

---

BLG 540E  
TEXT RETRIEVAL SYSTEMS

The Term Vocabulary and  
Postings Lists

Arzucan Özgür

# Previous lecture

---

- ▶ **Basic inverted indexes:**
  - ▶ Structure: Dictionary and Postings

BRUTUS → 

1	2	4	11	31	45	173	174
---	---	---	----	----	----	-----	-----

CAESAR → 

1	2	4	5	6	16	57	132	...
---	---	---	---	---	----	----	-----	-----

CALPURNIA → 

2	31	54	101
---	----	----	-----

- ▶ Key step in construction: Sorting
- ▶ **Boolean query processing**
  - ▶ Intersection by linear time “merging”
  - ▶ Simple optimizations
- ▶ **Overview of course topics**

# Does Google use the Boolean model?

---

- ▶ On Google, the default interpretation of a query  $[w_1 w_2 \dots w_n]$  is  $w_1 \text{ AND } w_2 \text{ AND } \dots \text{ AND } w_n$
- ▶ Cases where you get hits that do not contain one of the  $w_i$  :
  - ▶ anchor text
  - ▶ page contains variant of  $w_i$  (morphology, spelling correction, synonym)
  - ▶ long queries ( $n$  large)
  - ▶ boolean expression generates very few hits
- ▶ Simple Boolean vs. Ranking of result set
  - ▶ Simple Boolean retrieval returns matching documents in no particular order.
  - ▶ Google (and most well designed Boolean engines) rank the result set – they rank good hits (according to some estimator of relevance) higher than bad hits.



