Introduction to Information Retrieval
http://informationretrieval.org

IIR 2: The term vocabulary and postings lists

Hinrich Schütze

Institute for Natural Language Processing, Universität Stuttgart

2008.04.28
Overview

1. Recap
2. The term vocabulary
3. Skip pointers
4. Phrase queries
Outline

1. Recap
2. The term vocabulary
3. Skip pointers
4. Phrase queries
Inverted index

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

<table>
<thead>
<tr>
<th>Term</th>
<th>Dictionary Postings</th>
</tr>
</thead>
<tbody>
<tr>
<td>Brutus</td>
<td>1 2 4 11 31 45 173 174</td>
</tr>
<tr>
<td>Caesar</td>
<td>1 2 4 5 6 16 57 132 ...</td>
</tr>
<tr>
<td>Calpurnia</td>
<td>2 31 54 101</td>
</tr>
</tbody>
</table>
Intersecting two postings lists

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

Calpurnia → 2 → 31 → 54 → 101

Intersection →
Intersecting two postings lists

Brutus → 1 → 2 → 4 → 11 → 31 → 45 → 173 → 174
Calpurnia → 2 → 31 → 54 → 101
Intersection → →
Intersecting two postings lists

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

**Calpurnia** → 2 → 31 → 54 → 101

Intersection →
Intersecting two postings lists

Brutus: $\rightarrow 1 \rightarrow 2 \rightarrow 4 \rightarrow 11 \rightarrow 31 \rightarrow 45 \rightarrow 173 \rightarrow 174$

Calpurnia: $\rightarrow 2 \rightarrow 31 \rightarrow 54 \rightarrow 101$

Intersection: $\rightarrow 2$
Intersecting two postings lists

Brutus: 1 → 2 → 4 → 11 → 31 → 45 → 173 → 174
Calpurnia: 2 → 31 → 54 → 101
Intersection: 2
## Intersecting two postings lists

<table>
<thead>
<tr>
<th>Term</th>
<th>Postings List</th>
</tr>
</thead>
<tbody>
<tr>
<td>Brutus</td>
<td>1 → 2 → 4 → 11 → 31 → 45 → 173 → 174</td>
</tr>
<tr>
<td>Calpurnia</td>
<td>2 → 31 → 54 → 101</td>
</tr>
<tr>
<td>Intersection</td>
<td>2</td>
</tr>
</tbody>
</table>
Intersecting two postings lists

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

Calpurnia → 2 → 31 → 54 → 101

Intersection → 2
Intersecting two postings lists

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

**Calpurnia** → 2 → 31 → 54 → 101

Intersection → 2 → 31
Intersecting two postings lists

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

**Calpurnia** → 2 → 31 → 54 → 101

Intersection → 2 → 31
Intersecting two postings lists

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

**Calpurnia** → 2 → 31 → 54 → 101

Intersection → 2 → 31
Intersecting two postings lists

Brutus → 1 → 2 → 4 → 11 → 31 → 45 → 173 → 174
Calpurnia → 2 → 31 → 54 → 101
Intersection → 2 → 31
Intersecting two postings lists

\[
\begin{align*}
\text{BRUTUS} & \quad \rightarrow \quad 1 \rightarrow 2 \rightarrow 4 \rightarrow 11 \rightarrow 31 \rightarrow 45 \rightarrow 173 \rightarrow 174 \\
\text{CALPURNIA} & \quad \rightarrow \quad 2 \rightarrow 31 \rightarrow 54 \rightarrow 101 \\
\text{Intersection} & \quad \Rightarrow \quad 2 \rightarrow 31
\end{align*}
\]
Intersecting two postings lists

- **Brutus**: $1 \rightarrow 2 \rightarrow 4 \rightarrow 11 \rightarrow 31 \rightarrow 45 \rightarrow 173 \rightarrow 174$
- **Calpurnia**: $2 \rightarrow 31 \rightarrow 54 \rightarrow 101$
- **Intersection**: $2 \rightarrow 31$

- Linear in the length of the postings lists.
Constructing the inverted index: Sort postings

<table>
<thead>
<tr>
<th>term</th>
<th>docID</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>1</td>
</tr>
<tr>
<td>did</td>
<td>1</td>
</tr>
<tr>
<td>enact</td>
<td>1</td>
</tr>
<tr>
<td>julius</td>
<td>1</td>
</tr>
<tr>
<td>caesar</td>
<td>1</td>
</tr>
<tr>
<td>I</td>
<td>1</td>
</tr>
<tr>
<td>was</td>
<td>1</td>
</tr>
<tr>
<td>killed</td>
<td>1</td>
</tr>
<tr>
<td>i’</td>
<td>1</td>
</tr>
<tr>
<td>the</td>
<td>1</td>
</tr>
<tr>
<td>capitol</td>
<td>1</td>
</tr>
<tr>
<td>brutus</td>
<td>1</td>
</tr>
<tr>
<td>killed</td>
<td>1</td>
</tr>
<tr>
<td>me</td>
<td>1</td>
</tr>
<tr>
<td>so</td>
<td>2</td>
</tr>
<tr>
<td>let</td>
<td>2</td>
</tr>
<tr>
<td>it</td>
<td>2</td>
</tr>
<tr>
<td>be</td>
<td>2</td>
</tr>
<tr>
<td>with</td>
<td>2</td>
</tr>
<tr>
<td>caesar</td>
<td>2</td>
</tr>
<tr>
<td>the</td>
<td>2</td>
</tr>
<tr>
<td>noble</td>
<td>2</td>
</tr>
<tr>
<td>brutus</td>
<td>2</td>
</tr>
<tr>
<td>hath</td>
<td>2</td>
</tr>
<tr>
<td>told</td>
<td>2</td>
</tr>
<tr>
<td>you</td>
<td>2</td>
</tr>
<tr>
<td>caesar</td>
<td>2</td>
</tr>
<tr>
<td>was</td>
<td>2</td>
</tr>
<tr>
<td>ambitious</td>
<td>2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>term</th>
<th>docID</th>
</tr>
</thead>
<tbody>
<tr>
<td>ambitious</td>
<td>2</td>
</tr>
<tr>
<td>be</td>
<td>2</td>
</tr>
<tr>
<td>brutus</td>
<td>1</td>
</tr>
<tr>
<td>brutus</td>
<td>2</td>
</tr>
<tr>
<td>capitol</td>
<td>1</td>
</tr>
<tr>
<td>caesar</td>
<td>1</td>
</tr>
<tr>
<td>caesar</td>
<td>2</td>
</tr>
<tr>
<td>did</td>
<td>1</td>
</tr>
<tr>
<td>enact</td>
<td>1</td>
</tr>
<tr>
<td>hath</td>
<td>1</td>
</tr>
<tr>
<td>i’</td>
<td>1</td>
</tr>
<tr>
<td>I</td>
<td>1</td>
</tr>
<tr>
<td>I</td>
<td>1</td>
</tr>
<tr>
<td>i’</td>
<td>1</td>
</tr>
<tr>
<td>it</td>
<td>2</td>
</tr>
<tr>
<td>julius</td>
<td>1</td>
</tr>
<tr>
<td>killed</td>
<td>1</td>
</tr>
<tr>
<td>killed</td>
<td>1</td>
</tr>
<tr>
<td>let</td>
<td>2</td>
</tr>
<tr>
<td>l</td>
<td>1</td>
</tr>
<tr>
<td>l</td>
<td>1</td>
</tr>
<tr>
<td>me</td>
<td>1</td>
</tr>
<tr>
<td>me</td>
<td>1</td>
</tr>
<tr>
<td>me</td>
<td>1</td>
</tr>
<tr>
<td>noble</td>
<td>2</td>
</tr>
<tr>
<td>so</td>
<td>2</td>
</tr>
<tr>
<td>the</td>
<td>1</td>
</tr>
<tr>
<td>the</td>
<td>2</td>
</tr>
<tr>
<td>the</td>
<td>2</td>
</tr>
<tr>
<td>told</td>
<td>2</td>
</tr>
<tr>
<td>told</td>
<td>2</td>
</tr>
<tr>
<td>you</td>
<td>2</td>
</tr>
<tr>
<td>you</td>
<td>2</td>
</tr>
<tr>
<td>was</td>
<td>1</td>
</tr>
<tr>
<td>was</td>
<td>2</td>
</tr>
<tr>
<td>with</td>
<td>2</td>
</tr>
</tbody>
</table>
Westlaw: Example queries

