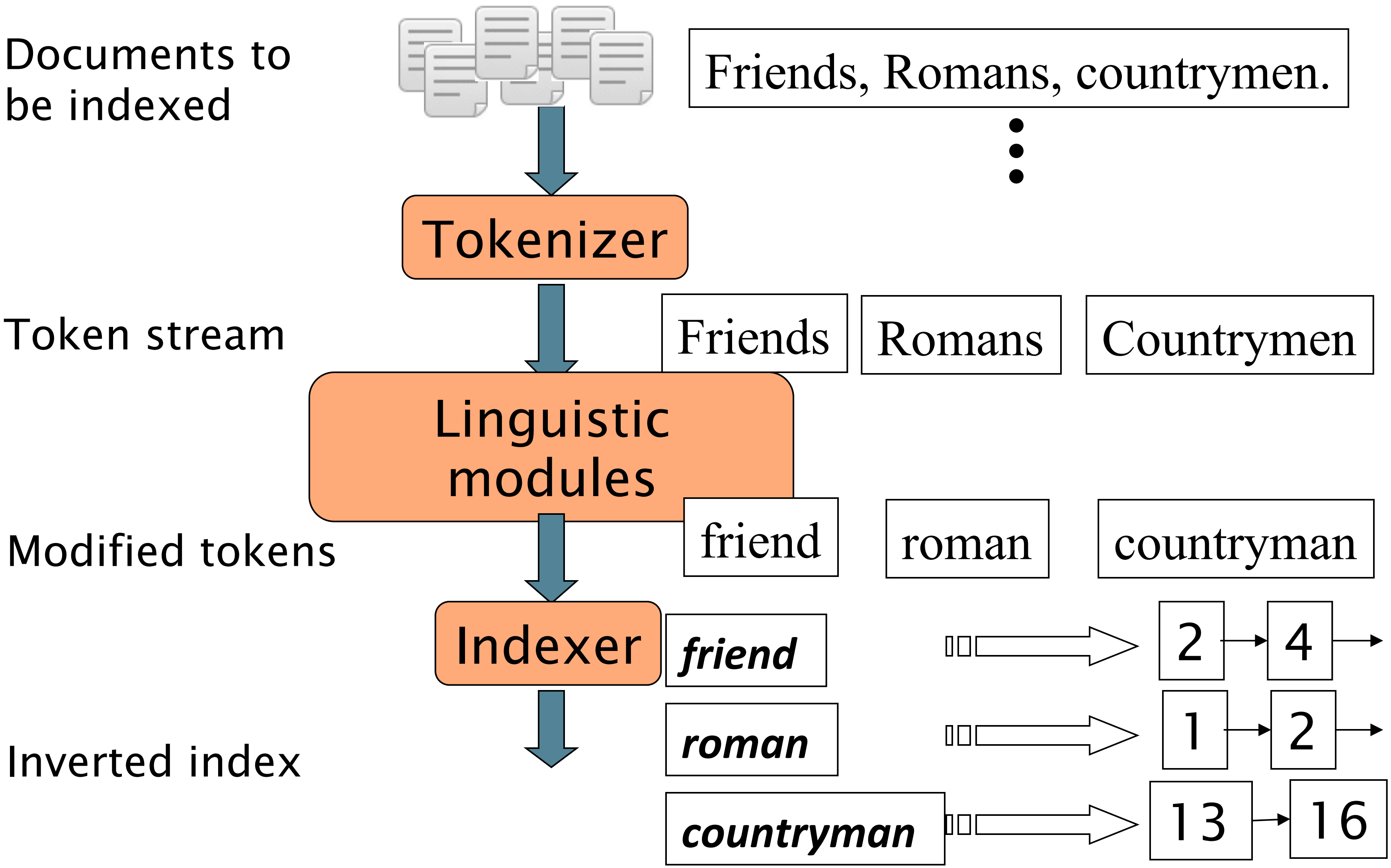
Introduction to

Information Retrieval

Document ingestion

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?
  - CP1252 (1 byte encoding for Latin and other western languages), UTF-8, …

- Each of above is a classification problem (learnt already in NLP or can be seen again).
- But these tasks are often done heuristically (learn from themselves through trial & error or rules…)

Complications: Format/language

- Document or its components can contain multiple languages/formats
  - French email with a German pdf attachment.
  - French email quote clauses from an English-language
  - Urdu text containing English/Arabic words/sentences.
  - A single index may contain terms from many languages.

