NPFL103: Information Retrieval (1)
Introduction, Boolean retrieval, Inverted index, Text processing

Pavel Pecina
pecina@ufal.mff.cuni.cz

Institute of Formal and Applied Linguistics
Faculty of Mathematics and Physics
Charles University

Original slides are courtesy of Hinrich Schütze, University of Stuttgart.
Introduction
**Information retrieval** (IR) is **finding** material (**usually documents**) of an **unstructured** nature (**usually text**) that satisfies an **information need** from within **large collections** (**usually stored on computers**).
Boolean retrieval
Boolean retrieval

- Boolean model is arguably the simplest model to base an information retrieval system on.

- Queries are Boolean expressions, e.g., **CAESAR AND BRUTUS**

- The search engine returns all documents that satisfy the Boolean expression.

Does Google use the Boolean model?
Does Google use the Boolean model?

- On Google, the default interpretation of a query \([w_1 \ w_2 \ ... \ w_n]\) is

  \[w_1 \ \text{AND} \ w_2 \ \text{AND} \ ... \ \text{AND} \ w_n\]

- Cases where you get hits that do not contain one of the \(w_i\):
  - anchor text
  - page contains variant of \(w_i\) (morphology, spelling, synonymy)
  - long queries (\(n\) large)
  - boolean expression generates very few hits

- Simple Boolean vs. Ranking of result set
  - Simple Boolean retrieval returns documents in no particular order.
  - Google (and most well designed Boolean engines) rank the result set – they rank good hits higher than bad hits (according to some estimator of relevance).
Inverted index
Unstructured data in 1650: Plays of William Shakespeare
Unstructured data in 1650

- Which plays of Shakespeare contain the words BRUTUS AND CAESAR, but not CALPURNIA?

- One could grep all of Shakespeare’s plays for BRUTUS and CAESAR, then strip out lines containing CALPURNIA.

- Why is grep not the solution?
  - Slow (for large collections)
  - grep is line-oriented, IR is document-oriented
  - “not CALPURNIA” is non-trivial
  - Other operations (e.g. search for ROMANS near COUNTRY) infeasible
# Term-document incidence matrix

<table>
<thead>
<tr>
<th></th>
<th>Anthony and Caesar</th>
<th>The Tempest</th>
<th>Hamlet</th>
<th>Othello</th>
<th>Macbeth</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ANTHONY</strong></td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td><strong>BRUTUS</strong></td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td><strong>CAESAR</strong></td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td><strong>CALPURNIA</strong></td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td><strong>CLEOPATRA</strong></td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td><strong>MERCY</strong></td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td><strong>WORSE</strong></td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Entry is 1 if term occurs. Example: **CALPURNIA** occurs in *Julius Caesar*.
Entry is 0 if term doesn’t occur. Example: **CALPURNIA** doesn’t occur in *The tempest*. 

---

**82x200**

**Introduction**

**Boolean retrieval**

**Inverted index**

**Boolean queries**

**Text processing**

**Phrase queries**

**Proximity search**
Incidence vectors

- So we have a 0/1 vector for each term.

- To answer the query **Brutus and Caesar and not Calpurnia:**
  1. Take the vectors for Brutus, Caesar, and Calpurnia
  2. Complement the vector of Calpurnia
  3. Do a (bitwise) AND on the three vectors:
     
     $110100 \text{ AND } 110111 \text{ AND } 101111 = 100100$
### 0/1 vector for Brutus

<table>
<thead>
<tr>
<th>Anthony and Cleopatra</th>
<th>Anthony</th>
<th>Julius Caesar</th>
<th>The Tempest</th>
<th>Hamlet</th>
<th>Othello</th>
<th>Macbeth</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>ANTHONY</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>BRUTUS</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>CAESAR</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>CALPURNIA</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>CLEOPATRA</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>MERCY</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>WORSER</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td><strong>result:</strong></td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Anthony and Cleopatra, Act III, Scene ii:

Agrippa [Aside to Domitius Enobarbus]: Why, Enobarbus, When Antony found Julius Caesar dead, He cried almost to roaring; and he wept When at Philippi he found Brutus slain.

Hamlet, Act III, Scene ii:

Lord Polonius: I did enact Julius Caesar: I was killed i’ the Capitol; Brutus killed me.
Consider $N = 10^6$ documents, each with about 1000 tokens
⇒ total of $10^9$ tokens

On average 6 bytes per token, including spaces and punctuation
⇒ size of document collection is about $6 \cdot 10^9 = 6$ GB

Assume there are $M = 500,000$ distinct terms in the collection
⇒ $M = 500,000 \times 10^6 = \text{half a trillion} \, 0s \text{ and } 1s.$

But the matrix has no more than one billion 1s.
⇒ Matrix is extremely sparse.

What is a better representations?
⇒ We only record the 1s.
For each term $t$, we store a list of all documents that contain $t$.