*Information need:* Information on the legal theories involved in preventing the disclosure of trade secrets by employees formerly employed by a competing company
*Query:* “trade secret” /s disclos! /s prevent /s employe!

*Information need:* Requirements for disabled people to be able to access a workplace
*Query:* disab! /p access! /s work-site work-place (employment /3 place)

*Information need:* Cases about a host’s responsibility for drunk guests
*Query:* host! /p (responsib! liab!) /p (intoxicat! drunk!) /p guest
Outline

1. Recap
2. The term vocabulary
3. Skip pointers
4. Phrase queries
Terms and documents

• Last lecture: Simple Boolean retrieval system
Terms and documents

- Last lecture: Simple Boolean retrieval system
- Our assumptions were:
Terms and documents

- Last lecture: Simple Boolean retrieval system
- Our assumptions were:
  - We know what a document is.
Terms and documents

- Last lecture: Simple Boolean retrieval system
- Our assumptions were:
  - We know what a document is.
  - We know what a term is.
Terms and documents

- Last lecture: Simple Boolean retrieval system
- Our assumptions were:
  - We know what a document is.
  - We know what a term is.
- Both issues can be complex in reality.
Terms and documents

- Last lecture: Simple Boolean retrieval system
- Our assumptions were:
  - We know what a document is.
  - We know what a term is.
- Both issues can be complex in reality.
- We’ll look a little bit at what a document is.
Recap The term vocabulary Skip pointers Phrase queries

Terms and documents

- Last lecture: Simple Boolean retrieval system
- Our assumptions were:
  - We know what a document is.
  - We know what a term is.
- Both issues can be complex in reality.
- We’ll look a little bit at what a document is.
- But mostly at terms: How do we define and process the vocabulary of terms of a collection?
Before we can even start worrying about terms …
Before we can even start worrying about terms . . .

. . . need to deal with format and language of each document.
Parsing a document

- Before we can even start worrying about terms . . .
- . . . need to deal with format and language of each document.
- What format is it in? pdf, word, excel, html etc.
Parsing a document

- Before we can even start worrying about terms ...
- ...need to deal with format and language of each document.
- What format is it in? pdf, word, excel, html etc.
- What language is it in?
Parsing a document

- Before we can even start worrying about terms ...
- ...need to deal with format and language of each document.
- What format is it in? pdf, word, excel, html etc.
- What language is it in?
- What character set is in use?
Before we can even start worrying about terms...

...need to deal with format and language of each document.

What format is it in? pdf, word, excel, html etc.

What language is it in?

What character set is in use?

Each of these is a classification problem, which we will study later in this course (IIR 13).
Before we can even start worrying about terms . . .

. . . need to deal with format and language of each document.

What format is it in? pdf, word, excel, html etc.

What language is it in?

What character set is in use?

Each of these is a classification problem, which we will study later in this course (IIR 13).

Alternative: use heuristics
A single index usually contains terms of several languages.
Format/Language: Complications

- A single index usually contains terms of several languages.
- Sometimes a document or its components contain multiple languages/formats.
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
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

What is the document unit for indexing?
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
- What is the document unit for indexing?
- A file?
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
What is the document unit for indexing?
  - A file?
  - An email?
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
- What is the document unit for indexing?
  - A file?
  - An email?
- An email with 5 attachments?
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

What is the document unit for indexing?

- A file?
- An email?
- An email with 5 attachments?
- A group of files (ppt or latex in HTML)?
Terms
Definitions

- **Word** – A delimited string of characters as it appears in the text.
Definitions

- **Word** – A delimited string of characters as it appears in the text.
- **Term** – A “normalized” word (case, morphology, spelling etc); an equivalence class of words.
Definitions

- **Word** – A delimited string of characters as it appears in the text.
- **Term** – A “normalized” word (case, morphology, spelling etc); an equivalence class of words.
- **Token** – An instance of a word or term occurring in a document.
Definitions

- **Word** – A delimited string of characters as it appears in the text.
- **Term** – A “normalized” word (case, morphology, spelling etc); an equivalence class of words.
- **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.
Type/token distinction: Example

- *In June, the dog likes to chase the cat in the barn.*
Type/token distinction: Example

- *In June, the dog likes to chase the cat in the barn.*
- How many tokens? How many types?
Input:

Friends, Romans, countrymen.  So let it be with Caesar  . . .
Recall: Inverted index construction

**Input:**

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

**Output:**

friend  roman  countryman  so  . . .
Recall: Inverted index construction

- **Input:**
  
  Friends, Romans, countrymen.  
  So let it be with Caesar . . .

- **Output:**
  
  friend  roman  countryman  so  . . .

- Each token is a candidate for a postings entry.
Recall: Inverted index construction

- 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?
Why tokenization is difficult – even in English

Example: *Mr. O’Neill thinks that the boys’ stories about Chile’s capital aren’t amusing.*

Tokenize this sentence
One word or two? (or several)

- Hewlett-Packard
One word or two? (or several)

- Hewlett-Packard
- State-of-the-art
One word or two? (or several)

- Hewlett-Packard
- State-of-the-art
- co-education
One word or two? (or several)

- Hewlett-Packard
- State-of-the-art
- co-education
- the hold-him-back-and-drag-him-away maneuver
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
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
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
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
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

- 3/12/91
Numbers

- 3/12/91
- 12/3/91
Numbers

- 3/12/91
- 12/3/91
- Mar 12, 1991
Numbers

- 3/12/91
- 12/3/91
- Mar 12, 1991
- B-52
Numbers

- 3/12/91
- 12/3/91
- Mar 12, 1991
- B-52
- 100.2.86.144
Numbers

- 3/12/91
- 12/3/91
- Mar 12, 1991
- B-52
- 100.2.86.144
- (800) 234-2333
Numbers