- There are commercial and open source libraries that can handle a lot of this stuff

Complications: What is a document?

We return from our query "documents" but there are often interesting questions of grain size (granularity):

What is a unit document?

- A file?
- An email? (Perhaps one of many in a single mbox file)
  - What about an email with 5 attachments?
- A group of files (e.g., PPT or LaTeX split over HTML pages)

Introduction to

Information Retrieval

Tokens

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

But what are valid tokens to emit?

Tokenization

- Issues in tokenization:
  - ***Finland's capital*** →
    Finland AND ***s***? ***Finlands***? ***Finland's***?
  - ***Hewlett-Packard*** → ***Hewlett*** and ***Packard*** as two tokens?
    - ***state-of-the-art***: break up hyphenated sequence.
    - ***co-education***
    - ***lowercase***, ***lower-case***, ***lower case*** ?
    - It can be effective to get the user to put in possible hyphens
  - ***San Francisco***: one token or two?
    - How do you decide it is one token?

Numbers

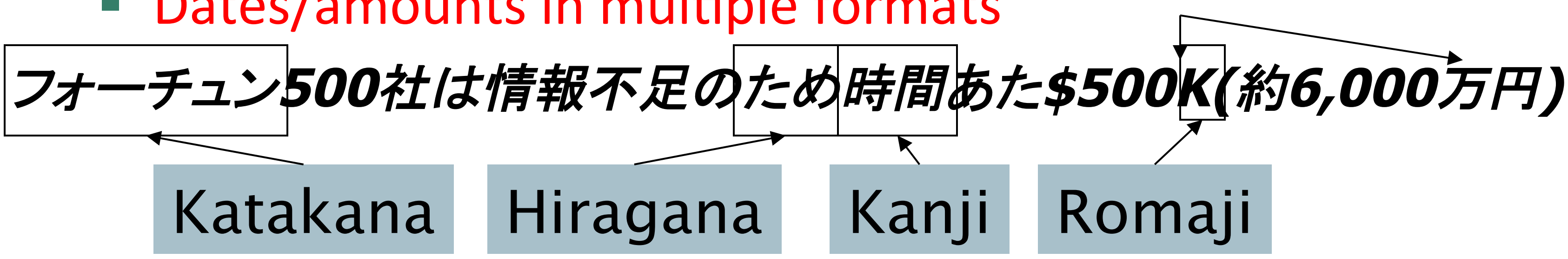
- ***3/20/91*** ***Mar. 20, 1991*** ***20/3/91***
- ***55 B.C.***
- ***B-52***
- ***(800) 234-2333***
  - Often have embedded spaces
  - Older IR systems may not index numbers
    - But often very useful: think about things like looking up error codes/stacktraces on the web

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 Urdu) is basically written from right to left, but with certain items like numbers are written from left to right.
- Words are separated, but letter forms within a word form complex ligatures

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

← → ← → ← START

'Algeria achieved its independence in 1962 after 132 years of French occupation.'

- With Unicode, the surface presentation is complex, but the stored form is straightforward

Introduction to

Information Retrieval

Terms

The things indexed in an IR system

Stop words

- Exclude commonest words from the dictionary:
  - They have little semantic content: *the, a, and, to, be*
  - There are a lot of them: ~30% of postings for top 30 words
- But the trend is away from doing this:
  - Good compression techniques (IIR 5) means the space for including stop words in a system is very small
  - Good query optimization techniques (IIR 7) 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", "To be or not to be"
    - "Relational" queries: "flights to London"

Normalization to terms

- "Normalize" words in indexed as well as query text 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

- Implicitly define equivalence classes of terms 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
- Approach:
  - How are 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

- Tokenization and normalization may depend on the language and so is intertwined with language detection

Morgen will ich in MIT …

Is this German "mit"?