<table>
<thead>
<tr>
<th>Term</th>
<th>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>

**Inverted Index**

- **Dictionary**
- **Postings**
Inverted index construction

1. Collect the documents to be indexed:

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

2. Tokenize the text, turning each document into a list of tokens:

   Friends Romans countrymen So ...

3. Do linguistic preprocessing, producing a list of normalized tokens, which are the indexing terms:

   friend roman countryman so ...

4. Index the documents that each term occurs in by creating an inverted index, consisting of a dictionary and postings.
Tokenization and preprocessing

Doc 1. I did enact Julius Caesar: I was killed i’ the Capitol; Brutus killed me.

Doc 2. So let it be with Caesar. The noble Brutus hath told you Caesar was ambitious:

Doc 1. i did enact julius caesar i was killed i’ the capitol brutus killed me

Doc 2. so let it be with caesar the noble brutus hath told you caesar was ambitious
### Boolean retrieval

#### Boolean queries

<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>caesar</td>
<td>2</td>
</tr>
<tr>
<td>did</td>
<td>1</td>
</tr>
<tr>
<td>enact</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>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>1</td>
</tr>
<tr>
<td>me</td>
<td>1</td>
</tr>
<tr>
<td>me</td>
<td>1</td>
</tr>
<tr>
<td>noble</td>
<td>1</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>1</td>
</tr>
<tr>
<td>told</td>
<td>2</td>
</tr>
<tr>
<td>you</td>
<td>1</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>1</td>
</tr>
<tr>
<td>with</td>
<td>2</td>
</tr>
</tbody>
</table>

### Inverted index

#### Text processing

**Doc 1.** I did enact julius caesar i was killed i’ the capitol brutus killed me

**Doc 2.** so let it be with caesar the noble brutus hath told you caesar was ambitious
Split the result into dictionary and postings file

<table>
<thead>
<tr>
<th>Word</th>
<th>Dictionary (posting)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Brutus</td>
<td>1 2 4 11 31 45 173 174</td>
</tr>
</tbody>
</table>
| Caesar | 1 2 4 5 6 16 57 132 ...
| Calpurnia | 2 31 54 101         |
Boolean queries
Consider the query: **BRUTUS AND CALPURNIA**

To find all matching documents using inverted index:

1. Locate **BRUTUS** in the dictionary
2. Retrieve its postings list from the postings file
3. Locate **CALPURNIA** in the dictionary
4. Retrieve its postings list from the postings file
5. Intersect the two postings lists
6. Return intersection to user
Intersecting two postings lists

\[
\text{Brutus} \quad \longrightarrow \quad 1 \rightarrow 2 \rightarrow 4 \rightarrow 11 \rightarrow 31 \rightarrow 45 \rightarrow 173 \rightarrow 174
\]

\[
\text{Calpurnia} \quad \longrightarrow \quad 2 \rightarrow 31 \rightarrow 54 \rightarrow 101
\]

\[
\text{Intersection} \quad \Longrightarrow \quad 2 \rightarrow 31
\]

- This is linear in the length of the postings lists.
- Note: This only works if postings lists are sorted.
Intersecting two postings lists

\[
\text{\textsc{Intersect}}(p_1, p_2)
\]

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

4. \textbf{then} \(\text{ADD}(\text{answer}, \text{docID}(p_1))\)
5. \(p_1 \leftarrow \text{next}(p_1)\)
6. \(p_2 \leftarrow \text{next}(p_2)\)
7. \textbf{else if} \(\text{docID}(p_1) < \text{docID}(p_2)\)

8. \textbf{then} \(p_1 \leftarrow \text{next}(p_1)\)
9. \textbf{else} \(p_2 \leftarrow \text{next}(p_2)\)
10. \textbf{return} \(\text{answer}\)
Boolean queries

- Boolean model can answer any query that is a Boolean expression.
  - Boolean queries use AND, OR and NOT to join query terms.
  - Views each document as a set of terms.
  - Is precise: Document matches condition or not.

- Primary commercial retrieval tool for 3 decades

- Many professional searchers (e.g., lawyers) still like Boolean queries.
  - You know exactly what you are getting.
Text processing
So far: Simple Boolean retrieval system

Our assumptions were:

1. We know what a document is.
2. We can “machine-read” each document.

This can be complex in reality.
Parsing a document

- We 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 (see later)
- Alternative: use heuristics
A single index usually contains terms of several languages.

- Sometimes a document or its components contain multiple languages/formats (e.g., 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)?

Upshot: Answering the question “what is a document?” is not trivial and requires some design decisions.
Definitions

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

- **Term** – A “normalized” word (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.
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* → *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 and language detection interact.

Example:

- PETER WILL NICHT MIT. → MIT = mit
- He got his PhD from MIT. → MIT ≠ mit
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?
Exercises

- How many word tokens? How many word types?