- 3/12/91
- 12/3/91
- Mar 12, 1991
- B-52
- 100.2.86.144
- (800) 234-2333
- 800.234.2333
Numbers

- 3/12/91
- 12/3/91
- Mar 12, 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.
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 in Chinese

The two characters can be treated as one word meaning ‘monk’ or as a sequence of two words meaning ‘and’ and ‘still’.
Other cases of “no whitespace”

- Compounds in Dutch and German
Other cases of “no whitespace”

- Compounds in Dutch and German
- Computerlinguistik → Computer + Linguistik
Other cases of “no whitespace”

- Compounds in Dutch and German
- Computerlinguistik → Computer + Linguistik
- Lebensversicherungsgesellschaftsangestellter
Other cases of “no whitespace”

- Compounds in Dutch and German
- Computerlinguistik → Computer + Linguistik
- Lebensversicherungsgesellschaftsangestellter
  → leben + versicherung + gesellschaft + angestellter
Other cases of “no whitespace”

- Compounds in Dutch and German
  - Computerlinguistik $\rightarrow$ Computer + Linguistik
  - Lebensversicherungsgesellschaftsangestellter
    $\rightarrow$ leben + versicherung + gesellschaft + angestellter
- Inuit: tusaaatsiarunnannngittualuujunga (I can’t hear very well.)
Other cases of “no whitespace”

- Compounds in Dutch and German
  - Computerlinguistik → Computer + Linguistik
- Lebensversicherungsgesellschaftsangestellter
  - → leben + versicherung + gesellschaft + angestellter
- Inuit: tusaatsiarunnanngittualuujunga (I can’t hear very well.)
- Swedish, Finnish, Greek, Urdu, many other languages
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、もっていない」を募集します。皆様が日ごろ「もっていない」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!
\[
\begin{align*}
\text{كتاب} & \iff \text{un bātik} \\
/\text{kitābun}/ & \text{‘a book’}
\end{align*}
\]
Recap

The term vocabulary

Skip pointers

Phrase queries

Arabic script: Bidirectionality

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

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

Bidirectionality is not a problem if text is coded in Unicode.
Back to English
Normalization

- Need to “normalize” terms in indexed text as well as query terms into the same form.
Normalization

- Need to “normalize” terms in indexed text as well as query terms into the same form.
- Example: We want to match *U.S.A.* and *USA*
Normalization

- Need to “normalize” terms in indexed text as well as query terms into the same form.
- Example: We want to match *U.S.A.* and *USA*
- We most commonly implicitly define equivalence classes of terms.
Normalization

- Need to “normalize” terms in indexed text as well as query terms into the same form.
- Example: We want to match *U.S.A.* and *USA*
- We most commonly implicitly define equivalence classes of terms.
- Alternatively: do asymmetric expansion
Normalization

- Need to “normalize” terms in indexed text as well as query terms into the same form.
- Example: We want to match *U.S.A.* and *USA*
- We most commonly implicitly define *equivalence classes* of terms.
- Alternatively: do asymmetric expansion
  - window → window, windows
Normalization

- Need to “normalize” terms in indexed text as well as query terms into the same form.
- Example: We want to match *U.S.A.* and *USA*
- We most commonly implicitly define equivalence classes of terms.
- Alternatively: do asymmetric expansion
  - window → window, windows
  - windows → Windows, windows
Normalization

- Need to “normalize” terms in indexed text as well as query terms into the same form.
- Example: We want to match *U.S.A.* and *USA*
- We most commonly implicitly define equivalence classes of terms.
- Alternatively: do asymmetric expansion
  - window → window, windows
  - windows → Windows, windows
  - Windows (no expansion)
Need to “normalize” terms in indexed text as well as query terms into the same form.

Example: We want to match *U.S.A.* and *USA*

We most commonly implicitly define *equivalence classes* of terms.

Alternatively: do asymmetric expansion

- window → window, windows
- windows → Windows, windows
- Windows (no expansion)

More powerful, but less efficient
Normalization

- Need to “normalize” terms in indexed text as well as query terms into the same form.
- Example: We want to match *U.S.A.* and *USA*
- We most commonly implicitly define equivalence classes of terms.
- Alternatively: do asymmetric expansion
  - window → window, windows
  - windows → Windows, windows
  - Windows (no expansion)
- More powerful, but less efficient
- Why don’t you want to put *window*, *Window*, *windows*, and *Windows* in the same equivalence class?
Normalization: Other languages

- Accents: résumé vs. resume (simple omission of accent)
Normalization: Other languages

- Accents: résumé vs. resume (simple omission of accent)
- Umlauts: Universität vs. Universitaet (substitution with special letter sequence “ae”)

Normalization: Other languages

- 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?
Normalization: Other languages

- 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. (Polish?)
Normalization: Other languages

- 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. (Polish?)
- Normalization and language detection interact.
Normalization: Other languages

- 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. (Polish?)
- Normalization and language detection interact.
- PETER WILL NICHT MIT. → MIT = mit
Normalization: Other languages

- 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. (Polish?)
- Normalization and language detection interact.
- \textit{PETER WILL NICHT MIT.} $\rightarrow$ MIT $\neq$ mit
- \textit{He got his PhD from MIT.} $\rightarrow$ MIT $\neq$ mit
Case folding

- Reduce all letters to lower case
Case folding

- Reduce all letters to lower case
- Possible exceptions: capitalized words in mid-sentence
Case folding

- Reduce all letters to lower case
- Possible exceptions: capitalized words in mid-sentence
- MIT vs. mit
Case folding

- Reduce all letters to lower case
- Possible exceptions: capitalized words in mid-sentence
  - MIT vs. mit
  - Fed vs. fed
Case folding

- Reduce all letters to lower case
- Possible exceptions: 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.
Stop words

- stop words = extremely common words which would appear to be of little value in helping select documents matching a user need
Stop words

- stop words = extremely common words which would appear to be of little value in helping select documents matching a user need
- Examples: a, an, and, are, as, at, be, by, for, from, has, he, in, is, it, its, of, on, that, the, to, was, were, will, with
Stop words

- stop words = extremely common words which would appear to be of little value in helping select documents matching a user need
- Examples: a, an, and, are, as, at, be, by, for, from, has, he, in, is, it, its, of, on, that, the, to, was, were, will, with
- Stop word elimination used to be standard in older IR systems.
Stop words

- stop words = extremely common words which would appear to be of little value in helping select documents matching a user need
- Examples: *a, an, and, are, as, at, be, by, for, from, has, he, in, is, it, its, of, on, that, the, to, was, were, will, with*
- Stop word elimination used to be standard in older IR systems.
- But you need stop words for phrase queries, e.g. “King of Denmark”
Stop words

- stop words = extremely common words which would appear to be of little value in helping select documents matching a user need
- Examples: a, an, and, are, as, at, be, by, for, from, has, he, in, is, it, its, of, on, that, the, to, was, were, will, with
- Stop word elimination used to be standard in older IR systems.
- But you need stop words for phrase queries, e.g. “King of Denmark”
- Most web search engines index stop words.
More equivalence classing

- Soundex: IIR 3 (phonetic equivalence, Tchebyshev = Chebyssheff)
More equivalence classing

- **Soundex**: IIR 3 (phonetic equivalence, Tchebyshev = Chebysheff)
- **Thesauri**: IIR 9 (semantic equivalence, car = automobile)
What does Google do?

