

CS6322: Information Retrieval

Sanda Harabagiu

Lecture 2: The term vocabulary and postings lists



Recap of the 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

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

Documents to be indexed.



Friends, Romans, countrymen.

⋮

Tokenizer

Token stream.

Friends

Romans

Countrymen

Linguistic modules

Modified tokens.

friend

roman

countryman

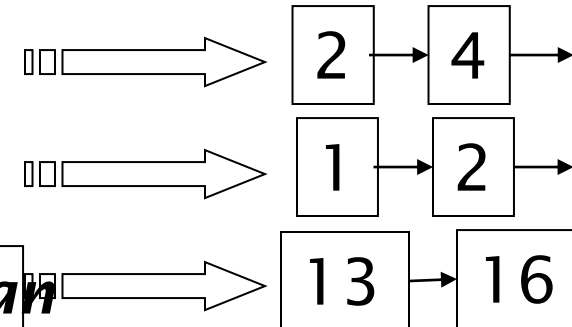
Indexer

Inverted index.

friend

roman

countryman



Parsing a document

- What format is it in?
 - pdf/word/excel/html?
- What language is it in?
- What character set is in use?

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
 - French email with a German 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:
 - ***Finland's capital*** →
Finland? 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. 12, 1991* *20/3/91*
- *55 B.C.*
- *B-52*
- *My PGP key is 324a3df234cb23e*
- *(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: Lecture 3)
 - Will often index “meta-data” separately
 - Creation date, format, etc.

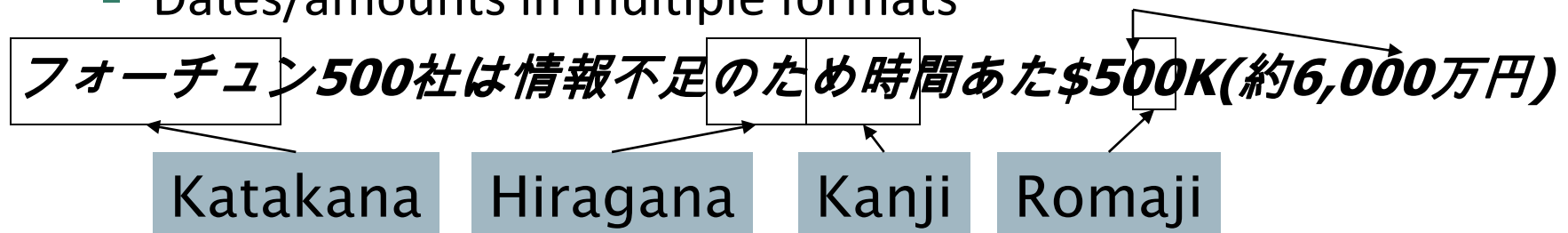
Tokenization: language issues

- 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
 - Internationalization!
- 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.'
- With Unicode, the surface presentation is complex, but the stored form is straightforward

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: ~30% of postings for top 30 words
- But the trend is away from doing this:
 - Good compression techniques (lecture 5) means the space for including stopwords in a system is very small
 - Good query optimization techniques (lecture 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

- 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*
 - *Japanese use of kana vs. Chinese characters*
- 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 into the same form

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...
- Google example:
 - Query **C.A.T.**
 - #1 result is for “cat” (well, Lolcats) *not* Caterpillar Inc.



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
- More in lectures 3 and 9

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

Criteria for Judging Stemmers

- There are three criteria for evaluating stemmers:
 1. *Correctness*
 2. *Retrieval efficiency*
 3. *Compression performance*
- There are two ways in which stemming can be incorrect:
 1. Over-stemming (too much of the term is removed!)
 - *Over-stemming can cause unrelated terms to be related*
 - *The effect on IR performance is retrieval of non-relevant documents*
 2. Under-stemming (too little of the term is removed)
 - *Under-stemming will prevent related terms from being conflated*
 - *The effect of under-stemming on IR performance is that relevant documents will not be retrieved.*

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.*

Porter algorithm

The Porter algorithm consists of a *set of condition/action rules*.

The conditions fall into three classes:

1. conditions on the stem,
2. condition on the suffix,
3. and conditions on the rules.

There are several types of stem conditions.

indicate an optional occurrence. Some examples of measures for terms follows.

Typical rules in Porter

- *sses* → *ss*
- *ies* → *i*
- *ational* → *ate*
- *tional* → *tion*

- Weight of word sensitive rules
- $(m > 2)$ *EMENT* →
 - *replacement* → *replac*
 - *cement* → *cement*

Conditions for Porter Algorithm

Condition 1. The *measure*, denoted m , of a stem is based on its alternate vowel-consonant sequences.

Vowels are *a, e, i, o, u*, and *y* if *y* is preceded by a consonant.