- Crucial: Need to "normalize" indexed text as well as query terms identically

Case folding

- Reduce all letters to lower case
  - exception: upper case in mid-sentence?
    - e.g., General Motors
    - Fed vs. fed
    - SAIL vs. sail
  - Often best to lower case everything, since users will use lowercase regardless of 'correct' capitalization…

- Longstanding Google example: [fixed in 2011…]
  - Query C.A.T.
  - #1 result is for "cats".

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*** *as well* (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
- More in IIR 3 and IIR 9

Introduction to

Information Retrieval

Stemming and Lemmatization

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" suggests crude affix chopping
  - 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*
- *ies → i*
- *ational → ate*
- *tional → tion*
- More than one character before EMENT
  - *(m>1) EMENT →*
    - *replacement → replac*
    - *cement → cement*

https://tartarus.org/martin/PorterStemmer/

Other stemmers

- Other stemmers exist:
  - Lovins stemmer
    - http://www.comp.lancs.ac.uk/computing/research/stemming/general/lovins.htm
    - Single-pass, longest suffix removal (about 250 rules)
  - Paice/Husk stemmer: http://paicehusk.appspot.com/
  - Snowball stemmer: https://snowballstem.org/
- Use Full morphological analysis (lemmatization)
  - At most modest benefits for retrieval as compared to stemmer (IR, page 33)

Language-specificity

- Discussed methods 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

Does stemming help?

- English: very mixed results. Helps recall for some queries but harms precision on others
- Definitely useful for Spanish, German, Finnish, …
  - 30% performance gains for Finnish!

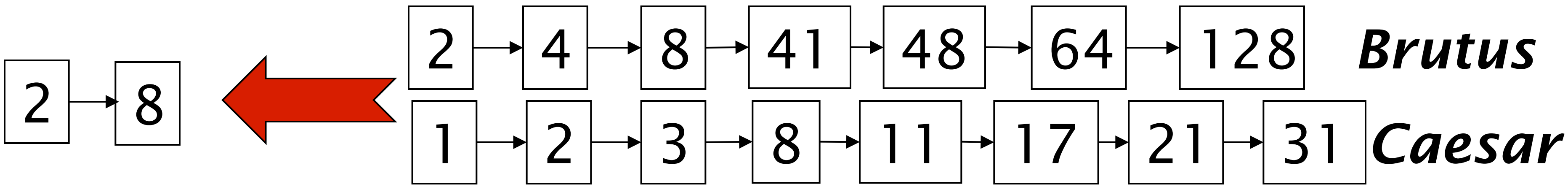
Introduction to

Information Retrieval

Faster postings merges:
Skip pointers/Skip lists

Recall basic merge

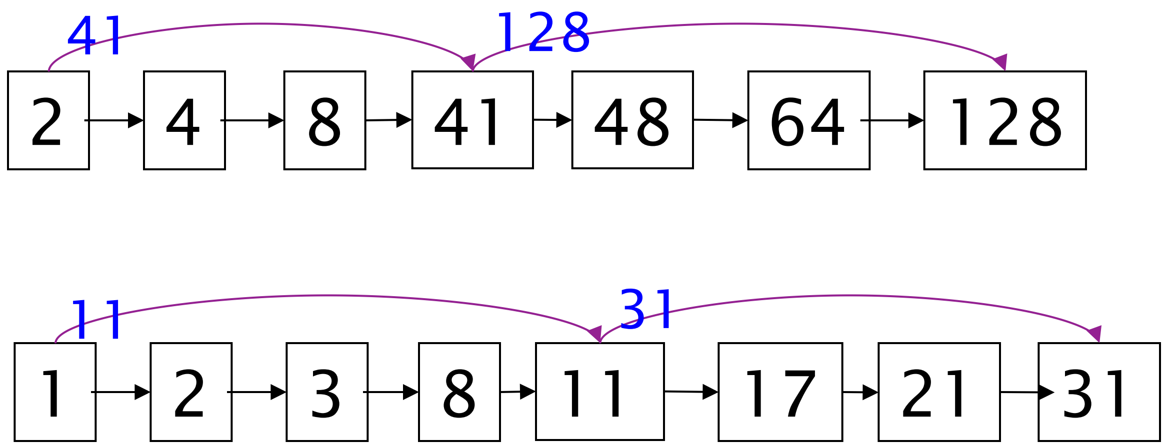
- Walk through the two postings simultaneously, in time linear in the total number of postings entries