- Stop words
- Normalization
- Tokenization
- Lowercasing
- Stemming
- Non-latin alphabets
- Umlauts
- Compounds
- Numbers
Lemmatization

- Reduce inflectional/variant forms to base form
Lemmatization

- Reduce inflectional/variant forms to base form
- Example: *am, are, is* $\rightarrow$ *be*
Lemmatization

- Reduce inflectional/variant forms to base form
- Example: *am, are, is* → *be*
- Example: *car, cars, car’s, cars’* → *car*
Lemmatization

- Reduce inflectional/variant forms to base form
- Example: *am, are, is* → *be*
- Example: *car, cars, car’s, cars’* → *car*
- Example: *the boy’s cars are different colors* → *the boy car be different color*
Lemmatization

- Reduce inflectional/variant forms to base form
- Example: *am, are, is* → *be*
- Example: *car, cars, car’s, cars’* → *car*
- Example: *the boy’s cars are different colors* → *the boy car be different color*
- Lemmatization implies doing “proper” reduction to dictionary headword form (the **lemma**).
Lemmatization

- Reduce inflectional/variant forms to base form
- Example: \textit{am, are, is} $\rightarrow$ \textit{be}
- Example: \textit{car, cars, car’s, cars’} $\rightarrow$ \textit{car}
- Example: \textit{the boy’s cars are different colors} $\rightarrow$ \textit{the boy car be different color}
- Lemmatization implies doing “proper” reduction to dictionary headword form (the \textit{lemma}).
- Inflectional morphology (\textit{cutting} $\rightarrow$ \textit{cut}) vs. derivational morphology (\textit{destruction} $\rightarrow$ \textit{destroy})
Stemming

- Definition of stemming: Crude heuristic process that chops off the ends of words in the hope of achieving what “principled” lemmatization attempts to do with a lot of linguistic knowledge.
Stemming

- Definition of stemming: Crude heuristic process that chops off the ends of words in the hope of achieving what “principled” lemmatization attempts to do with a lot of linguistic knowledge.
- Language dependent
Stemming

- Definition of stemming: Crude heuristic process that chops off the ends of words in the hope of achieving what “principled” lemmatization attempts to do with a lot of linguistic knowledge.
- Language dependent
- Often inflectional and derivational
Stemming

- Definition of stemming: Crude heuristic process that chops off the ends of words in the hope of achieving what “principled” lemmatization attempts to do with a lot of linguistic knowledge.
- Language dependent
- Often inflectional and derivational
- Example for derivational: \textit{automate, automatic, automation} all reduce to \textit{automat}
Porter algorithm

- Most common algorithm for stemming English
Porter algorithm

- Most common algorithm for stemming English
- Results suggest that it is at least as good as other stemming options
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
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
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.
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
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
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 $\rightarrow$ replac
  - cement $\rightarrow$ cement
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

<table>
<thead>
<tr>
<th>Rule</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>SSES → SS</td>
<td>caresses → caress</td>
</tr>
<tr>
<td>IES → I</td>
<td>ponies → poni</td>
</tr>
<tr>
<td>SS → SS</td>
<td>caress → caress</td>
</tr>
<tr>
<td>S →</td>
<td>cats → cat</td>
</tr>
</tbody>
</table>
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 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.

*Lovins stemmer:* 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.

*Paice stemmer:* 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.
Does stemming improve effectiveness?

- In general, stemming increases effectiveness for some queries, and decreases effectiveness for others.
Does stemming improve effectiveness?

- In general, stemming increases effectiveness for some queries, and decreases effectiveness for others.
- Porter Stemmer equivalence class *oper* contains all of *operate operating operates operation operative operatives operat* *ional*. 
Does stemming improve effectiveness?

- In general, stemming increases effectiveness for some queries, and decreases effectiveness for others.
- Porter Stemmer equivalence class `oper` contains all of `operate operating operates operation operative operatives operat operational`.
- Queries where stemming hurts: “operational AND research”, “operating AND system”, “operative AND dentistry”
Interesting issues in your native language?
Outline

1 Recap

2 The term vocabulary

3 Skip pointers

4 Phrase queries
Recall basic intersection algorithm

Brutus → 1 → 2 → 4 → 11 → 31 → 45 → 173 → 174
Calpurnia → 2 → 31 → 54 → 101

Intersection →
Recall basic intersection algorithm

- **Brutus**: 1 → 2 → 4 → 11 → 31 → 45 → 173 → 174
- **Calpurnia**: 2 → 31 → 54 → 101
- **Intersection**: →
Recall basic intersection algorithm

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

Calpurnia → 2 → 31 → 54 → 101

Intersection →
Recall basic intersection algorithm

Brutus: \[ \rightarrow \quad 1 \rightarrow 2 \rightarrow 4 \rightarrow 11 \rightarrow 31 \rightarrow 45 \rightarrow 173 \rightarrow 174 \]

Calpurnia: \[ \rightarrow \quad 2 \rightarrow 31 \rightarrow 54 \rightarrow 101 \]

Intersection: \[ \rightarrow \quad 2 \]
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**  $\rightarrow$  1 $\rightarrow$ 2 $\rightarrow$ 4 $\rightarrow$ 11 $\rightarrow$ 31 $\rightarrow$ 45 $\rightarrow$ 173 $\rightarrow$ 174

**Calpurnia**  $\rightarrow$  2 $\rightarrow$ 31 $\rightarrow$ 54 $\rightarrow$ 101

**Intersection**  $\rightarrow$  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

Brutus → 1 → 2 → 4 → 11 → 31 → 45 → 173 → 174
Calpurnia → 2 → 31 → 54 → 101
Intersection → 2 → 31
Recap The term vocabulary  Skip pointers  Phrase queries

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

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

Can we do better?
Skip pointers

Skip pointers allow us to skip postings that will not figure in the search results.
Skip pointers allow us to skip postings that will not figure in the search results.

