Indexing, cont’d

PHRASE QUERIES AND POSITIONAL INDEXES
Phrase Queries

• Want to be able to answer queries such as “dalhousie university” – as a phrase

• Thus the sentence “I went to university at dalhousie” is not a match.
  – The concept of phrase queries has proven easily understood by users; one of the few “advanced search” ideas that works
  – Many more queries are implicit phrase queries

• For this, it no longer suffices to store only $\langle\text{term} : \text{docs}\rangle$ entries
A First Attempt: Biword indexes

• Index every consecutive pair of terms in the text as a phrase
• For example the text “Friends, Romans, Countrymen” would generate the biwords
  – *friends romans*
  – *romans countrymen*
• Each of these biwords is now a dictionary term
• Two-word phrase query-processing is now immediate.
One Motive

• Could be that most search queries are 2.4 words long.
• This applies to Web search only.
Longer Phrase Queries

- Longer phrases are processed as Boolean biword queries:

- *dalhousie university halifax canada* can be broken into the following Boolean query on biwords:

  *dalhousie university AND university halifax AND halifax canada*

- Without the documents, we cannot verify that the documents matching the above Boolean query do contain the phrase.

  Can have false positives!
Extended biwords

• Parse the indexed text and perform part-of-speech-tagging (POST).
• Bucket the terms into (say) Nouns (N) and articles/prepositions (X).
• Call any string of terms of the form NX*N an extended biword.
  – Each such extended biword is now made a term in the dictionary.
• Example: catcher in the rye
  
  N X X N

• Query processing: parse it into N’s and X’s
  – Segment query into enhanced biwords
  – Look up in index: catcher rye
Issues for biword indexes

• False positives, as noted before
• Index blowup due to bigger dictionary
  – Infeasible for more than biwords, big even for them

• Biword indexes are not the standard solution (for all biwords) but can be part of a compound strategy
Solution 2: Positional Indexes

- In the postings, store, for each term the position(s) in which tokens of it appear:

  <term, number of docs containing term;
doc1: position1, position2 ... ;
doc2: position1, position2 ... ;
  etc.>
Positional index example

For phrase queries, we use a merge algorithm recursively at the document level.

But we now need to deal with more than just equality.

Which of docs 1, 2, 4, 5 could contain “to be or not to be”? 

```
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⟩; …
```
Processing a phrase query

• Extract inverted index entries for each distinct term: \textit{to, be, or, not}.
• Merge their \textit{doc:position} lists to enumerate all positions with \textit{“to be or not to be”}.
  – \textit{to}:
    • 2:1,17,74,222,551; 4:8,\textbf{16,190,429,433}; 7:13,23,191; ...
  – \textit{be}:
    • 1:17,19; \textbf{4:17,191,291,430,434}; 5:14,19,101; ...
• Same general method for proximity searches

Sec. 2.4.2
Proximity queries

- **LIMIT! /3 STATUTE /3 FEDERAL /2 TORT**
  - Again, here, /k means “within k words of”.

- Clearly, positional indexes can be used for such queries; biword indexes cannot.

- **Exercise:** Adapt the linear merge of postings to handle proximity queries. Can you make it work for any value of k?
  - This is a little tricky to do correctly and efficiently
  - See Figure 2.12 of IIR
  - There’s likely to be a problem on it!
Figure 2.12
Text Book

Please run it through this example:

to:

4: 8, 16, 190, 429, 433;

be:

4: 17, 191, 291, 430, 434;

Proximity on both sides of the term

```
FUNCTION positionalIntersect(p1, p2, k)
  answer <- NIL
  while p1 != NIL and p2 != NIL
    do if docID(p1) = docID(p2)
      then l <- NIL
        pp1 <- positions(p1)
        pp2 <- positions(p2)
        while pp1 != NIL
          do while pp2 != NIL
            do if |pos(pp1) - pos(pp2)| <= k
              then ADD(l, pos(pp2))
              else if pos(pp2) > pos(pp1)
                then break
                  pp2 <- next(pp2)
            while l != NIL and |l[0] - pos(pp1)| > k
              do DELETE(l[0])
                for each ps in l
                  do ADD(answer, ⟨docID(p1), pos(pp1), ps⟩)
                    pp1 <- next(pp1)
                    p1 <- next(p1)
                    p2 <- next(p2)
            else if docID(p1) < docID(p2)
              then p1 <- next(p1)
              else p2 <- next(p2)
      return answer
```
Positional index size

• You can compress position values/offsets.
• Nevertheless, a positional index expands postings storage **substantially**
• Nevertheless, a positional index is now standardly used because of the power and usefulness of phrase and proximity queries ... whether used explicitly or implicitly in a ranking retrieval system.
Positional index size

• Need an entry for each occurrence, not just once per document

• Index size depends on average document size
  – Average web page has <1000 terms
  – Books and some epic poems ... easily 100,000 terms

• Consider a term with frequency 0.1%

<table>
<thead>
<tr>
<th>Document size</th>
<th>Postings</th>
<th>Positional postings</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>100,000</td>
<td>1</td>
<td>100</td>
</tr>
</tbody>
</table>
Rules of thumb

• A positional index is 2–4 as large as a non-positional index
• Positional index size 35–50% of volume of original text
• Caveat: all of this holds for “English-like” languages
Combination schemes

• These two approaches can be profitably combined
  – For particular phrases ("Michael Jackson", "Britney Spears") it is inefficient to keep on merging positional postings lists
    • Even more so for phrases like "The Who"

• Williams et al. (2004) evaluate a more sophisticated mixed indexing scheme
  – A typical web query mixture was executed in ¼ of the time of using just a positional index
  – It required 26% more space than having a positional index alone
Resources for today’s lecture