# Plan for this lecture

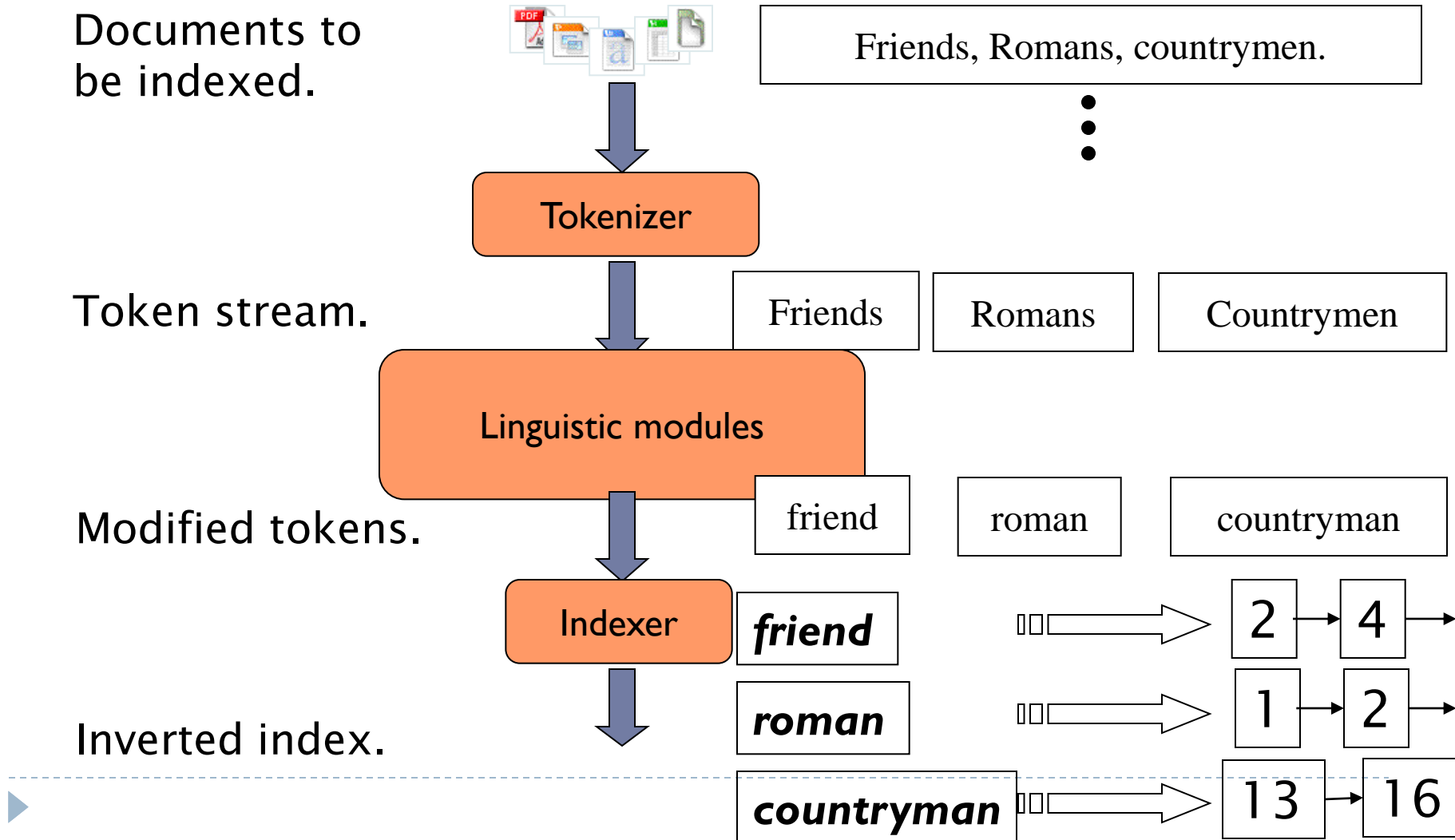
---

## Elaborate basic indexing

- ▶ Preprocessing to form the term vocabulary
  - ▶ Documents
  - ▶ Tokenization
  - ▶ What *terms* do we put in the index?
- ▶ Postings
  - ▶ Faster merges: skip lists
  - ▶ Positional postings and phrase queries



# Recall the basic indexing pipeline



# Parsing a document

---

- ▶ What format is it in?
  - ▶ pdf/word/excel/html?
- ▶ What language is it in?
- ▶ What character set is in use?
  - ▶ ASCII, Unicode UTF-8, national or vendor specific

Each of these is a classification problem, which we will study later in the course.

But these tasks are often done heuristically ...



# Complications: Format/language

---

- ▶ Documents being indexed can include docs from many different languages
  - ▶ A single index may have to contain terms of several languages.
- ▶ Sometimes a document or its components can contain multiple languages/formats
  - ▶ Turkish email with an English pdf attachment.
- ▶ What is a unit document?
  - ▶ A file?
  - ▶ An email? (Perhaps one of many in an mbox.)
  - ▶ An email with 5 attachments?
  - ▶ A group of files (PPT or LaTeX as HTML pages)



---

# Tokens and Terms



---





# Tokenization

---

- ▶ Input: “***Friends, Romans and Countrymen***”
- ▶ Output: Tokens
  - ▶ ***Friends***
  - ▶ ***Romans***
  - ▶ ***Countrymen***
- ▶ A **token** is an instance of a sequence of characters
- ▶ Each such token is now a candidate for an index entry, after further processing
  - ▶ Described below
- ▶ But what are valid tokens to emit?



# Tokenization

---

- ▶ Issues in tokenization:
  - ▶ *Turkey's capital* →  
*Turkey? Turkeys? Turkey's?*
  - ▶ *Hewlett-Packard* → *Hewlett* and *Packard* as two tokens?
    - ▶ **state-of-the-art**: break up hyphenated sequence.
    - ▶ **co-education**
    - ▶ **lowercase, lower-case, lower case ?**
  - ▶ *Istanbul Technical University*: one token or two tokens, or three tokens?
    - ▶ How do you decide it is one token?



# Numbers

---

- ▶ **2/18/2011** **Feb. 18, 2011**  
**18/2/2011**
  - ▶ **55 B.C.**
  - ▶ **(800) 234-2333**
    - ▶ Often have embedded spaces
    - ▶ Older IR systems may not index numbers
      - ▶ But often very useful: think about things like looking up error codes/stacktraces on the web
      - ▶ (One answer is using n-grams: will be discussed later in the course)
    - ▶ Will often index “meta-data” separately
      - ▶ Creation date, format, etc.
- 



# Tokenization: language issues

---

## ▶ French

- ▶ **L'ensemble** → one token or two?
  - ▶ **L ? L' ? Le ?**
  - ▶ Want **l'ensemble** to match with **un ensemble**
    - Until at least 2003, it didn't on Google

