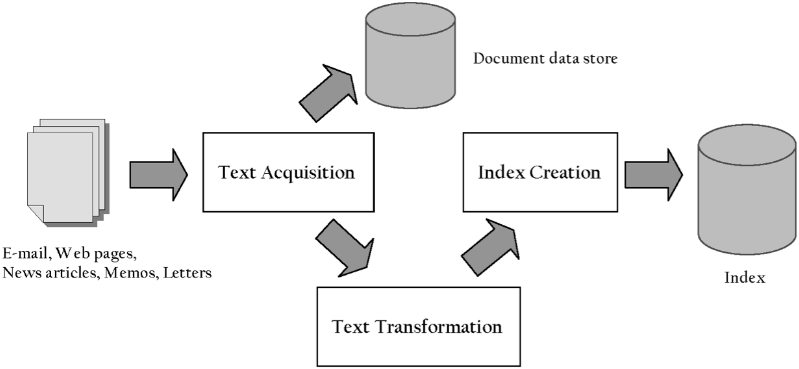
September 20, 2023

Reading material: Chapter 2 of IIR

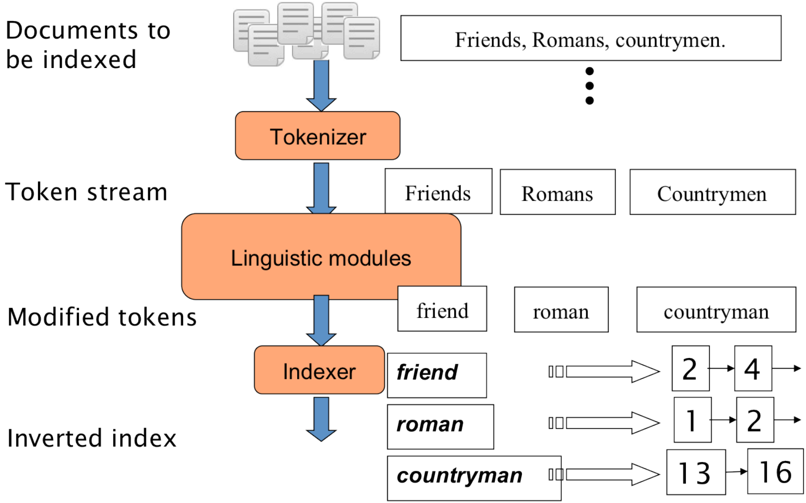
Overview

1. Documents
2. Terms
    - General + Non-English
    - Case folding
    - stop words
    - Stemming
    - lemmatization
3. Skip pointers
4. Phrase queries

Index construction



Index construction



Outline

1. Documents
2. Terms
   - General + Non-English
   - Case folding
   - stop words
   - Stemming
   - lemmatization
3. Skip pointers
4. Phrase queries

Documents

- Last lecture: Simple Boolean retrieval system
- Our assumptions were:
  - We know what a document is.
  - We can "machine-read" each document.
- This can be complex in reality.

Document format

- What format is it in?
  - pdf/word/excel/html?
- What language is it in?
- What character set is in use?
  - (CP1252, UTF-8, …)

what is a document

- We return from our query “documents”. but there are often interesting questions of grain size:
    - What is a unit document?
    - A file?
    - a chapter of a book? a section? a page?
    - An email? (Perhaps one of many in a single mbox file)
    - What about an email with 5 attachments?
    - A group of files (e.g., PPT or LaTeX split over HTML pages)

Format/Language: Complications

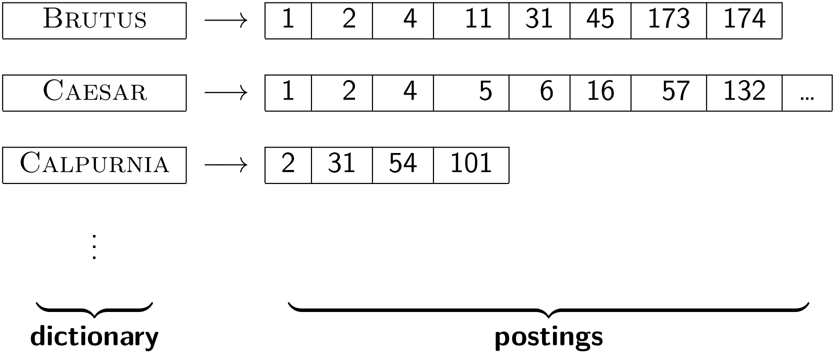
- A single index usually contains terms of several languages.
- Sometimes a document or its components contain multiple languages/formats.
  - French email with Spanish pdf attachment
- Also: XML

Outline

1. Documents
2. Terms
   - General + Non-English
   - Case folding
   - stop words
   - Stemming
   - lemmatization
3. Skip pointers
4. Phrase queries

what is a term?

For each term *t*, we store a list of all documents that contain *t*.



BRUTUS ⟶ 1 | 2 | 4 | 11 | 31 | 45 | 173 | 174

CAESAR ⟶ 1 | 2 | 4 | 5 | 6 | 16 | 57 | 132 | ...

CALPURNIA ⟶ 2 | 31 | 54 | 101

⋮

dictionary

postings

but what is a term?

Outline

1. Documents
2. Terms
   - General + Non-English
   - Case folding
   - stop words
   - Stemming
   - lemmatization
3. Skip pointers
4. Phrase queries

Definitions of token, type and term

- Token – An instance of a word or term occurring in a document.
- Type – The same as a term in most cases: an equivalence class of tokens.
- Term – A “normalized” word (case, morphology, spelling etc); an equivalence class of words.
- How many tokens? How many types?
  - *In June, the dog likes to chase the cat in the barn.*
  - *Mr. O’Neill thinks that the boys’ stories about Chile’s capital aren’t amusing.*

Examples for tokenization

- *Mr. O'Neill thinks that the boys' stories about Chile's capital aren't amusing.*
- O'Neill
  - → neill?
  - →oneill?
  - →o'neill?
  - → o AND neill ?
  - → o' AND neill?
- aren't
  - → aren't
  - → arent
  - → are n't
  - → are AND not

Approaches to dealing with token variants

- The problem: how to match *U.S.A.* with *USA*, or *Today* with *today*?
- Solution 1: "normalize" words in indexed text as well as query terms into the same form.
  - *U.S.A.* → *USA*
  - implicitly define equivalence classes of terms (i.e, U.S.A and USA).
- Solution 2: do expansion. e.g., U.S.A → U.S.A. USA.
  - Query expansion: query *U.S.A* → *U.S.A OR USA*
  - Index expansion: both *U.S.A* and *USA* are indexed terms
  - More powerful, but less efficient.