Consonants are all letters that are not vowels. Let C stand for a sequence of consonants, and V for a sequence of vowels.

The measure m , then, is defined as

$$[C](VC)^m[V]$$

the superscript m in the equation, which is the measure, indicates the number of VC sequences. Square brackets indicate an optional occurrence. Some examples of measures for terms follows.

Measure	Examples
m=0	TR, EE, TREE, Y, BY
m=1	TROUBLE, OATS, TREES, IVY
m=2	TROUBLES, PRIVATE, LOATEN

Condition 2. *<X> —the stem ends with a given letter X

Condition 3. *v* —the stem contains a vowel

Condition 4. *d —the stem ends in a double consonant

Condition 5. *o —the stem ends with a consonant-vowel-consonant, sequence, where the final consonant is not w, x, or y.

Suffix Conditions

Suffix conditions take the form:

(current_suffix == pattern).

Rule conditions take the form: (rule was used).

Actions are rewrite rules of the form:

old_suffix->new_suffix

The rules are divided into steps. The rules in a step are examined in sequence, and only one rule from a step can apply. The *longest possible suffix is always removed* because of the ordering of the rules within a step.

Rules

- In a set of rules written one beneath the others, only one is obeyed !
 - ➔ The rule with the longest matching S_1 for the given word

- Example:

if $SSES \rightarrow SS$

$IES \rightarrow I$

$SS \rightarrow$

$ES \rightarrow E$

$CARESSES \rightarrow CARESS$

$CARES \rightarrow CARE$

The Porter Algorithm

Step 1.a

SSES → SS

IES → I

SS → SS

S →

Step 1.b

(m>0) EED → EE

(*v*) ED →

(*v*) ING →

Examples:

caresses → caress

ponies → poni

ties → ti

caress → caress

cats → cat

agreed → agree

plastered → plaster

bled → bled

sing → sing

Special Case

if 2nd or 3rd rules from Step 1.b is successful then

AT → ATE

BL → BLE

IZ → IZE

(*d and not (*L or *S or *Z))
→ single letter

conflat(ed) → conflate

troubl(ing) → trouble

siz(ed) → size

hopp(ing) → hop

tann(ed) → tan

Fall(ing) → fall

Fail(ing) → fail

Fil(ing) → file

(m=1 and *o) → E

The rule to map to a single letter causes the removal of one of the double letter pair. The -E is put back on -AT, -BL and -IZ, so that the suffixes -ALE, -BLE and -IZE can be recognized later. This E may be removed in step 4.

The Porter Algorithm - cont

Step 1.c

$(*v*) Y \rightarrow i$

happy \rightarrow happi

sky \rightarrow sky

Step 1 deals with plurals and past participles. The subsequent steps are much more straightforward.

The Porter Algorithm - cont

Step 2:

(m>0) ATIONAL	→ ATE	relational	→ relate
(m>0) TIONAL	→ TION	conditional	→ condition
		rational	→ rational
(m>0) ENCI	→ ENCE	valenci	→ valence
(m>0) ANCI	→ ANCE	hesitanci	→ hesitance
(m>0) IZER	→ IZE	digitizer	→ digitize
(m>0) ABLI	→ ABLE	conformabli	→ conformable
(m>0) ALLI	→ AL	radicalli	→ radical
(m>0) ENTLI	→ ENT	differentli	→ different
(m>0) ELI	→ E	vileli	→ vile
(m>0) OUSLI	→ OUS	analogousli	→ analogous
(m>0) IZATION	→ IZE	vietnamization	→ vietnamize
(m>0) ATION	→ ATE	predication	→ predicate
(m>0) ATOR	→ ATE	operator	→ operate

The Porter Algorithm - cont

(m>0) ALISM	→ AL	feudalism	→ feudal
(m>0) IVENESS	→ IVE	decisiveness	→ decisive
(m>0) FULNESS	→ FUL	hopefulness	→ hopeful
(m>0) OUSNESS	→ OUS	callousness	→ callous
(m>0) ALITI	→ AL	formaliti	→ formal
(m>0) IVITI	→ IVE	sensitiviti	→ sensitive
(m>0) BILITI	→ BLE	sensibiliti	→ sensible

the test for the string S1 can be made fast by doing a program switch on the penultimate letter of the word being tested. This gives a fairly even breakdown of the possible values of the string S1. It will be seen in fact that the S1-strings in step 2 are presented here in the alphabetical order of their penultimate letter. Similar techniques may be applied in the other steps

The Porter Algorithm - cont

Step 3

(m>0) ICATE → IC
 (m>0) ATIVE →
 (m>0) ALIZE → AL
 (m>0) ICITI → IC
 (m>0) ICAL → IC
 (m>0) FUL →
 (m>0) NESS →

triplicate → triplic
 formative → form
 formalize → formal
 electriciti → electric
 electrical → electric
 hopeful → hope
 goodness → good