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

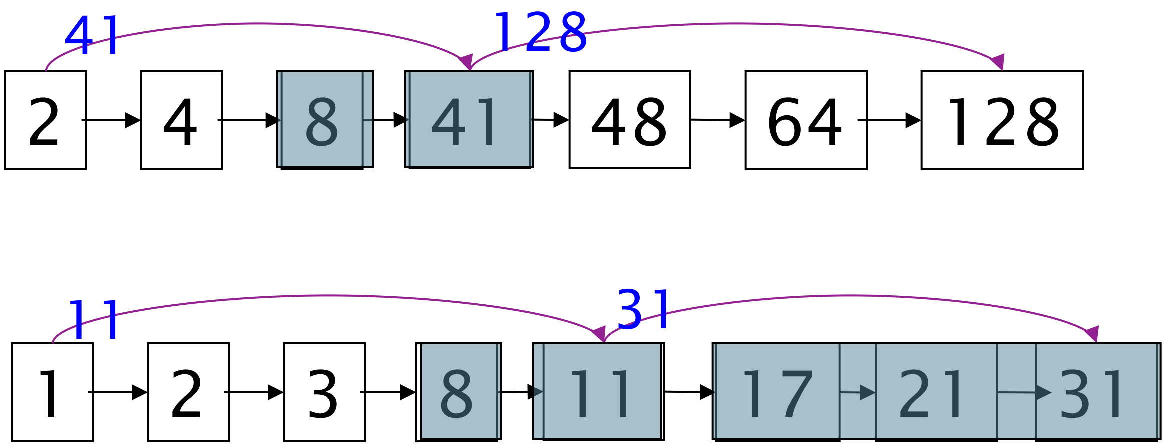
Can we do better?
Yes (if the index isn't changing too fast).

Augment postings with skip pointers (at indexing time)



- Why? To skip postings that are irrelevant.
- 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.

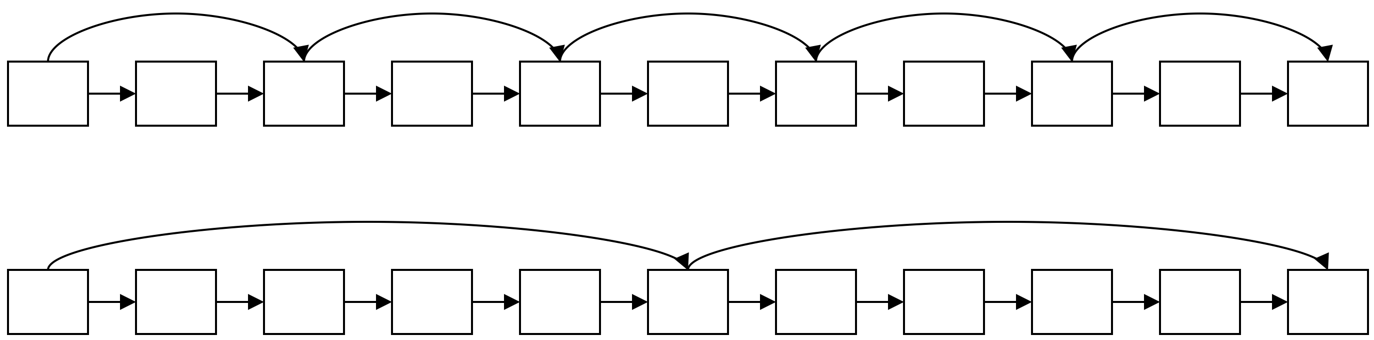
But the skip successor of **11** on the lower list is **31**, so we can skip ahead past the intervening postings.

Algorithm

```
INTERSECTWITHSKIPS(p1, p2)
answer ← ⟨ ⟩
while p1 ≠ NIL and p2 ≠ NIL
do if docID(p1) = docID(p2)
     then ADD(answer, docID(p1))
          p1 ← next(p1)
          p2 ← next(p2)
     else if docID(p1) < docID(p2)
            then if hasSkip(p1) and (docID(skip(p1)) ≤ docID(p2))
                   then while hasSkip(p1) and (docID(skip(p1)) ≤ docID(p2))
                          do p1 ← skip(p1)
                   else p1 ← next(p1)
            else if hasSkip(p2) and (docID(skip(p2)) ≤ docID(p1))
                   then while hasSkip(p2) and (docID(skip(p2)) ≤ docID(p1))
                          do p2 ← skip(p2)
                   else p2 ← next(p2)
return answer
```

Where do we place skips?