This makes intersecting postings lists more efficient.
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 though basic intersection is linear.
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 though basic intersection is linear.
- Where do we put skip pointers?
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 though basic intersection is linear.
- Where do we put skip pointers?
- How do we make sure results don’t change?
Skip lists

Recap The term vocabulary Skip pointers Phrase queries

Schütze: The term vocabulary and postings lists
Basic idea

**Brutus**

2 → 4 → 8 → 16 → 32 → 64 → 128

**Caesar**

1 → 2 → 3 → 5 → 8 → 17 → 21 → 31 → 75 → 81 → 84 → 89 → 92
Basic idea

Brutus  
\[
\begin{array}{cccccccc}
2 & 4 & 8 & 16 & 32 & 64 & 128 \\
\end{array}
\]

Caesar  
\[
\begin{array}{cccccccc}
1 & 2 & 3 & 5 & 8 & 17 & 21 & 31 & 75 & 81 & 84 & 89 & 92 \\
\end{array}
\]
Basic idea

Brutus: 2 → 4 → 8 → 16 → 32 → 64 → 128

Caesar: 1 → 2 → 3 → 5 → 8 → 17 → 21 → 31 → 75 → 81 → 84 → 89 → 92
Intersecting with skip pointers

\[\text{INTERSECTWITHSKIPS} (p_1, p_2)\]

1. \(answer \leftarrow \langle \rangle\)
2. \(\text{while } p_1 \neq \text{NIL and } p_2 \neq \text{NIL}\)
3. \(\text{do if } \text{docID}(p_1) = \text{docID}(p_2)\)
   4. \(\text{then } \text{ADD}(answer, \text{docID}(p_1))\)
   5. \(p_1 \leftarrow \text{next}(p_1)\)
   6. \(p_2 \leftarrow \text{next}(p_2)\)
4. \(\text{else if } \text{docID}(p_1) < \text{docID}(p_2)\)
   5. \(\text{then if } \text{hasSkip}(p_1) \text{ and } (\text{docID}(\text{skip}(p_1)) \leq \text{docID}(p_2))\)
   6. \(\text{then } p_1 \leftarrow \text{skip}(p_1)\)
   7. \(\text{else } p_1 \leftarrow \text{next}(p_1)\)
   8. \(\text{else if } \text{hasSkip}(p_2) \text{ and } (\text{docID}(\text{skip}(p_2)) \leq \text{docID}(p_1))\)
   9. \(\text{then } p_2 \leftarrow \text{skip}(p_2)\)
10. \(\text{else } p_2 \leftarrow \text{next}(p_2)\)
11. \(\text{return } answer\)
Where do we place skips?

- Tradeoff: number of items skipped vs. frequency skip can be taken
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.
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 cannot 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.
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.
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 relatively static; harder in a dynamic environment because of updates.
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 relatively static; harder in a dynamic environment because of updates.
- How much do skip pointers help?
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 relatively static; harder in a dynamic environment because of updates.
- How much do skip pointers help?
- They used to help lot.
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 relatively static; harder in a dynamic environment because of updates.
- How much do skip pointers help?
- They used to help lot.
- With today’s fast CPUs, they don’t help that much anymore.
Outline

1 Recap
2 The term vocabulary
3 Skip pointers
4 Phrase queries
Phrase queries

We want to answer a query such as “stanford university” – as a phrase.
Phrase queries

- We want to answer a query such as “stanford university” – as a phrase.
- Thus *The inventor Stanford Ovshinsky never went to university* shouldn’t be a match.
Phrase queries

- We want to answer a query such as “stanford university” – as a phrase.
- Thus *The inventor Stanford Ovshinsky never went to university* shouldn’t be a match.
- The concept of phrase query has proven easily understood by users.
Phrase queries

- We want to answer a query such as “stanford university” – as a phrase.
- Thus *The inventor Stanford Ovshinsky never went to university* shouldn’t be a match.
- The concept of phrase query has proven easily understood by users.
- About 10% of web queries are phrase queries.
Phrase queries

- We want to answer a query such as “stanford university” – as a phrase.
- Thus *The inventor Stanford Ovshinsky never went to university* shouldn’t be a match.
- The concept of phrase query has proven easily understood by users.
- About 10% of web queries are phrase queries.
- Consequence for inverted index: no longer suffices to store docIDs in postings lists.
Phrase queries

- We want to answer a query such as “stanford university” – as a phrase.
- Thus *The inventor Stanford Ovshinsky never went to university* shouldn’t be a match.
- The concept of phrase query has proven easily understood by users.
- About 10% of web queries are phrase queries.
- Consequence for inverted index: no longer suffices to store docIDs in postings lists.
- Any ideas?
Biword indexes

- Index every consecutive pair of terms in the text as a phrase.
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”
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.
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"
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.
Extended biwords

- Parse each document and perform part-of-speech tagging
Extended biwords

- Parse each document and perform part-of-speech tagging
- Bucket the terms into (say) nouns (N) and articles/prepositions (X)
Extended biwords

- Parse each document and perform part-of-speech tagging
- Bucket the terms into (say) nouns (N) and articles/prepositions (X)
- Now deem any string of terms of the form NX*N to be an extended biword
Extended biwords

- Parse each document and perform part-of-speech tagging
- Bucket the terms into (say) nouns (N) and articles/prepositions (X)
- Now deem any string of terms of the form NX*N to be an extended biword

Examples: catcher in the rye king of Denmark
N X X N N X N
Extended biwords

- Parse each document and perform part-of-speech tagging
- Bucket the terms into (say) nouns (N) and articles/prepositions (X)
- Now deem any string of terms of the form NX*N to be an extended biword
- Examples: catcher in the rye king of Denmark
  N X X N N X N
- Include extended biwords in the term vocabulary
Extended biwords

- Parse each document and perform part-of-speech tagging
- Bucket the terms into (say) nouns (N) and articles/prepositions (X)
- Now deem any string of terms of the form NX*N to be an extended biword
- Examples: catcher in the rye king of Denmark
  - N X X N N X N
- Include extended biwords in the term vocabulary
- Queries are processed accordingly
Issues with biword indexes

- Why are biword indexes rarely used?
Issues with biword indexes

- Why are biword indexes rarely used?
- False positives, as noted above
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

- Positional indexes are a more efficient alternative to biword indexes.
- Postings lists in a **nonpositional** index: each posting is just a docID
- Postings lists in a **positional** index: each posting is a docID and a list of positions
- Example: $to \_1$ $be \_2$ $or \_3$ $not \_4$ $to \_5$ $be \_6$

**TO**, 993427:

$\langle 1, 6: \langle 7, 18, 33, 72, 86, 231 \rangle ;$

$2, 5: \langle 1, 17, 74, 222, 255 \rangle ;$

$4, 5: \langle 8, 16, 190, 429, 433 \rangle ;$

$5, 2: \langle 363, 367 \rangle ;$

$7, 3: \langle 13, 23, 191 \rangle ; \ldots \rangle$

**BE**, 178239:

$\langle 1, 2: \langle 17, 25 \rangle ;$

$4, 5: \langle 17, 191, 291, 430, 434 \rangle ;$

$5, 3: \langle 14, 19, 101 \rangle ; \ldots \rangle$
Positional indexes

- Positional indexes are a more efficient alternative to biword indexes.
- Postings lists in a **nonpositional** index: each posting is just a docID.
- Postings lists in a **positional** index: each posting is a docID and a list of positions.
- Example: $to_1 \ be_2 \ or_3 \ not_4 \ to_5 \ be_6$

**TO**, 993427:
\[
\langle 1, 6: \langle 7, 18, 33, 72, 86, 231 \rangle; \\
2, 5: \langle 1, 17, 74, 222, 255 \rangle; \\
4, 5: \langle 8, 16, 190, 429, 433 \rangle; \\
5, 2: \langle 363, 367 \rangle; \\
7, 3: \langle 13, 23, 191 \rangle; \ldots \rangle
\]

**BE**, 178239:
\[
\langle 1, 2: \langle 17, 25 \rangle; \\
4, 5: \langle 17, 191, 291, 430, 434 \rangle; \\
5, 3: \langle 14, 19, 101 \rangle; \ldots \rangle
\]
Positional indexes