Step 4

(m>0) AL →
 (m>0) ANCE →
 (m>0) ENCE →
 (m>0) ER →
 (m>0) IC →
 (m>0) ABLE →
 (m>0) IBLE →

revival → reviv
 allowance → allow
 inference → infer
 airliner → airlin
 gyroscopic → gyroscop
 adjustable → adjust
 defensible → defens

The Porter Algorithm - cont

(m>0) ANT	→	irritant	→ irrit
(m>0) EMENT	→	replacement	→ replac
(m>0) MENT	→	adjustment	→ adjust
(m>0) ENT	→	dependent	→ depend
(m>0) and (*S or *T)) ION	→	adoption	→ adopt
(m>0) OU	→	homologou	→ homolog
(m>0) ISM	→	communism	→ commun
(m>0) ATE	→	activate	→ activ
(m>0) ITI	→	angulariti	→ angular
(m>0) OUS	→	homologous	→ homolog
(m>0) IVE	→	effective	→ effect
(m>0) IZE	→	bowdlerize	→ bowdler

The suffixes are now removed. All that remains is a little tidying up.

The Porter Algorithm - cont

Step 5a

(m>1) E →
 probate → probat
 rate → rate
(m=1 and not *o) →
 cease → ceas

Step 5b

(m> 1 and *d and *L) → single letter
 controll → control
 roll → roll

The algorithm is careful not to remove a suffix when the stem is too short, the length of the stem being given by its measure, m. There is no linguistic basis for this approach. It was merely observed that m could be used quite effectively to help decide whether or not it was wise to take off a suffix.

Porter Stemmer

- Assumption: terms with a common stem have similar meanings

CONNECT

CONNECTED

CONNECTING

CONNECTION

CONNECTIONS

- *Frequently the performance of an IR system would be improved if term groups are CONFLATED into a single term.*
- How ? ➔ Remove suffixes: -ED, -ING, ION, IONS

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 – first cut

ensemble.french

時間.japanese

MIT.english

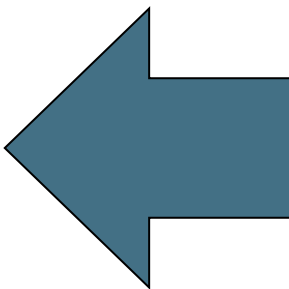
mit.german

guaranteed.english

entries.english

sometimes.english

tokenization.english

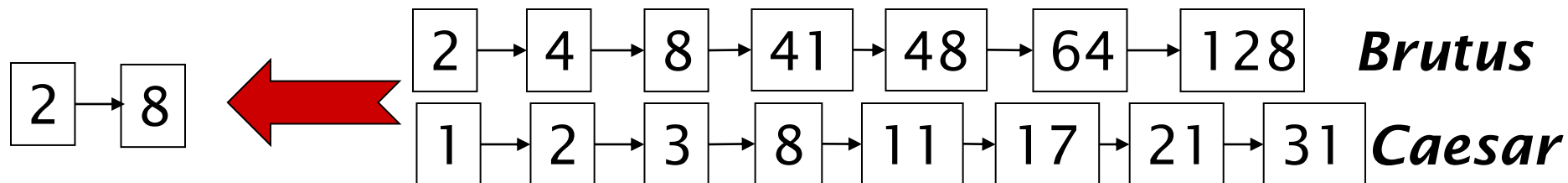


These may be grouped by language (or not...). More on this in ranking/query processing.

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

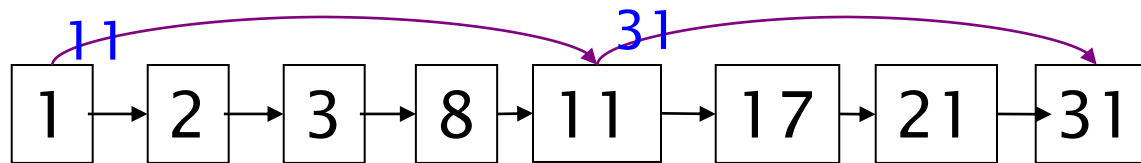
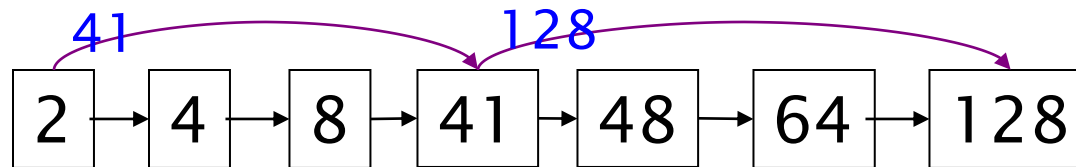


If the list lengths are m and n , the merge takes $O(m+n)$ operations.