- Tradeoff:
  - More skips → shorter skip spans ⇒ more likely to skip. But lots of comparisons to skip pointers.
  - Fewer skips → few pointer comparison, but then long skip spans ⇒ few successful skips.



Placing skips

- Simple heuristic: for postings of length *L*, use $\sqrt{L}$ evenly-spaced skip pointers [Moffat and Zobel 1996]
- 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 [Bahle et al. 2002]

Home Work # 2(a)

Exercise 2.6 [⋆]

We have a two-word query. For one term the postings list consists of the following 16 entries:

[4,6,10,12,14,16,18,20,22,32,47,81,120,122,157,180]

and for the other it is the one entry postings list:

[47].

Work out how many comparisons would be done to intersect the two postings lists with the following two strategies. Briefly justify your answers:

a. Using standard postings lists

b. Using postings lists stored with skip pointers, with a skip length of $\sqrt{P}$, as suggested in Section 2.3.

Homework # 2(b)

- Fast phrase querying with combined indexes (Williams, Zobel, Bahle 2004)
- Efficient phrase querying with an auxiliary index (Bahle, Williams, Zobel 2002)
- A skip list cookbook (Pugh 1990)

- Read these articles and submit one page summary for each of them. It can happen I will ask about it some day.

Positional postings and phrase queries

- **Phrase queries**
  - To answer queries such as "***stanford university***" – as a phrase
    - Sentence *"I went to university **at** Stanford"* is not a match.
    - Sentence *"The inventor Stanford **Ovshinsky never went to** university"* is not a match.
  - Phrase queries has proven easily understood by users;
  - 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 and two-word phrase query-processing is now immediate.

Longer phrase queries

- Longer phrases are processed as we did with wild-cards:
  - ***stanford university palo alto*** can be broken into the Boolean query on biwords:
  - ***stanford university*** *AND* ***university palo*** *AND* ***palo alto***
- Without the docs, we cannot verify that the docs matching the above Boolean query do contain the actual phrase words in a sequence. They can be at dispersed locations in a doc resulting occurrence of given phrase positively while it is non-existing.

Can have false positives!

Extended biwords

- Parse the text using part-of-speech-tagging (POST).
- 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
- Biword indexes are not the standard solution but can be part of a compound strategy

Positional indexes

- Biword index is not the standard solution.

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⟩; . . . ⟩

▶ **Figure 2.11** Positional index example. The word to has a document frequency 993,477, and occurs 6 times in document 1 at positions 7, 18, 33, etc.

Positional indexes

- Suppose the postings lists for *to* and *be* are as in Figure 2.11, and the query is "to be or not to be".
- The postings lists to access are: to, be, or, not. We will examine intersecting the postings lists for to and be. We first look for documents that contain both terms e.g. 1, 4, and 5.
- 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⟩; ...⟩

Positional indexes

- Then, we look for places in the lists where there is an occurrence of *be* with a token index one higher than a position of *to*,
- to, 993427:
  ~~⟨1, 6: ⟨7, 18, 33, 72, 86, 231⟩;~~
  4, 5: ⟨~~8~~, 16, 190, 429, 433⟩;
  ~~5, 2: ⟨363, 367⟩;~~
- be, 178239:
  ~~⟨ 1, 2: ⟨17, 25⟩;~~
  4, 5: ⟨17, 191, ~~291~~, 430, 434⟩;
  ~~5, 3: ⟨14, 19, 101⟩;~~ ...⟩

Positional indexes

- and then we look for another occurrence of each word with token index 4 higher than the first occurrence. In the above lists, the pattern of occurrences that is a possible match is:

- to, 993427:
  4, 5: ⟨~~16, 190,~~ 429, 433⟩;
  ...⟩
- be, 178239:
  4, 5: ⟨~~17, 191,~~ 430, 434⟩;
  ...⟩

