



Lect 8: Rhrase Query

Dr. Subrat Kumar Nayak

Associate Professor

Dept. of CSE, ITER, SOADU

Phase Query

- We want to be able to answer queries such as “**stanford university**” – as a phrase
- Thus the sentence “*I went to university at Stanford*” 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*
- About 10% of web queries are phrase queries.
- Consequence for inverted index: it no longer suffices to store docIDs in postings lists for terms.

<term : docs> entries

- Two ways of extending the inverted index:
 - biword index
 - positional index

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

- Longer phrases can be processed by breaking them down
- ***stanford university palo alto*** can be broken into the Boolean query on biwords:

stanford university AND university palo AND palo alto

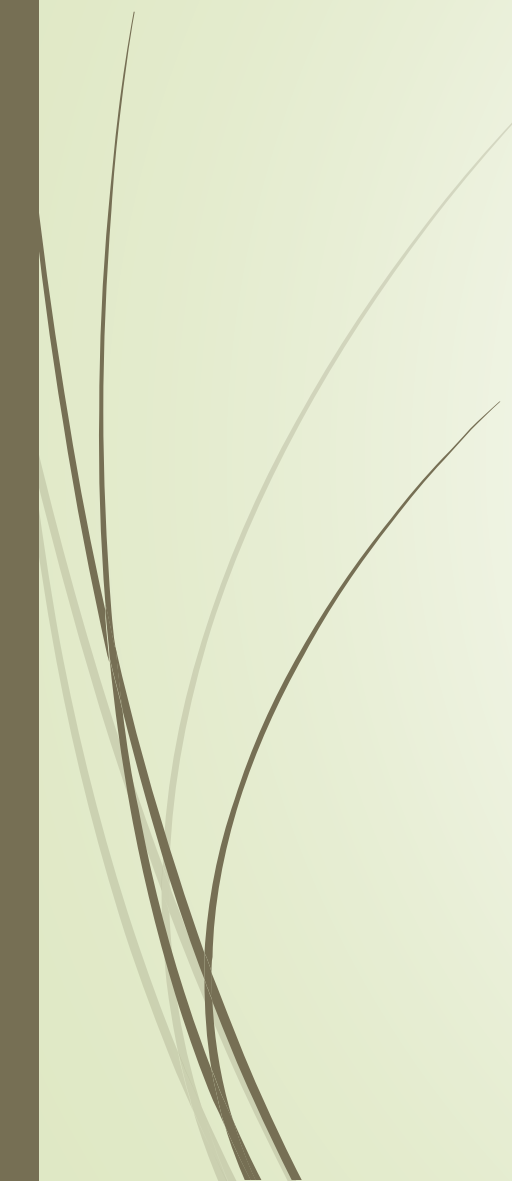
- We need to do post-filtering of hits to identify subset that actually contains the **4-word phrase**.

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

Can have false positives!



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

- Positional indexes are a more efficient alternative to byword 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**
- 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 indexes: Example

Query: "*to*₁ *be*₂ *or*₃ *not*₄ *to*₅ *be*₆"

TO, 993427:

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

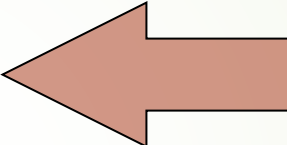
BE, 178239:

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

Document 4 is a match!

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