Why expansion approach can be more powerful

- Asymmetric expansion
  - window → window, windows
  - windows → Windows, windows
  - Windows (no expansion)
- better than putting *window*, *Window*, *windows*, and *Windows* in the same equivalence class

Tokenization: Recall construction of inverted index

- Input:
  Friends, Romans, countrymen. | So let it be with Caesar | ...
- Output:
  friend | roman | countryman | so | ...
- Each token is a candidate for a postings entry.
- What are valid tokens to emit?

Tokenization problems: One word or two? (or several)

- Hewlett-Packard
- State-of-the-art
- co-education
- the hold-him-back-and-drag-him-away maneuver
- data base
- San Francisco
- Los Angeles-based company
- cheap San Francisco-Los Angeles fares
- York University vs. New York University

Numbers

- 3/20/91
- 20/3/91
- Mar 20, 1991
- B-52
- 100.2.86.144
- (800) 234-2333
- 800.234.2333
- Older IR systems may not index numbers ...
- ...but generally it's a useful feature.

Ambiguous segmentation in Chinese

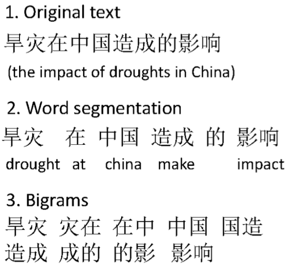
no spaces between words

莎拉波娃现在居住在美国东南部的佛罗里达。今年４月９日，莎拉波娃在美国第一大城市纽约度过了１８岁生日。生日派对上，莎拉波娃露出了甜美的微笑。

ambiguous segmentation

和尚 The two characters can be treated as one word meaning ‘monk’ or as a sequence of two words meaning ‘and’ and ‘still’.

Chinese and japanese



1. Original text

旱灾在中国造成的影响

(the impact of droughts in China)

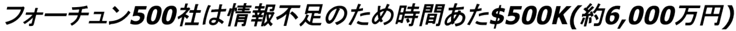
2. Word segmentation

旱灾	在	中国	造成	的	影响
drought	at	china	make		impact

3. Bigrams

旱灾 灾在 在中 中国 国造
造成 成的 的影 影响

- Chinese and Japanese have no spaces between words:
  - 结婚的和尚未结婚的
    结婚 (married) 的和 (and) 尚未 (not yet) 结婚 (married) 的
    结婚 (married) 的和尚 (monk) 未 (not) 结婚 (married) 的
  - Not always guaranteed a unique tokenization
- Further complicated in Japanese, with multiple alphabets intermingled



フォーチュン500社は情報不足のため時間あた$500K(約6,000万円)

Other cases of “no whitespace”

- Compounds in Dutch, German, Swedish
- Computerlinguistik → Computer + Linguistik
- Lebensversicherungsgesellschaftsangestellter
- → leben + versicherung + gesellschaft + angestellter
- Inuit: tusaatsiarunnanngittualuujunga (I can’t hear very well.)
- Many other languages with segmentation difficulties: Finnish, Urdu, ...

Japanese

ノーベル平和賞を受賞したワンガリ・マータイさんが名誉会長を務めるＭＯＴＴＡＩＮＡＩキャンペーンの一環として、毎日新聞社とマガジンハウスは「私 の、もったいない」を募集します。皆様が日ごろ「もったいない」と感じて実践していることや、それにまつわるエピソードを８００字以内の文章にまとめ、簡 単な写真、イラスト、図などを添えて１０月２０日までにお送りください。大賞受賞者には、５０万円相当の旅行券とエコ製品２点の副賞が贈られます。

Japanese

ノーベル平和賞を受賞したワンガリ・マータイさんが名誉会長を務めるＭＯＴＴＡＩＮＡＩキャンペーンの一環として、毎日新聞社とマガジンハウスは「私 の、もったいない」を募集します。皆様が日ごろ「もったいない」と感じて実践していることや、それにまつわるエピソードを８００字以内の文章にまとめ、簡 単な写真、イラスト、図などを添えて１０月２０日までにお送りください。大賞受賞者には、５０万円相当の旅行券とエコ製品２点の副賞が贈られます。

4 different “alphabets”: Chinese characters, hiragana syllabary for inflectional endings and function words, katakana syllabary for transcription of foreign words and other uses, and latin. No spaces (as in Chinese).

Japanese

ノーベル平和賞を受賞したワンガリ・マータイさんが名誉会長を務めるＭＯＴＴＡＩＮＡＩキャンペーンの一環として、毎日新聞社とマガジンハウスは「私の、もったいない」を募集します。皆様が日ごろ「もったいない」と感じて実践していることや、それにまつわるエピソードを８００字以内の文章にまとめ、簡単な写真、イラスト、図などを添えて１０月２０日までにお送りください。大賞受賞者には、５０万円相当の旅行券とエコ製品２点の副賞が贈られます。

4 different “alphabets”: Chinese characters, hiragana syllabary for inflectional endings and function words, katakana syllabary for transcription of foreign words and other uses, and latin. No spaces (as in Chinese).

End user can express query entirely in hiragana!

Arabic script: Bidirectionality

استقلت الجزائر في سنة 1962 بعد 132 عاما من الاحتلال الفرنسي.

← → ← → ← START

‘Algeria achieved its independence in 1962 after 132 years of French occupation.’

Bidirectionality is not a problem if text is coded in Unicode.