Positional indexes

- Same concept within *k* word proximity searches, like
  - employment /3 place
- Here, /*k* means "within *k* words of (on either side)". Clearly, positional indexes can be used for such queries; bi-word indexes cannot.
- Figure 2.12 an algorithm for satisfying within *k* word proximity searches;

Positional indexes

```
PositionalIntersect(p1, p2, k)
answer ← ⟨ ⟩
while p1 ≠ NIL and p2 ≠ NIL
do if docID(p1) = docID(p2)
      then l ← ⟨ ⟩
           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 ≠ ⟨ ⟩ and |l[0] − pos(pp1)| > k
              do Delete(l[0])
              for each ps ∈ 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
```

- https://gist.github.com/pj4dev/33fdaafc4205b927642927193bbf1f3b

Positional indexes

- **Positional index size**
  - You can compress position values/offsets: we'll talk about that later in next lectures.
  - 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
  - SEC filings, books, even some epic poems (heroic based poems)… easily 100,000 terms
- Consider a term with frequency 1 in 1000 terms on average.

Document size	Expected postings	Expected entries in positional posting
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 (Positional index and Biword Index) 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) evaluated 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

Class Exercise

- **Exercise 2.9**
  - Below is a part of index with positions in the form doc1: ⟨pos1,pos2,pos3,...⟩; and doc2: ⟨pos1,pos2,...⟩
    - angels: 2 : ⟨36, 174, 252, 651⟩; 4 : ⟨12, 22, 102, 432⟩; 7 : ⟨17⟩;
    - fools: 2 : ⟨1,17,74,222⟩; 4 : ⟨8,78,108,458⟩; 7 : ⟨3,13,23,193⟩;
    - fear: 2 : ⟨87, 704, 722, 901⟩; 4 : ⟨13, 43, 113, 433⟩; 7 : ⟨18, 328, 528⟩;
    - in: 2 : ⟨3,37,76,444,851⟩; 4 : ⟨10,20,110,470,500⟩; 7 : ⟨5,15,25,195⟩;
    - rush: 2 : ⟨2,66,194,321,702⟩; 4 : ⟨9,69,149,429,569⟩; 7 : ⟨4,14,404⟩;
    - to: 2 : ⟨47, 86, 234, 999⟩; 4 : ⟨14, 24, 774, 944⟩; 7 : ⟨199, 319, 599, 709⟩;
    - tread: 2 : ⟨57, 94, 333⟩; 4 : ⟨15, 35, 155⟩; 7 : ⟨20, 320⟩;
    - where: 2 : ⟨67, 124, 393, 1001⟩; 4 : ⟨11, 41, 101, 421, 431⟩; 7 : ⟨15, 35, 735⟩;
  - The following terms are phrase queries. Which documents correspond to the following queries and on which positions?
    - a) "fools rush in”
    - b) "fools rush in" AND "angels fear to tread".
    - c) The index is incorrect. How?

Class Exercise

- **Exercise 2.9 (Solution)**
  - In order to retrieve the query it is necessary that the words are in a sequence. That is, if the word *angels* is in document 1 on position 3, then the word *fear* have to be in the same document on the position 4.
  - For the exercise **a)** we calculate all possible positions of the phrase.
    - Word *fools* appears in document 2 on positions ⟨1, 17, 74, 222⟩. That means that the word *rush* has to appear on positions ⟨2, 18, 75, 223⟩ and the word *in* on positions ⟨3, 19, 76, 224⟩. Similar process is applied on documents 4 and 7 which retrieves the requested results.
    - Fools:2 : ⟨1,17,74,222⟩; 4 : ⟨8,78,108,458⟩; 7 : ⟨3,13,23,193⟩;
    - rush: 2 : ⟨2,66,194,321,702⟩; 4 : ⟨9,69,149,429,569⟩; 7 : ⟨4,14,404⟩;
    - in: 2 : ⟨3,37,76,444,851⟩; 4 : ⟨10,20,110,470,500⟩; 7 : ⟨5,15,25,195⟩;
  - RESULT: <doc2, doc4, doc7>