- Positional indexes are a more efficient alternative to biword indexes.
- Postings lists in a nonpositional index: each posting is just a docID
- Postings lists in a positional index: each posting is a docID and a list of positions
- Example: \(to_1 \ be_2 \ or_3 \ not_4 \ to_5 \ be_6\)

**TO**, 993427:
\[
\langle 1, 6: \langle 7, 18, 33, 72, 86, 231 \rangle; \ \\
2, 5: \langle 1, 17, 74, 222, 255 \rangle; \ \\
4, 5: \langle 8, 16, 190, 429, 433 \rangle; \ \\
5, 2: \langle 363, 367 \rangle; \ \\
7, 3: \langle 13, 23, 191 \rangle; \ldots \rangle
\]

**BE**, 178239:
\[
\langle 1, 2: \langle 17, 25 \rangle; \ \\
4, 5: \langle 17, 191, 291, 430, 434 \rangle; \ \\
5, 3: \langle 14, 19, 101 \rangle; \ldots \rangle
\]
Positional indexes

- Positional indexes are a more efficient alternative to biword indexes.
- Postings lists in a **nonpositional** index: each posting is just a docID
- Postings lists in a **positional** index: each posting is a docID and a list of positions
- Example: \( to_1 \ be_2 \ or_3 \ not_4 \ to_5 \ be_6 \)

**TO, 993427:**
\[
\langle 1, 6: \langle 7, 18, 33, 72, 86, 231 \rangle; \\
2, 5: \langle 1, 17, 74, 222, 255 \rangle; \\
4, 5: \langle 8, 16, 190, 429, 433 \rangle; \\
5, 2: \langle 363, 367 \rangle; \\
7, 3: \langle 13, 23, 191 \rangle; \ldots \rangle
\]

**BE, 178239:**
\[
\langle 1, 2: \langle 17, 25 \rangle; \\
4, 5: \langle 17, 191, 291, 430, 434 \rangle; \\
5, 3: \langle 14, 19, 101 \rangle; \ldots \rangle
\]
Positional indexes

- Positional indexes are a more efficient alternative to biword indexes.
- Postings lists in a nonpositional index: each posting is just a docID.
- Postings lists in a positional index: each posting is a docID and a list of positions.
- Example: \(to_1\) \(be_2\) \(or_3\) \(not_4\) \(to_5\) \(be_6\)

**TO, 993427:**
\[
\langle 1, 6: \langle 7, 18, 33, 72, 86, 231 \rangle; \\
2, 5: \langle 1, 17, 74, 222, 255 \rangle; \\
4, 5: \langle 8, 16, 190, 429, 433 \rangle; \\
5, 2: \langle 363, 367 \rangle; \\
7, 3: \langle 13, 23, 191 \rangle; \ldots \rangle
\]

**BE, 178239:**
\[
\langle 1, 2: \langle 17, 25 \rangle; \\
4, 5: \langle 17, 191, 291, 430, 434 \rangle; \\
5, 3: \langle 14, 19, 101 \rangle; \ldots \rangle
\]
Positional indexes

- Positional indexes are a more efficient alternative to biword indexes.
- Postings lists in a **nonpositional** index: each posting is just a docID
- Postings lists in a **positional** index: each posting is a docID and a list of positions
- Example: \textit{to}$_1$ \textit{be}$_2$ \textit{or}$_3$ \textit{not}$_4$ \textit{to}$_5$ \textit{be}$_6$

**TO**, 993427:
\[
\langle 1, 6: \langle 7, 18, 33, 72, 86, 231 \rangle; \\
2, 5: \langle 1, 17, 74, 222, 255 \rangle; \\
4, 5: \langle 8, 16, 190, 429, 433 \rangle; \\
5, 2: \langle 363, 367 \rangle; \\
7, 3: \langle 13, 23, 191 \rangle; \ldots \rangle
\]

**BE**, 178239:
\[
\langle 1, 2: \langle 17, 25 \rangle; \\
4, 5: \langle 17, 191, 291, 430, 434 \rangle; \\
5, 3: \langle 14, 19, 101 \rangle; \ldots \rangle
\]
Positional indexes

- Positional indexes are a more efficient alternative to biword indexes.
- Postings lists in a nonpositional index: each posting is just a docID
- Postings lists in a positional index: each posting is a docID and a list of positions
- Example: to₁ be₂ or₃ not₄ to₅ be₆

**TO, 993427:**

\[
\langle 1, 6: \langle 7, 18, 33, 72, 86, 231 \rangle; \\
2, 5: \langle 1, 17, 74, 222, 255 \rangle; \\
4, 5: \langle 8, 16, 190, 429, 433 \rangle; \\
5, 2: \langle 363, 367 \rangle; \\
7, 3: \langle 13, 23, 191 \rangle; \ldots \rangle
\]

**BE, 178239:**

\[
\langle 1, 5: \langle 17, 25 \rangle; \\
4, 5: \langle 17, 191, 291, 430, 434 \rangle; \\
5, 3: \langle 14, 19, 101 \rangle; \ldots \rangle
\]
Positional indexes

- Positional indexes are a more efficient alternative to biword indexes.
- Postings lists in a nonpositional index: each posting is just a docID
- Postings lists in a positional index: each posting is a docID and a list of positions
- Example: $to_1 \ be_2 \ or_3 \ not_4 \ to_5 \ be_6$

**TO, 993427:**

\[
\langle 1, 6: \langle 7, 18, 33, 72, 86, 231\rangle; \\
2, 5: \langle 1, 17, 74, 222, 255\rangle; \\
4, 5: \langle 8, 16, 190, 429, 433\rangle; \\
5, 2: \langle 363, 367\rangle; \\
7, 3: \langle 13, 23, 191\rangle; \ldots \rangle
\]

**BE, 178239:**

\[
\langle 1, 2: \langle 17, 25\rangle; \\
4, 5: \langle 17, 191, 291, 430, 434\rangle; \\
5, 3: \langle 14, 19, 101\rangle; \ldots \rangle
\]
Positional indexes

- Positional indexes are a more efficient alternative to biword indexes.
- Postings lists in a nonpositional index: each posting is just a docID
- Postings lists in a positional index: each posting is a docID and a list of positions
- Example: to₁ be₂ or₃ not₄ to₅ be₆

TO, 993427:
- 1, 6: ⟨7, 18, 33, 72, 86, 231⟩;
- 2, 5: ⟨1, 17, 74, 222, 255⟩;
- 4, 5: ⟨8, 16, 190, 429, 433⟩;
- 5, 2: ⟨363, 367⟩;
- 7, 3: ⟨13, 23, 191⟩; ... 