Accents and diacritics

- Accents: résumé vs. resume (simple omission of accent)
- Umlauts: Universität vs. Universitaet (substitution with special letter sequence “ae”)
- Most important criterion: How are users likely to write their queries for these words?
- Even in languages that standardly have accents, users often do not type them.

Outline

1. Documents
2. Terms
   - General + Non-English
   - Case folding
   - stop words
   - Stemming
   - lemmatization
3. Skip pointers
4. Phrase queries

Case folding

- Reduce all letters to lower case
- Even though case can be semantically meaningful
  - capitalized words in mid-sentence
  - MIT vs. mit
  - Fed vs. fed
  - ...
- It's often best to lowercase everything since users will use lowercase regardless of correct capitalization.

Outline

1. Documents
2. Terms
   - General + Non-English
   - Case folding
   - stop words
   - Stemming
   - lemmatization
3. Skip pointers
4. Phrase queries

Stop words

- stop words = extremely common words which would appear to be of little value in helping select documents matching a user need
- There are a lot of them: 30 % of postings for top 30 words
- Stopwords are typically function words:
    - determiners (a, the), prepositions (on, above), conjunctions (and, but)
- May also be corpus-specific: “plot”in the IMDB corpus
- Assumption: stopwords are unimportant because they are frequent in every document
- But you need stop words for phrase queries, e.g. “King of Denmark”

practice of removing stop words: older systems

- The earliest systems used stopword lists of 200-300 terms
- To improve efficiency and effectiveness
- Very frequent terms were problematic for early retrieval models (e.g, OR operations in ranked boolean)

practice of removing stop words: modern search engines

Most web search engines index stop words.

- possibly ignore them at query-time if they seem unimportant
- Good compression techniques (IIR 5) means the space for including stop words in a system is very small
- Good query optimization techniques (IIR 7) mean you pay little at query time for including stop words.
- You need them for:
  - Phrase queries: “King of Denmark”
  - Various song titles, etc.: “Let it be”, “To be or not to be”
  - “Relational”queries: “flights to London”

stopwords in Lucene

- The default stop words set in StandardAnalyzer and EnglishAnalyzer
- defined in StopAnalyzer.ENGLISH_STOP_WORDS_SET

"a", "an", "and", "are", "as", "at", "be", "but", "by", "for", "if", "in", "into", "is", "it", "no", "not", "of", "on", "or", "such", "that", "the", "their", "then", "there", "these", "they", "this", "to", "was", "will", "with"

Lemur Stopword List

first 60 (sorted alphabetically)

```
a all amongst anywhere become besides about almost an
apart becomes between above alone and are becoming
beyond according along another around been both
across already any as before but after also anybody
at beforehand by afterwards although anyhow av behind
can again always anyone be being can against am
anything became below cannot albeit among anyway
because beside canst
```

Outline

1. Documents
2. Terms
   - General + Non-English
   - Case folding
   - stop words
   - Stemming
   - lemmatization
3. Skip pointers
4. Phrase queries

Morphological Analysis in information retrieval

- Basic question: words occur in different forms.
- Do we want to treat different forms as different index terms?
- Conflation: treating different (inflectional and derivational) variants as the same index term

Morphology

- Inflectional morphology: changes to a word that encode its grammatical usage (e.g., tense, number, person).
  - say vs. said, cat vs. cats, see vs. sees
- Derivational morphology: changes to a word to make a new word with related meaning
  - organize, organization, organizational
- Compounding: combining words to form new ones
  - shipwreck, outbound, beefsteak
  - more common in other languages (e.g., german)

Stemming

- Definition of stemming: Crude heuristic process that chops off the ends of words
- Language dependent
- Often inflectional or derivational
  - Inflectional: e.g., *organizes* → *organize*
  - Derivational: *automate, automatic, automation* → *automat*

Porter algorithm

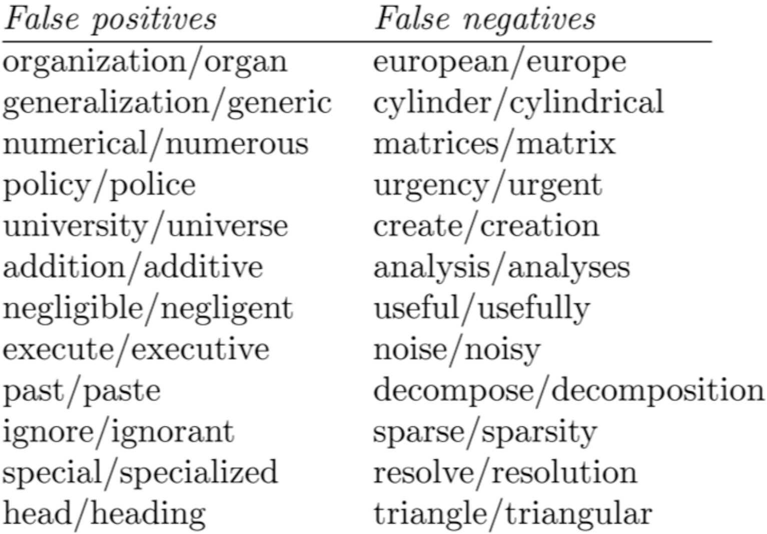
- Most common algorithm for stemming English
- Results suggest that it is at least as good as other stemming options
- Conventions + 5 phases of reductions
- Phases are applied sequentially
- Each phase consists of a set of commands.
    - Sample command: Delete final *ement* if what remains is longer than 1 character
        - replacement → replac
        - cement → cement
- Sample convention: Of the rules in a compound command, select the one that applies to the longest suffix.