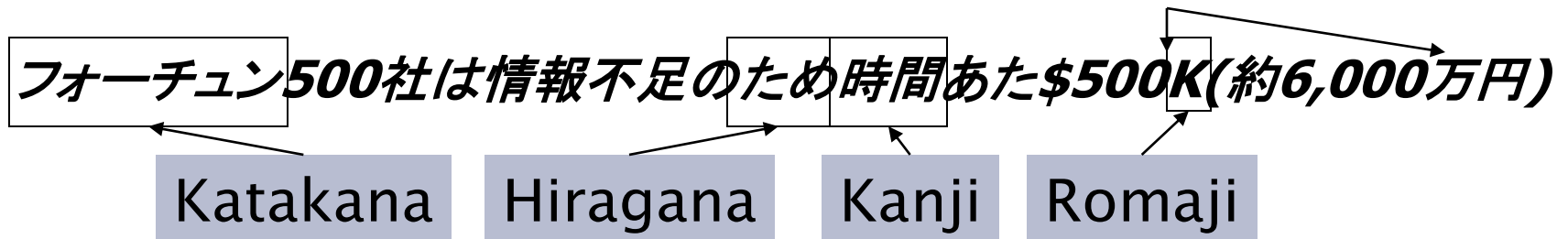
## ▶ German noun compounds are not segmented

- ▶ **Lebensversicherungsgesellschaftsangestellter**
- ▶ 'life insurance company employee'
- ▶ German retrieval systems benefit greatly from a **compound splitter** module
  - Can give a 15% performance boost for German



# Tokenization: language issues

- ▶ Chinese and Japanese have no spaces between words:
  - ▶ 莎拉波娃现在居住在美国东南部的佛罗里达。
  - ▶ Not always guaranteed a unique tokenization
- ▶ Further complicated in Japanese, with multiple alphabets intermingled
  - ▶ Dates/amounts in multiple formats



End-user can express query entirely in hiragana!

# Tokenization: language issues

---

- ▶ Arabic (or Hebrew) is basically written right to left, but with certain items like numbers written left to right
- ▶ Words are separated, but letter forms within a word form complex ligatures

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

← start

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

# Stop words

---

- ▶ With a stop list, you exclude from the dictionary entirely the commonest words. Intuition:
  - ▶ They have little semantic content: *the, a, and, to, be*
  - ▶ There are a lot of them.
- ▶ But the trend is away from doing this:
  - ▶ Good compression techniques (will be discussed later in the course) means the space for including stopwords in a system is very small
  - ▶ Good query optimization techniques mean you pay little at query time for including stop words.
  - ▶ You need them for:
    - ▶ Phrase queries: “King of Denmark”
    - ▶ Various song titles, etc.: “Let it be”, well known verse “To be or not to be”
    - ▶ “Relational” queries: “flights to Istanbul”



# Normalization to terms

---

- ▶ We need to “normalize” words in indexed text as well as query words into the same form
  - ▶ We want to match **U.S.A.** and **USA**
- ▶ Result is terms: a **term** is a (normalized) word type, which is an entry in our IR system dictionary
- ▶ We most commonly implicitly define equivalence classes of terms by, e.g.,
  - ▶ deleting periods to form a term
    - ▶ **U.S.A., USA** ( **USA**
  - ▶ deleting hyphens to form a term
    - ▶ **anti-discriminatory, antidiscriminatory** ( **antidiscriminatory**





# Normalization: other languages

---

- ▶ Accents: e.g., French ***résumé*** vs. ***resume***.
- ▶ Umlauts: e.g., German: ***Tuebingen*** vs. ***Tübingen***
  - ▶ Should be equivalent
- ▶ Most important criterion:
  - ▶ How are your users like to write their queries for these words?
- ▶ Even in languages that standardly have accents, users often may not type them
  - ▶ Often best to normalize to a de-accented term
    - ▶ ***Tuebingen, Tübingen, Tubingen*** \ ***Tubingen***



# Normalization: other languages

---

- ▶ Normalization of things like date forms
  - ▶ **7月30日 vs. 7/30**
- ▶ Tokenization and normalization may depend on the language and so is intertwined with language detection
- ▶ Crucial: Need to “normalize” indexed text as well as query terms into the same form

*Morgen will ich in MIT ...*

Is this  
German “mit”?