BE, 178239:
- 1, 2: ⟨17, 25⟩;
- 4, 5: ⟨17, 191, 291, 430, 434⟩;
- 5, 3: ⟨14, 19, 101⟩; ...
Positional indexes

- Positional indexes are a more efficient alternative to biword indexes.
- Postings lists in a *nonpositional* index: each posting is just a docID
- Postings lists in a *positional* index: each posting is a docID and a list of positions
- Example: \textit{to}_1 \textit{be}_2 \textit{or}_3 \textit{not}_4 \textit{to}_5 \textit{be}_6

**TO, 993427:**
\[
\langle 1, 6: \langle 7, 18, 33, 72, 86, 231 \rangle; \\
2, 5: \langle 1, 17, 74, 222, 255 \rangle; \\
4, 5: \langle 8, 16, 190, 429, 433 \rangle; \\
5, 2: \langle 363, 367 \rangle; \\
7, 3: \langle 13, 23, 191 \rangle; \ldots \rangle
\]

**BE, 178239:**
\[
\langle 1, 2: \langle 17, 25 \rangle; \\
4, 5: \langle 17, 191, 291, 430, 434 \rangle; \\
5, 3: \langle 14, 19, 101 \rangle; \ldots \rangle
\]
Positional indexes

- Positional indexes are a more efficient alternative to biword indexes.
- Postings lists in a nonpositional index: each posting is just a docID.
- Postings lists in a positional index: each posting is a docID and a list of positions.
- Example: to₁ be₂ or₃ not₄ to₅ be₆

TO, 993427:
  ⟨1, 6: ⟨7, 18, 33, 72, 86, 231⟩;
  2, 5: ⟨1, 17, 74, 222, 255⟩;
  4, 5: ⟨8, 16, 190, 429, 433⟩;
  5, 2: ⟨363, 367⟩;
  7, 3: ⟨13, 23, 191⟩; . . .⟩

BE, 178239:
  ⟨1, 2: ⟨17, 25⟩;
  4, 5: ⟨17, 191, 291, 430, 434⟩;
  5, 3: ⟨14, 19, 101⟩; . . .⟩
Positional indexes

- Positional indexes are a more efficient alternative to biword indexes.
- Postings lists in a nonpositional index: each posting is just a docID
- Postings lists in a positional index: each posting is a docID and a list of positions
- Example: \(to_1\) \(be_2\) \(or_3\) \(not_4\) \(to_5\) \(be_6\)

\(TO, 993427:\)
\[
\langle 1, 6: \langle 7, 18, 33, 72, 86, 231 \rangle;  \\
2, 5: \langle 1, 17, 74, 222, 255 \rangle;  \\
4, 5: \langle 8, 16, 190, 429, 433 \rangle;  \\
5, 2: \langle 363, 367 \rangle;  \\
7, 3: \langle 13, 23, 191 \rangle; \ldots \rangle
\]

\(BE, 178239:\)
\[
\langle 1, 2: \langle 17, 25 \rangle;  \\
4, 5: \langle 17, 191, 291, 430, 434 \rangle;  \\
5, 3: \langle 14, 19, 101 \rangle; \ldots \rangle
\]
Positional indexes

- Positional indexes are a more efficient alternative to biword indexes.
- Postings lists in a nonpositional index: each posting is just a docID
- Postings lists in a positional index: each posting is a docID and a list of positions
- Example: $to_1\ be_2\ or_3\ not_4\ to_5\ be_6$

TO, 993427:
\[
\langle 1, 6: \langle 7, 18, 33, 72, 86, 231 \rangle; \\
2, 5: \langle 1, 17, 74, 222, 255 \rangle; \\
4, 5: \langle 8, 16, 190, 429, 433 \rangle; \\
5, 2: \langle 363, 367 \rangle; \\
7, 3: \langle 13, 23, 191 \rangle; \ldots \rangle
\]

BE, 178239:
\[
\langle 1, 2: \langle 17, 25 \rangle; \\
4, 5: \langle 17, 191, 291, 430, 434 \rangle; \\
5, 3: \langle 14, 19, 101 \rangle; \ldots \rangle
\]
Positional indexes

- Positional indexes are a more efficient alternative to biword indexes.
- Postings lists in a nonpositional index: each posting is just a docID
- Postings lists in a positional index: each posting is a docID and a list of positions
- Example: $to_1 \ be_2 \ or_3 \ not_4 \ to_5 \ be_6$

**TO**, 993427:

  \[
  \langle 1, 6: \langle 7, 18, 33, 72, 86, 231 \rangle; \\
  2, 5: \langle 1, 17, 74, 222, 255 \rangle; \\
  4, 5: \langle 8, 16, 190, 429, 433 \rangle; \\
  5, 2: \langle 363, 367 \rangle; \\
  7, 3: \langle 13, 23, 191 \rangle; \ldots \rangle
  \]

**BE**, 178239:

  \[
  \langle 1, 2: \langle 17, 25 \rangle; \\
  4, 5: \langle 17, 191, 291, 430, 434 \rangle; \\
  5, 3: \langle 14, 19, 101 \rangle; \ldots \rangle
  \]
Positional indexes

- Positional indexes are a more efficient alternative to biword indexes.
- Postings lists in a nonpositional index: each posting is just a docID.
- Postings lists in a positional index: each posting is a docID and a list of positions.
- Example: to₁ be₂ or₃ not₄ to₅ be₆

TO, 993427:
\[\langle 1, 6: \langle 7, 18, 33, 72, 86, 231 \rangle; 2, 5: \langle 1, 17, 74, 222, 255 \rangle; 4, 5: \langle 8, 16, 190, 429, 433 \rangle; 5, 2: \langle 363, 367 \rangle; 7, 3: \langle 13, 23, 191 \rangle; \ldots \]\n
BE, 178239:
\[\langle 1, 2: \langle 17, 25 \rangle; 4, 5: \langle 17, 191, 291, 430, 434 \rangle; 5, 3: \langle 14, 19, 101 \rangle; \ldots \]
Positional indexes

- Positional indexes are a more efficient alternative to biword indexes.
- Postings lists in a nonpositional index: each posting is just a docID
- Postings lists in a positional index: each posting is a docID and a list of positions
- Example: \(to_1\) \(be_2\) \(or_3\) \(not_4\) \(to_5\) \(be_6\)

**TO, 993427:**
\[
\langle 1, 6: \langle 7, 18, 33, 72, 86, 231 \rangle; \\
2, 5: \langle 1, 17, 74, 222, 255 \rangle; \\
4, 5: \langle 8, 16, 190, 429, 433 \rangle; \\
5, 2: \langle 363, 367 \rangle; \\
7, 3: \langle 13, 23, 191 \rangle \rangle
\]

**BE, 178239:**
\[
\langle 1, 2: \langle 17, 25 \rangle; \\
4, 5: \langle 17, 191, 291, 430, 434 \rangle; \\
5, 3: \langle 14, 19, 101 \rangle \rangle
\]
Positional indexes

- Positional indexes are a more efficient alternative to biword indexes.
- Postings lists in a nonpositional index: each posting is just a docID
- Postings lists in a positional index: each posting is a docID and a list of positions
- Example: to₁ be₂ or₃ not₄ to₅ be₆

TO, 993427:
  ⟨ 1, 6: ⟨7, 18, 33, 72, 86, 231⟩;
    2, 5: ⟨1, 17, 74, 222, 255⟩;
    4, 5: ⟨8, 16, 190, 429, 433⟩;
    5, 2: ⟨363, 367⟩;
    7, 3: ⟨13, 23, 191⟩; . . .⟩