Porter stemmer: A few rules

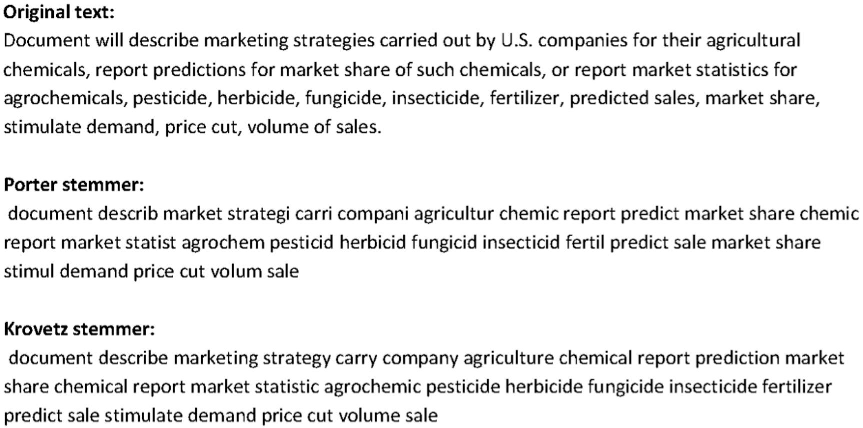
Rule			**Example**		
SSES	→	SS	caresses	→	caress
IES	→	I	ponies	→	poni
SS	→	SS	caress	→	caress
S	→		cats	→	cat

ATIONAL	→	ATE	relational	→	relate
TIONAL	→	TION	conditional	→	condition
ENCI	→	ENCE	valenci	→	valence
IZER	→	IZE	digitizer	→	digitize
...					

from `http://snowball.tartarus.org/algorithms/porter/stemmer.html`



False positives	*False negatives*
organization/organ	european/europe
generalization/generic	cylinder/cylindrical
numerical/numerous	matrices/matrix
policy/police	urgency/urgent
university/universe	create/creation
addition/additive	analysis/analyses
negligible/negligent	useful/usefully
execute/executive	noise/noisy
past/paste	decompose/decomposition
ignore/ignorant	sparse/sparsity
special/specialized	resolve/resolution
head/heading	triangle/triangular



Original text:
Document will describe marketing strategies carried out by U.S. companies for their agricultural chemicals, report predictions for market share of such chemicals, or report market statistics for agrochemicals, pesticide, herbicide, fungicide, insecticide, fertilizer, predicted sales, market share, stimulate demand, price cut, volume of sales.

Porter stemmer:
document describ market strategi carri compani agricultur chemic report predict market share chemic report market statist agrochem pesticid herbicid fungicid insecticid fertil predict sale market share stimul demand price cut volum sale

Krovetz stemmer:
document describe marketing strategy carry company agriculture chemical report prediction market share chemical report market statistic agrochemic pesticide herbicide fungicide insecticide fertilizer predict sale stimulate demand price cut volume sale

Three stemmers: A comparison

Sample text: Such an analysis can reveal features that are not easily visible from the variations in the individual genes and can lead to a picture of expression that is more biologically transparent and accessible to interpretation

Porter stemmer: such an **analysi** can reveal **featur** that **ar** not **easili visibl** from the **variat** in the **individu gene** and can lead to a **pictur** of **express** that is more **biolog transpar** and **access** to **interpret**

Lovins stemmer: such an **analys** can **reve featur** that **ar** not **eas vis** from **th vari** in **th individu gen** and can lead to a **pictur** of **expres** that is mor **biolog transpar** and acces to **interpres**

Paice stemmer: such an analys can rev feat that are not easy vis from the vary in the individ gen and can lead to a pict of express that is mor biolog transp and access to interpret

Outline

1. Documents
2. Terms
    - General + Non-English
    - Case folding
    - stop words
    - Stemming
    - lemmatization
3. Skip pointers
4. Phrase queries

Lemmatization

lemmatization(wiki)

in linguistics, is the process of grouping together the different inflected forms of a word so they can be analysed as a single item.

- use vocabulary
- Reduce inflectional/variant forms to base form
- Example:
  - *am, are, is was* → *be*
  - *better* → *good*
  - *car, cars, car's, cars'* → *car*
  - *the boy's cars are different colors* → *the boy car be different color*
- Lemmatization implies doing "proper" reduction to dictionary headword form (the lemma).

Does stemming/lematization improve effectiveness?

- In general, stemming increases effectiveness for some queries, and decreases effectiveness for others.
- Queries where stemming is likely to help: [tartan sweaters], [sightseeing tour san francisco]
- (equivalence classes: {sweater,sweaters}, {tour,tours})
- Porter Stemmer equivalence class
  oper = operate operating operates operation operative operatives operational.
- Queries where stemming hurts: [operational AND research], [operating AND system], [operative AND dentistry]

thesauri and soundex

- Do we handle synonyms and homonyms?
  - E.g., by hand-constructed equivalence classes
  - car = automobile color = colour
- We can rewrite to form equivalence-class terms
  - When the document contains automobile, index it under car-automobile (and vice-versa)
- Or we can expand a query
  - When the query contains automobile, look under car as well
- What about spelling mistakes?
- One approach is Soundex, which forms equivalence classes of words based on phonetic heuristics
- More in IIR 3 and IIR 9

More equivalence classing

- Soundex: IIR 3 (phonetic equivalence, Muller = Mueller)
- Thesauri: IIR 9 (semantic equivalence, car = automobile)

Outline

1. Documents
2. Terms
    - General + Non-English
    - Case folding
    - stop words
    - Stemming
    - lemmatization
3. Skip pointers
4. Phrase queries

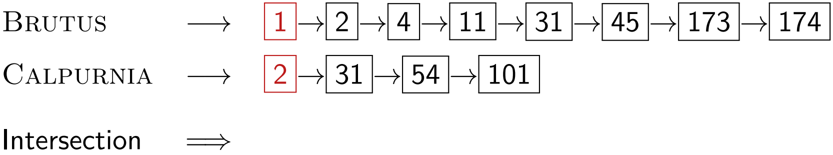
Recall basic intersection algorithm

$\textsc{Brutus} \longrightarrow \boxed{1} \rightarrow \boxed{2} \rightarrow \boxed{4} \rightarrow \boxed{11} \rightarrow \boxed{31} \rightarrow \boxed{45} \rightarrow \boxed{173} \rightarrow \boxed{174}$