# Case folding

---

- ▶ Reduce all letters to lower case
  - ▶ exception: upper case in mid-sentence?
    - ▶ e.g., **General Motors**
    - ▶ **Fed** vs. *fed*
  - ▶ Often best to lower case everything, since users will use lowercase regardless of ‘correct’ capitalization...
- ▶ Google example:
  - ▶ Query **C.A.T.**
  - ▶ #1 result used to be for “cat” (animal) *not* Caterpillar Inc. → Now it is the company (even when we query for cat)



# Normalization to terms

---

- ▶ An alternative to equivalence classing is to do asymmetric expansion
- ▶ An example of where this may be useful
  - ▶ Enter: **window**      Search: **window, windows**
  - ▶ Enter: **windows**      Search: **Windows, windows, window**
  - ▶ Enter: **Windows**      Search: **Windows**
- ▶ Potentially more powerful, but less efficient



# Thesauri and soundex

---

- ▶ Do we handle synonyms and homonyms?
  - ▶ E.g., by hand-constructed equivalence classes
    - ▶ **car** = **automobile** **color** = **colour**
  - ▶ We can rewrite to form equivalence-class terms
    - ▶ When the document contains **automobile**, index it under **car-automobile** (and vice-versa)
  - ▶ Or we can expand a query
    - ▶ When the query contains **automobile**, look under **car** as well
- ▶ What about spelling mistakes?
  - ▶ One approach is soundex, which forms equivalence classes of words based on phonetic heuristics



# Lemmatization

---

- ▶ Reduce inflectional/variant forms to base form
- ▶ E.g.,
  - ▶ *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

# Stemming

---

- ▶ Reduce terms to their “roots” before indexing
- ▶ “Stemming” suggest crude affix chopping
  - ▶ language dependent
  - ▶ e.g., ***automate(s), automatic, automation*** all reduced to ***automat***.

*for example compressed and compression are both accepted as equivalent to compress.*



for exampl compress and  
compress ar both accept  
as equival to compress

# Porter's algorithm

---

- ▶ **Commonest algorithm for stemming English**
  - ▶ Results suggest it's at least as good as other stemming options
- ▶ **Conventions + 5 phases of reductions**
  - ▶ phases applied sequentially
  - ▶ each phase consists of a set of commands
  - ▶ sample convention: *Of the rules in a compound command, select the one that applies to the longest suffix.*





# Typical rules in Porter

---

- ▶ *sses* → *ss*    ex: *caresses* → *caress*
- ▶ *ies* → *i*    ex: *ponies* → *poni*
- ▶ *ss* → *ss*    ex: *caress* → *caress*
- ▶ *s* →            ex: *cats* → *cat*
  
- ▶ Weight of word sensitive rules
- ▶     *(m > 1) EMENT* →
  - ▶ *replacement* → *replac*
  - ▶ *cement* → *cement*

# Other stemmers

---

- ▶ **Other stemmers exist, e.g., Lovins stemmer**
  - ▶ <http://www.comp.lancs.ac.uk/computing/research/stemming/general/lovins.htm>
  - ▶ Single-pass, longest suffix removal (about 250 rules)
- ▶ **Full morphological analysis – at most modest benefits for retrieval**
- ▶ **Do stemming and other normalizations help?**
  - ▶ English: very mixed results. Helps recall for some queries but harms precision on others
    - ▶ E.g., operative (dentistry)  $\Rightarrow$  oper
  - ▶ Definitely useful for Spanish, German, Finnish, ...
    - ▶ 30% performance gains for Finnish!



# Language-specificity

---

- ▶ Many of the above features embody transformations that are
  - ▶ Language-specific and
  - ▶ Often, application-specific
- ▶ These are “plug-in” addenda to the indexing process
- ▶ Both open source and commercial plug-ins are available for handling these



# Dictionary entries

---

*ensemble.french*

*時間.japanese*

*MIT.english*

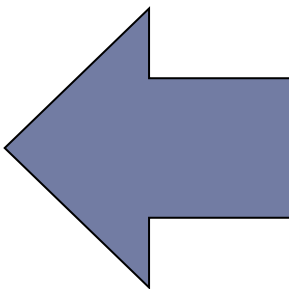
*mit.german*

*guaranteed.english*

*entries.english*

*sometimes.english*

*tokenization.english*



These may be grouped by  
language (or not...).  
More on this in  
ranking/query processing.