BE, 178239:
  ⟨ 1, 2: ⟨17, 25⟩;
    4, 5: ⟨17, 191, 291, 430, 434⟩;
    5, 3: ⟨14, 19, 101⟩; . . .⟩
Positional indexes

- Positional indexes are a more efficient alternative to biword indexes.
- Postings lists in a nonpositional index: each posting is just a docID
- Postings lists in a positional index: each posting is a docID and a list of positions
- Example: to₁ be₂ or₃ not₄ to₅ be₆

TO, 993427:

\[ \langle 1, 6: \langle 7, 18, 33, 72, 86, 231 \rangle; 2, 5: \langle 1, 17, 74, 222, 255 \rangle; 4, 5: \langle 8, 16, 190, 429, 433 \rangle; 5, 2: \langle 363, 367 \rangle; 7, 3: \langle 13, 23, 191 \rangle; \ldots \rangle \]

BE, 178239:

\[ \langle 1, 2: \langle 17, 25 \rangle; 4, 5: \langle 17, 191, 291, 430, 434 \rangle; 5, 3: \langle 14, 19, 101 \rangle; \ldots \rangle \]
Positional indexes

- Positional indexes are a more efficient alternative to biword indexes.
- Postings lists in a **nonpositional** index: each posting is just a docID
- Postings lists in a **positional** index: each posting is a docID and a list of positions
- Example: \(to_1 \, be_2 \, or_3 \, not_4 \, to_5 \, be_6\)

**TO**, 993427:
\[
\langle 1, 6: \langle 7, 18, 33, 72, 86, 231 \rangle; \\
2, 5: \langle 1, 17, 74, 222, 255 \rangle; \\
4, 5: \langle 8, 16, 190, 429, 433 \rangle; \\
5, 2: \langle 363, 367 \rangle; \\
7, 3: \langle 13, 23, 191 \rangle \ldots \rangle
\]

**BE**, 178239:
\[
\langle 1, 2: \langle 17, 25 \rangle; \\
4, 5: \langle 17, 191, 291, 430, 434 \rangle; \\
5, 3: \langle 14, 19, 101 \rangle \ldots \rangle
\]

Document 4 is a match!
Exercise

Shown below is a portion of a positional index in the format: term: doc1: 〈position1, position2, ...〉; doc2: 〈position1, position2, ...〉; etc.

ANGELS: 2: 〈36,174,252,651〉; 4: 〈12,22,102,432〉; 7: 〈17〉;
FEAR: 2: 〈87,704,722,901〉; 4: 〈13,43,113,433〉; 7: 〈18,328,528〉;
IN: 2: 〈3,37,76,444,851〉; 4: 〈10,20,110,470,500〉; 7: 〈5,15,25,195〉;
RUSH: 2: 〈2,66,194,321,702〉; 4: 〈9,69,149,429,569〉; 7: 〈4,14,404〉;
TO: 2: 〈47,86,234,999〉; 4: 〈14,24,774,944〉; 7: 〈199,319,599,709〉;
TREAD: 2: 〈57,94,333〉; 4: 〈15,35,155〉; 7: 〈20,320〉;
WHERE: 2: 〈67,124,393,1001〉; 4: 〈11,41,101,421,431〉; 7: 〈16,36,736〉;

Which document(s) if any match each of the following two queries, where each expression within quotes is a phrase query?: “fools rush in”, “fools rush in” AND “angels fear to tread”
Proximity search

- We just saw how to use a positional index for phrase searches.
We just saw how to use a positional index for phrase searches.

We can also use it for proximity search.
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 /3 place
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 /3 place
- Find all documents that contain EMPLOYMENT and PLACE within 3 words of each other.
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 /3 place
- Find all documents that contain EMPLOYMENT and PLACE within 3 words of each other.
- Employment agencies that place healthcare workers are seeing growth is a hit.
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 /3 place
- Find all documents that contain EMPLOYMENT and PLACE within 3 words of each other.

*Employment agencies that place healthcare workers are seeing growth* is a hit.

*Employment agencies that help place healthcare workers are seeing growth* is not a hit.
Proximity search

- Simplest algorithm: look at cross-product of positions of (i) EMPLOYMENT in document and (ii) PLACE in document
Proximity search

- 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
Proximity search

- 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.
Proximity search

- 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.
Phrase queries

"Proximity" intersection

**POSITIONALINTERSECT**($p_1, p_2, k$)

1. $answer \leftarrow \langle \rangle$
2. while $p_1 \neq \text{NIL}$ and $p_2 \neq \text{NIL}$
3. do if $\text{docID}(p_1) = \text{docID}(p_2)$
   
   then $l \leftarrow \langle \rangle$

   pp$_1 \leftarrow \text{positions}(p_1)$

   pp$_2 \leftarrow \text{positions}(p_2)$

   while $pp_1 \neq \text{NIL}$

   do while $pp_2 \neq \text{NIL}$

   do if $|\text{pos}(pp_1) - \text{pos}(pp_2)| \leq k$

   then $\text{ADD}(l, \text{pos}(pp_2))$

   else if $\text{pos}(pp_2) > \text{pos}(pp_1)$

   then break

   pp$_2 \leftarrow \text{next}(pp_2)$

   while $l \neq \langle \rangle$ and $|l[0] - \text{pos}(pp_1)| > k$

   do $\text{DELETE}(l[0])$

   for each $ps \in l$

   do $\text{ADD}(answer, \langle \text{docID}(p_1), \text{pos}(pp_1), ps \rangle)$

   pp$_1 \leftarrow \text{next}(pp_1)$

4. $p_1 \leftarrow \text{next}(p_1)$

5. $p_2 \leftarrow \text{next}(p_2)$

else if $\text{docID}(p_1) < \text{docID}(p_2)$

then $p_1 \leftarrow \text{next}(p_1)$

else $p_2 \leftarrow \text{next}(p_2)$

return $answer$
Biword indexes and positional indexes can be profitably combined.
Combination scheme

- Biword indexes and positional indexes can be profitably combined.
- Many biwords are extremely frequent: Michael Jackson, Britney Spears etc
Combination scheme

- Biword indexes and positional indexes can be profitably 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

- Biword indexes and positional indexes can be profitably 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.
Combination scheme

- Biword indexes and positional indexes can be profitably 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.
“Positional” queries on Google

- For web search engines, positional queries are much more expensive than regular Boolean queries.
“Positional” queries on Google

- For web search engines, positional queries are much more expensive than regular Boolean queries.
- Let’s look at the example of phrase queries.
For web search engines, positional queries are much more expensive than regular Boolean queries.

Let’s look at the example of phrase queries.

Why are they more expensive than regular Boolean queries?
“Positional” queries on Google

- For web search engines, positional queries are much more expensive than regular Boolean queries.
- Let’s look at the example of phrase queries.
- Why are they more expensive than regular Boolean queries?
- Can you demonstrate on Google that phrase queries are more expensive than Boolean queries?
Resources

- Chapter 2 of IIR
Chapter 2 of IIR

Resources at http://ifnlp.org/ir
Resources

- Chapter 2 of IIR
- Resources at http://ifnlp.org/ir
- Porter stemmer