$\textsc{Calpurnia} \longrightarrow \boxed{2} \rightarrow \boxed{31} \rightarrow \boxed{54} \rightarrow \boxed{101}$

Intersection $\Longrightarrow$

Recall basic intersection algorithm



Brutus ⟶ 1 → 2 → 4 → 11 → 31 → 45 → 173 → 174

Calpurnia ⟶ 2 → 31 → 54 → 101

Intersection ⟹

Recall basic intersection algorithm

BRUTUS $\longrightarrow$ $\boxed{1} \rightarrow \boxed{2} \rightarrow \boxed{4} \rightarrow \boxed{11} \rightarrow \boxed{31} \rightarrow \boxed{45} \rightarrow \boxed{173} \rightarrow \boxed{174}$

CALPURNIA $\longrightarrow$ $\boxed{2} \rightarrow \boxed{31} \rightarrow \boxed{54} \rightarrow \boxed{101}$

Intersection $\Longrightarrow$

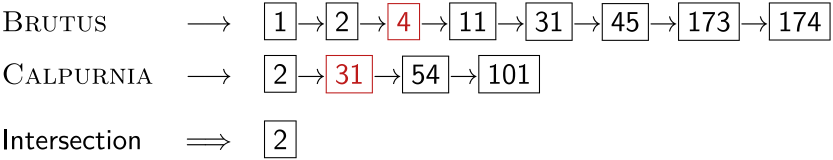
Recall basic intersection algorithm

$$\text{BRUTUS} \longrightarrow \boxed{1} \rightarrow \boxed{2} \rightarrow \boxed{4} \rightarrow \boxed{11} \rightarrow \boxed{31} \rightarrow \boxed{45} \rightarrow \boxed{173} \rightarrow \boxed{174}$$

$$\text{CALPURNIA} \longrightarrow \boxed{2} \rightarrow \boxed{31} \rightarrow \boxed{54} \rightarrow \boxed{101}$$

$$\text{Intersection} \Longrightarrow \boxed{2}$$

Recall basic intersection algorithm

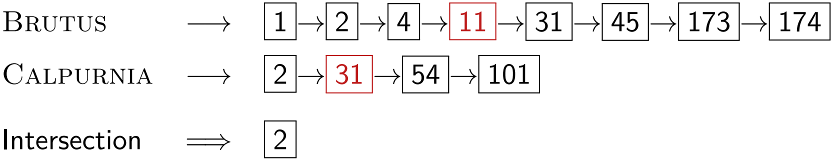


Brutus → 1 → 2 → 4 → 11 → 31 → 45 → 173 → 174

Calpurnia → 2 → 31 → 54 → 101

Intersection ⟹ 2

Recall basic intersection algorithm

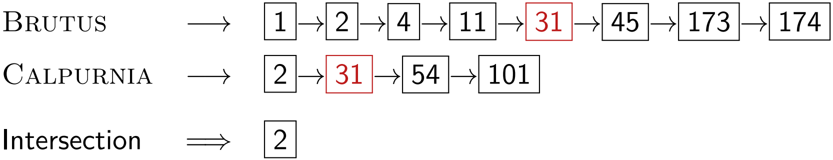


BRUTUS ⟶ 1 → 2 → 4 → 11 → 31 → 45 → 173 → 174

CALPURNIA ⟶ 2 → 31 → 54 → 101

Intersection ⟹ 2

Recall basic intersection algorithm

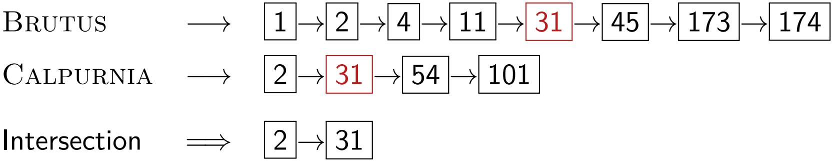


BRUTUS ⟶ $\boxed{1} \to \boxed{2} \to \boxed{4} \to \boxed{11} \to \boxed{31} \to \boxed{45} \to \boxed{173} \to \boxed{174}$

CALPURNIA ⟶ $\boxed{2} \to \boxed{31} \to \boxed{54} \to \boxed{101}$

Intersection ⟹ $\boxed{2}$

Recall basic intersection algorithm

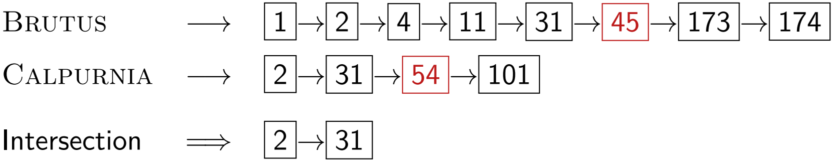


Brutus ⟶ 1 → 2 → 4 → 11 → 31 → 45 → 173 → 174

Calpurnia ⟶ 2 → 31 → 54 → 101

Intersection ⟹ 2 → 31

Recall basic intersection algorithm

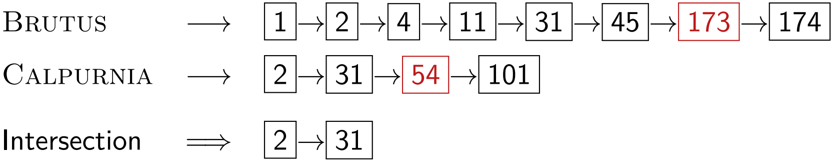


$\textsc{Brutus} \longrightarrow \boxed{1}\to\boxed{2}\to\boxed{4}\to\boxed{11}\to\boxed{31}\to\boxed{45}\to\boxed{173}\to\boxed{174}$

$\textsc{Calpurnia} \longrightarrow \boxed{2}\to\boxed{31}\to\boxed{54}\to\boxed{101}$