Class Exercise

- **Exercise 2.9 (Solution)**
  - For the exercise **b)** we find the requested positions for also the term *angels fear to tread*.
    - angels: 2 : ⟨36, 174, 252, 651⟩; 4 : ⟨12, 22, 102, 432⟩; 7 : ⟨17⟩;
    - fear: 2 : ⟨87, 704, 722, 901⟩; 4 : ⟨13, 43, 113, 433⟩; 7 : ⟨18, 328, 528⟩;
    - to: 2 : ⟨47, 86, 234, 999⟩; 4 : ⟨14, 24, 774, 944⟩; 7 : ⟨199, 319, 599, 709⟩;
    - tread: 2 : ⟨57, 94, 333⟩; 4 : ⟨15, 35, 155⟩; 7 : ⟨20, 320⟩;
  - RESULT: <doc1>
    - They appear in the correct order in **doc4**: {⟨12, 13, 14, 15⟩}. Taking the first part from **a)**, we only check whether the results overlap {**doc2**, **doc4**, **doc7**} ∩ {**doc4**} = **doc4**.
  - For the exercise **c)** we need to have a look into document 7, where on position 15 are two terms *in* and *where*.

Homework #2(c)

Exercise 2.10 [⋆]

Consider the following fragment of a positional index with the format:

word: document: ⟨position, position, . . .⟩; document: ⟨position, . . .⟩
. . .

Gates: 1: ⟨3⟩; 2: ⟨6⟩; 3: ⟨2,17⟩; 4: ⟨1⟩;
IBM: 4: ⟨3⟩; 7: ⟨14⟩;
Microsoft: 1: ⟨1⟩; 2: ⟨1,21⟩; 3: ⟨3⟩; 5: ⟨16,22,51⟩;

The /*k* operator, word1 /*k* word2 finds occurrences of word1 within *k* words of word2 (on either side), where *k* is a positive integer argument. Thus $k = 1$ demands that word1 be adjacent to word2.

a. Describe the set of documents that satisfy the query Gates /2 Microsoft.

b. Describe each set of values for *k* for which the query Gates /*k* Microsoft returns a different set of documents as the answer.

Homework #2(d)

- **Exercise 2.13** [★★]
- Suppose we wish to use a postings intersection procedure to determine simply the list of documents that satisfy a /*k* clause, rather than returning the list of positions, as in Figure 2.12 (page 42). For simplicity, assume $k \geq 2$. Let *L* denote the total number of occurrences of the two terms in the document collection (i.e., the sum of their collection frequencies). Which of the following is true? Justify your answer.
  a. The merge can be accomplished in a number of steps linear in *L* and independent of *k*, and we can ensure that each pointer moves only to the right.
  b. The merge can be accomplished in a number of steps linear in *L* and independent of *k*, but a pointer may be forced to move non-monotonically (i.e., to sometimes back up)
  c. The merge can require *kL* steps in some cases.

Homework #2(e)

- **Exercise 2.14** [⋆⋆]
- How could an IR system combine use of a positional index and use of stop words? What is the potential problem, and how could it be handled?

Homework #2(f)

- Visit the following link and build a positional index based search engine and then submit the report with output.
  - http://www.elemarjr.com/en/2018/02/phrase-queries-and-positional-indexes-in-c/
  - Presentation is due at any time during class hours.

Articles to be Read

- Spoken language identification:
  - Hughes, Baden, Timothy Baldwin, Steven Bird, Jeremy Nicholson, and Andrew MacKinlay. 2006. Reconsidering language identification for written language re- sources. In *Proc. International Conference on Language Resources and Evaluation*, pp. 485–488.
- Discussion of the positive and negative impact of stemming :
  - Hollink, Vera, Jaap Kamps, Christof Monz, and Maarten de Rijke. 2004. Monolingual document retrieval for European languages. *IR* 7(1):33–52.
- Skip pointer extended technique:
  - Boldi, Paolo, and Sebastiano Vigna. 2005. Compressed perfect embedded skip lists for quick inverted-index lookups. In *Proc. SPIRE*. Springer.
  - Strohman, Trevor, and W. Bruce Croft. 2007. Efficient document retrieval in main memory. In *Proc. SIGIR*, pp. 175–182. ACM Press.

Homework (Not for submission)

- Visit the following link; Execute the source code; See the errors in the output and try to remove it.
  - https://github.com/manning/MergeAlgorithms