Can we do better?

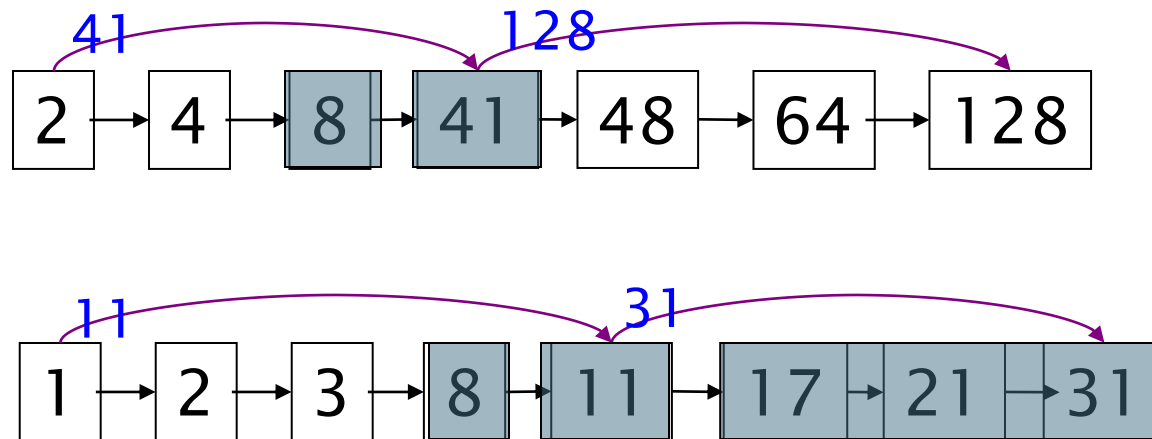
Yes (if index isn't changing too fast).

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



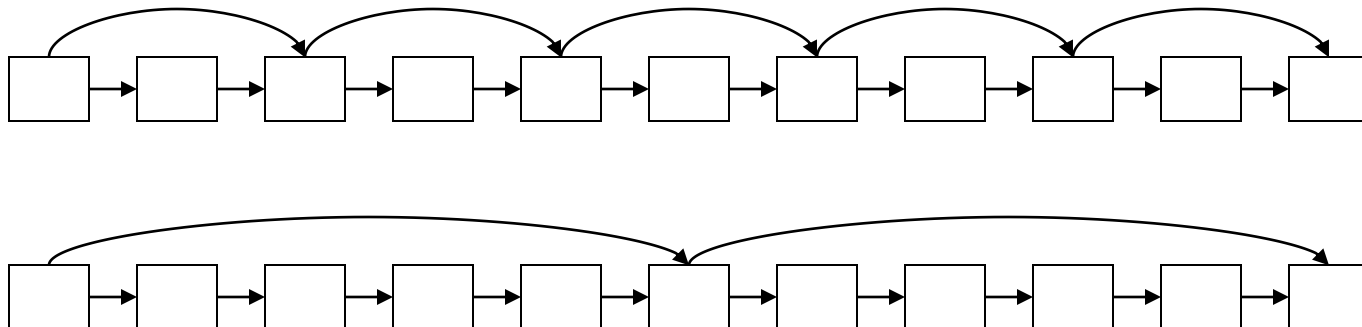
Suppose we've stepped through the lists until we process **8** on each list. We match it and advance.

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 comparisons, 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 (Bahle et al. 2002) 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 “*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*
- 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

- 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 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

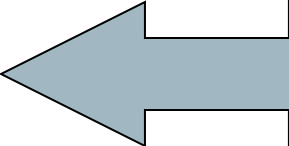
<*be*: 993427;

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.
- 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!

Positional index size

- You can compress position values/offsets: we'll talk about that in lecture 5
- 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 ... easily 100,000 terms
- Consider a term with frequency 0.1%



Document size	Postings	Positional postings
1 000	1	1
1 00,000	1	1 00

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 $\frac{1}{4}$ 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

- IIR 2
- MG 3.6, 4.3; MIR 7.2
- Porter's stemmer:
<http://www.tartarus.org/~martin/PorterStemmer/>
- Skip Lists theory: Pugh (1990)
 - Multilevel skip lists give same $O(\log n)$ efficiency as trees
- H.E. Williams, J. Zobel, and D. Bahle. 2004. "Fast Phrase Querying with Combined Indexes", ACM Transactions on Information Systems.
<http://www.seg.rmit.edu.au/research/research.php?author=4>
- D. Bahle, H. Williams, and J. Zobel. Efficient phrase querying with an auxiliary index. SIGIR 2002, pp. 215-221.