Intersection $\Longrightarrow \boxed{2}\to\boxed{31}$

Recall basic intersection algorithm

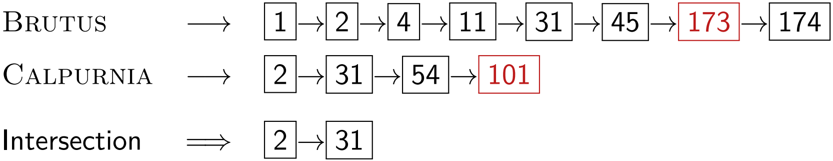


BRUTUS ⟶ 1 → 2 → 4 → 11 → 31 → 45 → 173 → 174

CALPURNIA ⟶ 2 → 31 → 54 → 101

Intersection ⟹ 2 → 31

Recall basic intersection algorithm



Brutus ⟶ 1 → 2 → 4 → 11 → 31 → 45 → 173 → 174

Calpurnia ⟶ 2 → 31 → 54 → 101

Intersection ⟹ 2 → 31

Recall basic intersection algorithm

$$\textsc{Brutus} \longrightarrow \boxed{1} \to \boxed{2} \to \boxed{4} \to \boxed{11} \to \boxed{31} \to \boxed{45} \to \boxed{173} \to \boxed{174}$$

$$\textsc{Calpurnia} \longrightarrow \boxed{2} \to \boxed{31} \to \boxed{54} \to \boxed{101}$$

$$\text{Intersection} \Longrightarrow \boxed{2} \to \boxed{31}$$

Recall basic intersection algorithm

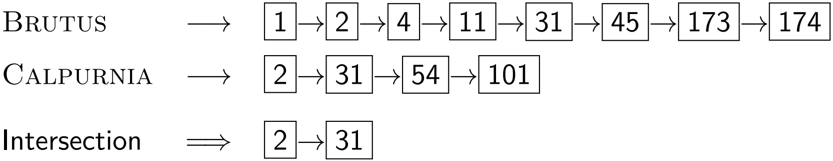
$$\textsc{Brutus} \longrightarrow \boxed{1} \to \boxed{2} \to \boxed{4} \to \boxed{11} \to \boxed{31} \to \boxed{45} \to \boxed{173} \to \boxed{174}$$

$$\textsc{Calpurnia} \longrightarrow \boxed{2} \to \boxed{31} \to \boxed{54} \to \boxed{101}$$

$$\text{Intersection} \Longrightarrow \boxed{2} \to \boxed{31}$$

- Linear in the length of the postings lists.

Recall basic intersection algorithm



BRUTUS $\longrightarrow$ $\boxed{1} \to \boxed{2} \to \boxed{4} \to \boxed{11} \to \boxed{31} \to \boxed{45} \to \boxed{173} \to \boxed{174}$

CALPURNIA $\longrightarrow$ $\boxed{2} \to \boxed{31} \to \boxed{54} \to \boxed{101}$

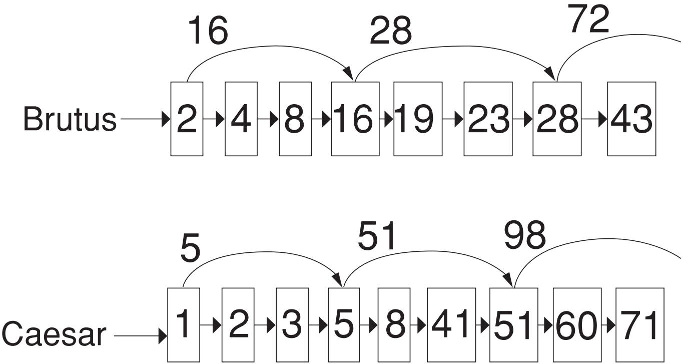
Intersection $\Longrightarrow$ $\boxed{2} \to \boxed{31}$

- Linear in the length of the postings lists.
- Can we do better?

Skip pointers

- Skip pointers allow us to skip postings that will not figure in the search results.
- This makes intersecting postings lists more efficient.
- Some postings lists contain several million entries – so efficiency can be an issue even if basic intersection is linear.
- Where do we put skip pointers?
- How do we make sure intsection results are correct?

Skip lists: Larger example



Intersecting with skip pointers

```
IntersectWithSkips($p_1, p_2$)
answer $\leftarrow \langle\ \rangle$
while $p_1 \neq$ NIL and $p_2 \neq$ NIL
do if $docID(p_1) = docID(p_2)$
      then Add($answer, docID(p_1)$)
           $p_1 \leftarrow next(p_1)$
           $p_2 \leftarrow next(p_2)$
      else if $docID(p_1) < docID(p_2)$
              then if $hasSkip(p_1)$ and $(docID(skip(p_1)) \leq docID(p_2))$
                      then while $hasSkip(p_1)$ and $(docID(skip(p_1)) \leq docID(p_2))$
                              do $p_1 \leftarrow skip(p_1)$
                      else $p_1 \leftarrow next(p_1)$
              else if $hasSkip(p_2)$ and $(docID(skip(p_2)) \leq docID(p_1))$
                      then while $hasSkip(p_2)$ and $(docID(skip(p_2)) \leq docID(p_1))$
                              do $p_2 \leftarrow skip(p_2)$
                      else $p_2 \leftarrow next(p_2)$
return answer
```

Where do we place skips?

- Tradeoff: number of items skipped vs. frequency skip can be taken
- More skips: Each skip pointer skips only a few items, but we can frequently use it.
- Fewer skips: Each skip pointer skips many items, but we can not use it very often.

Where do we place skips? (cont)

- Simple heuristic: for postings list of length P, use $\sqrt{P}$ evenly-spaced skip pointers.
- This ignores the distribution of query terms.
- Easy if the index is static; harder in a dynamic environment because of updates.
- How much do skip pointers help?
- They used to help a lot.

