CMPE 493
INTRODUCTION TO INFORMATION RETRIEVAL

Phrase Queries and Positional Indexes

Department of Computer Engineering, Boğaziçi University
October 27, 2020

Phrase queries

- Want to be able to answer queries such as "***Boğaziçi University***" – as a phrase
- Thus the sentence *"Sabancı University is at the other side of Boğaziçi Bridge."* is not a match.
- For this, it no longer suffices to store only <*term* : *docs*> 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.

Longer phrase queries

How would answer the phrase query

boğaziçi university students

boğaziçi university students can be broken into the Boolean query on biwords:

boğaziçi university *AND* ***university students***

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

Can have false positives!

Extended biwords

- Parse the indexed text and perform part-of-speech (POS) tagging.
- Bucket the terms into 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, for each ***term*** store the position(s) of its tokens in the documents:

<***term***, number of docs containing ***term***;
doc1: position1, position2 … ;
doc2: position1, position2 … ;
etc.>

Positional index example

<***be***: 993427;
1: 7, 18, 33, 72, 86, 231;
2: 3, 149;
4: 17, 191, 291, 430, 434;
5: 363, 367, …>

Which of docs 1,2,4,5 could contain "***to be or not to be***"?

- For phrase queries, we use a merge algorithm recursively at the document level
- But we now need to deal with more than just equality

Processing a phrase query

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

Proximity queries

- employment /3 place
  - Again, here, /*k* means "within *k* words of".
- Clearly, positional indexes can be used for such queries; biword indexes cannot.

Positional index size

- Rules of thumb for "English-like" languages:
  - A positional index is 2–4 as large as a non-positional index
  - Positional index size 35–50% of volume of original text

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.

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"***

References

- *Introduction to Information Retrieval*, chapter 2
  - http://nlp.stanford.edu/IR-book/information-retrieval-book.html