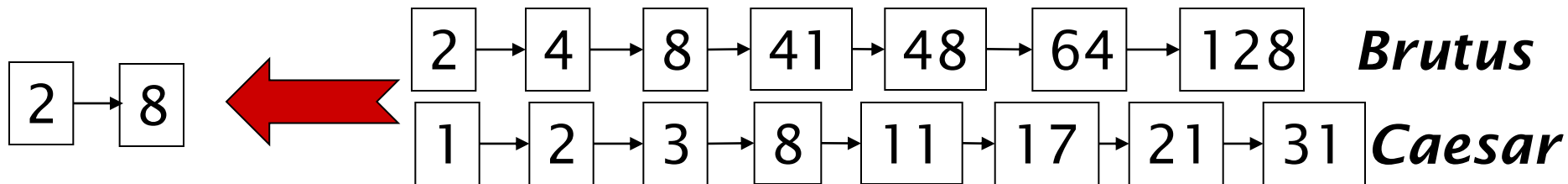
---

Faster postings merges:  
Skip pointers/Skip lists



# Recall basic merge

- ▶ Walk through the two postings simultaneously, in time linear in the total number of postings entries



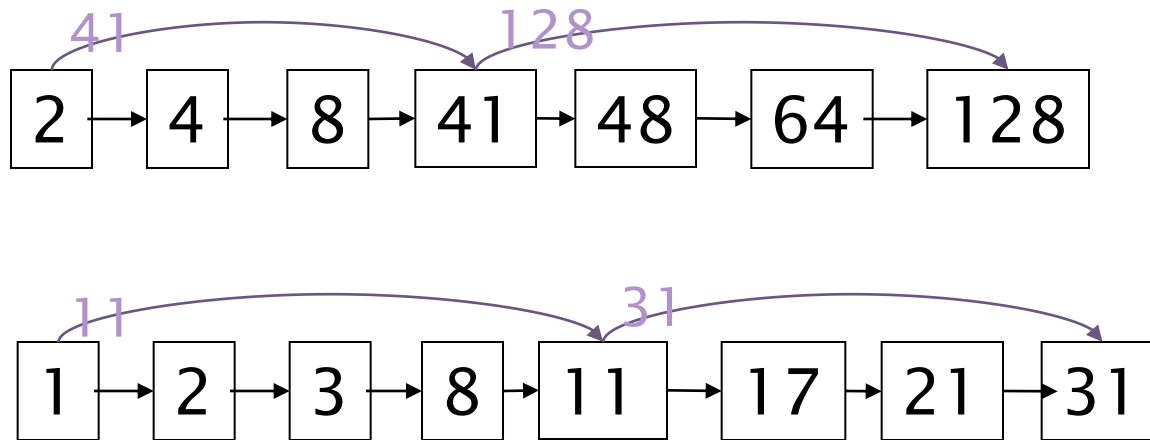
If the list lengths are  $m$  and  $n$ , the merge takes  $O(m+n)$  operations.

Can we do better?

Yes (if index isn't changing too fast).

# Augment postings with skip pointers (at indexing time)

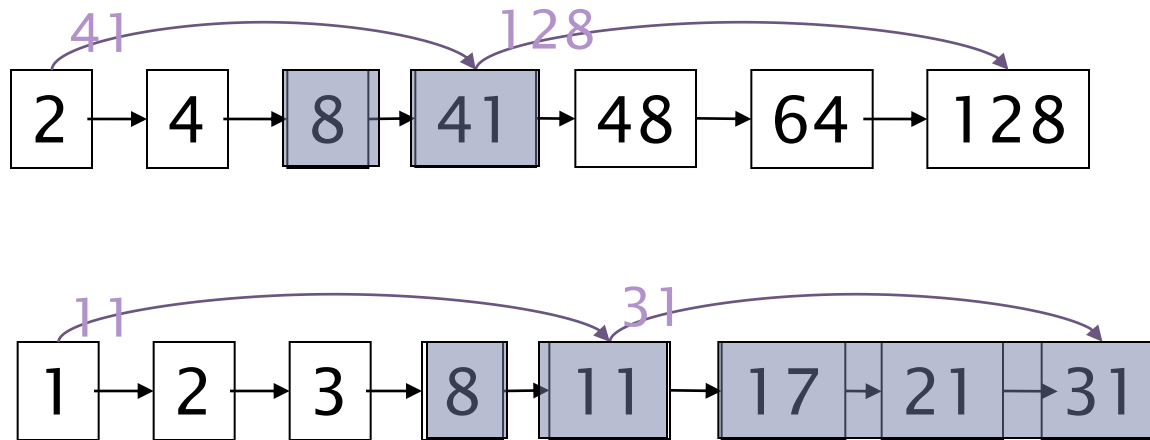
---



- ▶ Why?
  - ▶ To skip postings that will not figure in the search results.
  - ▶ How?
  - ▶ Where do we place skip pointers?
-

# Query processing with skip pointers

---



Suppose we've stepped through the lists until we process **8** on each list. We match it and advance.