Outline

1. Documents
2. Terms
   - General + Non-English
   - Case folding
   - stop words
   - Stemming
   - lemmatization
3. Skip pointers
4. Phrase queries

Phrase queries

- We want to answer a query such as [stanford university] – as a phrase.
- *The inventor Stanford Ovshinsky never went to university* should not be a match.
- The concept of phrase query has proven easily understood by users.
- Significant part of web queries are phrase queries (explicitly entered or interpreted as such)
- Consequence for inverted index: it no longer suffices to store docIDs in postings lists.
- Two ways of extending the inverted index:
  - biword index
  - positional index

Biword indexes

- Index every consecutive pair of terms in the text as a phrase.
- For example, *Friends, Romans, Countrymen* would generate two biwords: *"friends romans"* and *"romans countrymen"*
- Each of these biwords is now a vocabulary term.
- Two-word phrases can now easily be answered.

Longer phrase queries

- A long phrase like *"stanford university palo alto"* can be represented as the Boolean query "STANFORD UNIVERSITY" AND "UNIVERSITY PALO" AND "PALO ALTO"
- We need to do post-filtering of hits to identify subset that actually contains the 4-word phrase.

Issues with biword indexes

- Why are biword indexes rarely used?
  - False positives, as noted above
  - Index blowup due to very large term vocabulary

Positional indexes

- more efficient than biword indexes.
- nonpositional index: each posting is just a docID
- positional index: each posting is a docID and a list of positions
- E.g.,

  TO, 993427:
  ⟨ 1: ⟨7, 18, 33, 72, 86, 231⟩;
  2: ⟨1, 17, 74, 222, 255⟩;
  ...⟩

Positional indexes: Example

Query: *"to_1 be_2 or_3 not_4 to_5 be_6"*

TO, 993427:

$\langle$ 1: $\langle$7, 18, 33, 72, 86, 231$\rangle$;
2: $\langle$1, 17, 74, 222, 255$\rangle$;
4: $\langle$8, 16, 190, 429, 433$\rangle$;
5: $\langle$363, 367$\rangle$;
7: $\langle$13, 23, 191$\rangle$; …$\rangle$

BE, 178239:

$\langle$ 1: $\langle$17, 25$\rangle$;
4: $\langle$17, 191, 291, 430, 434$\rangle$;
5: $\langle$14, 19, 101$\rangle$; …$\rangle$

Positional indexes: Example

Query: “*to*$_1$ *be*$_2$ *or*$_3$ *not*$_4$ *to*$_5$ *be*$_6$”

TO, 993427:

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

BE, 178239:

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

Positional indexes: Example

Query: *"to_1 be_2 or_3 not_4 to_5 be_6"*

TO, 993427:

$\langle$ 1: $\langle$7, 18, 33, 72, 86, 231$\rangle$;
2: $\langle$1, 17, 74, 222, 255$\rangle$;
4: $\langle$8, 16, 190, 429, 433$\rangle$;
5: $\langle$363, 367$\rangle$;
7: $\langle$13, 23, 191$\rangle$; …$\rangle$

BE, 178239:

$\langle$ 1: $\langle$17, 25$\rangle$;
4: $\langle$17, 191, 291, 430, 434$\rangle$;
5: $\langle$14, 19, 101$\rangle$; …$\rangle$

Positional indexes: Example

Query: *“to*$_1$ *be*$_2$ *or*$_3$ *not*$_4$ *to*$_5$ *be*$_6$*”*

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

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

Positional indexes: Example

Query: "*to_1 be_2 or_3 not_4 to_5 be_6*"

TO, 993427:

$\langle$ 1: $\langle$7, 18, 33, 72, 86, 231$\rangle$;
2: $\langle$1, 17, 74, 222, 255$\rangle$;
4: $\langle$8, 16, 190, 429, 433$\rangle$;
5: $\langle$363, 367$\rangle$;
7: $\langle$13, 23, 191$\rangle$; …$\rangle$

BE, 178239:

$\langle$ 1: $\langle$17, 25$\rangle$;
4: $\langle$17, 191, 291, 430, 434$\rangle$;
5: $\langle$14, 19, 101$\rangle$; …$\rangle$

Positional indexes: Example

Query: “$\textit{to}_1$ $\textit{be}_2$ $\textit{or}_3$ $\textit{not}_4$ $\textit{to}_5$ $\textit{be}_6$”

TO, 993427:

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

BE, 178239:

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

Positional indexes: Example

Query: “to_1 be_2 or_3 not_4 to_5 be_6”

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

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

Positional indexes: Example

Query: *"to_1 be_2 or_3 not_4 to_5 be_6"*

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

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

Positional indexes: Example

Query: “to_1 be_2 or_3 not_4 to_5 be_6”

TO, 993427:

⟨ 1: ⟨7, 18, 33, 72, 86, 231⟩;

2: ⟨1, 17, 74, 222, 255⟩;

4: ⟨8, 16, 190, 429, 433⟩;

5: ⟨363, 367⟩;

7: ⟨13, 23, 191⟩; …⟩

BE, 178239:

⟨ 1: ⟨17, 25⟩;

4: ⟨17, 191, 291, 430, 434⟩;

5: ⟨14, 19, 101⟩; …⟩

Positional indexes: Example

Query: "*to*$_1$ *be*$_2$ *or*$_3$ *not*$_4$ *to*$_5$ *be*$_6$"

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

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

Positional indexes: Example

Query: "*to*$_1$ *be*$_2$ *or*$_3$ *not*$_4$ *to*$_5$ *be*$_6$"

TO, 993427:

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

BE, 178239:

⟨ 1: ⟨17, 25⟩;
4: ⟨17, 191, 291, 430, 434⟩;
5: ⟨14, 19, 101⟩; …⟩

Positional indexes: Example

Query: “*to*$_1$ *be*$_2$ *or*$_3$ *not*$_4$ *to*$_5$ *be*$_6$”

TO, 993427:

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

BE, 178239:

⟨ 1: ⟨17, 25⟩;
4: ⟨17, 191, 291, 430, 434⟩;
5: ⟨14, 19, 101⟩; …⟩

Positional indexes: Example

Query: "*to*$_1$ *be*$_2$ *or*$_3$ *not*$_4$ *to*$_5$ *be*$_6$"

TO, 993427:

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

BE, 178239:

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

Positional indexes: Example

Query: “to_1 be_2 or_3 not_4 to_5 be_6”