  **Example 1:** *In June, the dog likes to chase the cat in the barn.*

  **Example 2:** *Mr. O’Neill thinks that the boys’ stories about Chile’s capital aren’t amusing.*

- ...tokenization is difficult – even in English.
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.
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, German, Swedish
- Computerlinguistik → Computer + Linguistik
- Lebensversicherungsgesellschaftsangestellter
  → leben + versicherung + gesellschaft + angestellter
- Inuit: tusaatsiarunnanngittualuujunga (I can’t hear very well.)
- Other languages with segmentation difficulties: Finnish, Urdu ...
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
  - Latin

- No spaces (as in Chinese).
- End user can express query entirely in hiragana!
Arabic script

کتاب

un bātīk

/kitābun/ ‘a book’
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 “ä” and “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 (e.g. Czech)
Case folding

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

**Example:** 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

- 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: phonetic equivalence, e.g. Muller = Mueller
- Thesauri: semantic equivalence, e.g. car = automobile
Lemmatization

- Reduce inflectional/variant forms to base form

- Examples:
  - *am, are, is* → *be*
  - *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*).

- Two types:
  - inflectional (*cutting* → *cut*)
  - derivational (*destruction* → *destroy*)
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** (derivational): *automate, automatic, automation* all reduce to *automat*
Porter algorithm (1980)

- Most common algorithm for stemming English
- Results suggest that it is at least as good as other stemming options (1980!)
- Conventions + 5 phases of reductions 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</td>
<td>caresses → caress</td>
</tr>
<tr>
<td>IES</td>
<td>ponies → poni</td>
</tr>
<tr>
<td>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.

- Queries where stemming is likely to help:
  - \([\text{TARTAN SWEATERS}], [\text{SIGHTSEEING TOUR SAN FRANCISCO}]\)
  - equivalence classes: \(\{\text{sweater}, \text{sweaters}\}, \{\text{tour}, \text{tours}\}\)

- Queries where stemming hurts:
  - \([\text{OPERATIONAL RESEARCH}], [\text{OPERATING SYSTEM}], [\text{OPERATIVE DENTISTRY}]\)
  - Porter Stemmer equivalence class \(\text{oper}\) contains all of \(\text{operate}, \text{operating}, \text{operates}, \text{operation}, \text{operative}, \text{operatives}, \text{operational}\).
Phrase queries
Phrase queries

- We answer a query such as \texttt{[STANFORD UNIVERSITY]} – as a phrase.

- Thus \textit{The inventor Stanford Ovshinsky never went to university} should \textbf{not} 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: it no longer suffices to store docIDs in postings lists.

- Two ways of extending the inverted index:
  1. biword index
  2. positional index
Biword indexes

▶ Index every consecutive pair of terms in the text as a phrase.

▶ Example: *Friends, Romans, Countrymen* 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

- 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
Positional indexes: Example

Query: “to\textsubscript{1} be\textsubscript{2} or\textsubscript{3} not\textsubscript{4} to\textsubscript{5} be\textsubscript{6}”

TO, 993427:

\[
\begin{align*}
1: & \langle 7, 18, 33, 72, 86, 231 \rangle; \\
2: & \langle 1, 17, 74, 222, 255 \rangle; \\
4: & \langle 8, 16, 190, 429, 433 \rangle; \\
5: & \langle 363, 367 \rangle; \\
7: & \langle 13, 23, 191 \rangle; \ldots
\end{align*}
\]

BE, 178239:

\[
\begin{align*}
1: & \langle 17, 25 \rangle; \\
4: & \langle 17, 191, 291, 430, 434 \rangle; \\
5: & \langle 14, 19, 101 \rangle; \ldots
\end{align*}
\]

Document 4 is a match!
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 /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 all combinations 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**($p_1, p_2, k$)

```latex
answer \leftarrow \langle \rangle \\
while p_1 \neq \text{NIL} \text{ and } p_2 \neq \text{NIL} \\
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 |pos(pp_1) - pos(pp_2)| \leq k \\
then \text{ADD}(l, pos(pp_2)) \\
else if pos(pp_2) > pos(pp_1) \\
then break \\
pp_2 \leftarrow \text{next}(pp_2) \\
while l \neq \langle \rangle \text{ and } |l[0] - pos(pp_1)| > k \\
do \text{DELETE}(l[0]) \\
for each ps \in l \\
do \text{ADD}(answer, \langle \text{docID}(p_1), pos(pp_1), ps \rangle) \\
pp_1 \leftarrow \text{next}(pp_1) \\
p_1 \leftarrow \text{next}(p_1) \\
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
```
Combination scheme

- Biword indexes and positional indexes can be profitably combined.

- Many biwords extremely frequent: *Michael Jackson, Lady Gaga* 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.

- 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?