We then have **41** and **11** on the lower. **11** is smaller.

But the skip successor of **11** on the lower list is **31**, so we can skip ahead past the intervening postings.

---



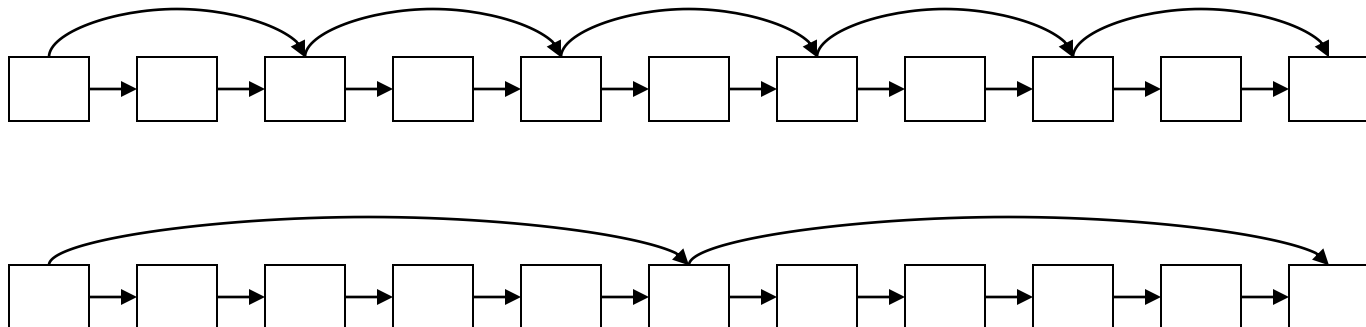


# Where do we place skips?

---

## ► Tradeoff:

- More skips  $\rightarrow$  shorter skip spans  $\Rightarrow$  more likely to skip. But lots of comparisons to skip pointers.
- Fewer skips  $\rightarrow$  few pointer comparison, but then long skip spans  $\Rightarrow$  few successful skips.



# Placing skips

---

- ▶ Simple heuristic: for postings of length  $L$ , use  $\sqrt{L}$  evenly-spaced skip pointers.
- ▶ This ignores the distribution of query terms.
- ▶ Easy if the index is relatively static; harder if  $L$  keeps changing because of updates.
- ▶ This definitely used to help; with modern hardware it may not unless you're memory-based
  - ▶ The I/O cost of loading a bigger postings list can outweigh the gains from quicker in memory merging!



---

## Phrase queries and positional indexes



# Phrase queries

---

- ▶ Want to be able to answer queries such as “***istanbul university***” – as a phrase
- ▶ Thus the sentence “*I went to university at Istanbul*” 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.



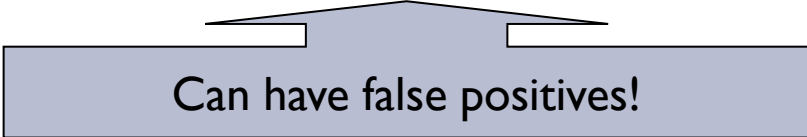
# Longer phrase queries

---

- ▶ ***istanbul university turkey*** can be broken into the Boolean query on biwords:

***istanbul university AND university turkey***

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-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 ... ;

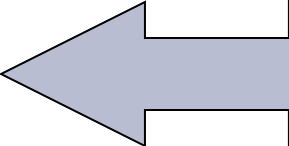
etc.>



# Positional index example

---

<*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

---

- ▶ 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.

# Positional index size

---

- ▶ You can compress position values/offsets: we'll talk about that later.
- ▶ 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, even some epic poems ... easily 100,000 terms
- ▶ Consider a term with frequency 0.1%

Document size	Postings	Positional postings
1000	1	1
100,000	1	100



# 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**”



# References

---

- ▶ *Introduction to Information Retrieval*, chapter 2
- ▶ The slides were adapted from the book's companion website:
  - ▶ <http://nlp.stanford.edu/IR-book/information-retrieval-book.html>
- ▶ Porter's stemmer:  
<http://www.tartarus.org/~martin/PorterStemmer/>