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

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

Positional indexes: Example

Query: “to_1 be_2 or_3 not_4 to_5 be_6”

TO, 993427:

$\langle$ 1: $\langle$7, 18, 33, 72, 86, 231$\rangle$;
2: $\langle$1, 17, 74, 222, 255$\rangle$;
4: $\langle$8, 16, 190, 429, 433$\rangle$;
5: $\langle$363, 367$\rangle$;
7: $\langle$13, 23, 191$\rangle$; ...$\rangle$

BE, 178239:

$\langle$ 1: $\langle$17, 25$\rangle$;
4: $\langle$17, 191, 291, 430, 434$\rangle$;
5: $\langle$14, 19, 101$\rangle$; ...$\rangle$

Positional indexes: Example

Query: *“to*$_1$ *be*$_2$ *or*$_3$ *not*$_4$ *to*$_5$ *be*$_6$*”*

TO, 993427:

$\langle$ 1: $\langle 7, 18, 33, 72, 86, 231\rangle$;
2: $\langle 1, 17, 74, 222, 255\rangle$;
4: $\langle 8, 16, 190, 429, 433\rangle$;
5: $\langle 363, 367\rangle$;
7: $\langle 13, 23, 191\rangle$; ...$\rangle$

BE, 178239:

$\langle$ 1: $\langle 17, 25\rangle$;
4: $\langle 17, 191, 291, 430, 434\rangle$;
5: $\langle 14, 19, 101\rangle$; ...$\rangle$

Positional indexes: Example

Query: “*to*$_1$ *be*$_2$ *or*$_3$ *not*$_4$ *to*$_5$ *be*$_6$”

TO, 993427:

⟨ 1: ⟨7, 18, 33, 72, 86, 231⟩;

2: ⟨1, 17, 74, 222, 255⟩;

4: ⟨8, 16, 190, 429, 433⟩;

5: ⟨363, 367⟩;

7: ⟨13, 23, 191⟩; ...⟩

BE, 178239:

⟨ 1: ⟨17, 25⟩;

4: ⟨17, 191, 291, 430, 434⟩;

5: ⟨14, 19, 101⟩; ...⟩

Positional indexes: Example

Query: *"to_1 be_2 or_3 not_4 to_5 be_6"*

TO, 993427:

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

BE, 178239:

⟨ 1: ⟨17, 25⟩;
4: ⟨17, 191, 291, 430, 434⟩;
5: ⟨14, 19, 101⟩; …⟩

Positional indexes: Example

Query: "to_1 be_2 or_3 not_4 to_5 be_6"

TO, 993427:

$\langle$ 1: $\langle$7, 18, 33, 72, 86, 231$\rangle$;
2: $\langle$1, 17, 74, 222, 255$\rangle$;
4: $\langle$8, 16, 190, 429, 433$\rangle$;
5: $\langle$363, 367$\rangle$;
7: $\langle$13, 23, 191$\rangle$; ...$\rangle$

BE, 178239:

$\langle$ 1: $\langle$17, 25$\rangle$;
4: $\langle$17, 191, 291, 430, 434$\rangle$;
5: $\langle$14, 19, 101$\rangle$; ...$\rangle$

Document 4 is a match!

Proximity search

- We just saw how to use a positional index for phrase searches.
- We can also use it for proximity search.
- For example: employment /4 place
- Find all documents that contain EMPLOYMENT and PLACE within 4 words of each other.
- *Employment agencies that place healthcare workers are seeing growth* is a hit.
- *Employment agencies that have learned to adapt now place healthcare workers* is not a hit.

Proximity search

- Use the positional index
- Simplest algorithm: look at cross-product of positions of (i) EMPLOYMENT in document and (ii) PLACE in document
- Very inefficient for frequent words, especially stop words
- Note that we want to return the actual matching positions, not just a list of documents.
- This is important for dynamic summaries etc.

“Proximity” intersection

```
PositionalIntersect($p_1, p_2, k$)
answer $\leftarrow \langle\,\rangle$
while $p_1 \neq$ NIL and $p_2 \neq$ NIL
do if $docID(p_1) = docID(p_2)$
      then $l \leftarrow \langle\,\rangle$
           $pp_1 \leftarrow positions(p_1)$
           $pp_2 \leftarrow positions(p_2)$
           while $pp_1 \neq$ NIL
           do while $pp_2 \neq$ NIL
              do if $|pos(pp_1) - pos(pp_2)| \leq k$
                    then Add($l, pos(pp_2)$)
                    else if $pos(pp_2) > pos(pp_1)$
                            then break
                 $pp_2 \leftarrow next(pp_2)$
              while $l \neq \langle\,\rangle$ and $|l[0] - pos(pp_1)| > k$
              do Delete($l[0]$)
              for each $ps \in l$
              do Add($answer, \langle docID(p_1), pos(pp_1), ps\rangle$)
              $pp_1 \leftarrow next(pp_1)$
           $p_1 \leftarrow next(p_1)$
           $p_2 \leftarrow next(p_2)$
      else if $docID(p_1) < docID(p_2)$
             then $p_1 \leftarrow next(p_1)$
             else $p_2 \leftarrow next(p_2)$
return answer
```

Combination scheme

- Biword indexes and positional indexes can be combined.
- Many biwords are extremely frequent: Michael Jackson, Britney Spears etc
- For these biwords, increased speed compared to positional postings intersection is substantial.
- Combination scheme: Include frequent biwords as vocabulary terms in the index. Do all other phrases by positional intersection.
- Williams et al. (2004) evaluate a more sophisticated mixed indexing scheme. Faster than a positional index, at a cost of 26% more space for index.

recap

- tokenization. reduce the vocabulary
  - normalization. hyphen, numbers, case-folding, other languages. diacrtic (accent and umlauts)
  - remove stop words.
  - stemming, lemmatization. porter's stemming algorithm
- sublinear algorithm for posting intersection
  - skip pointer
- phrase queries
  - biword index
  - postional index
  - proximity search