Introduction to

Information Retrieval

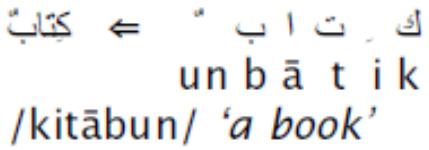
Term vocabulary and postings lists – preprocessing steps

Agenda

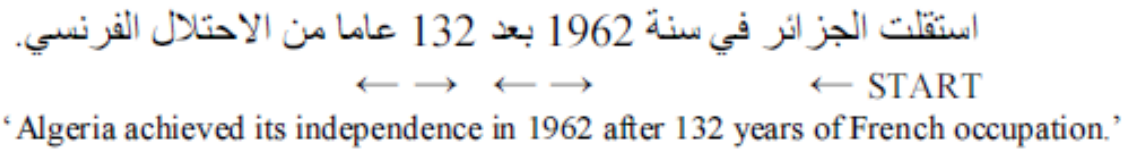
- Recall the major steps in inverted index construction:
  1. Collect the documents to be indexed.
  2. Tokenize the text (Tokenization is the process of chopping character streams into tokens).
  3. Do linguistic preprocessing (deals with building equivalence classes of tokens which are the set of terms that are indexed) of tokens.
  4. Index the documents that each term occurs in.
- In this chapter we discuss
  - how the basic unit of a document can be defined and how the character sequence that it comprises is determined
  - Linguistic issues of tokenization and linguistic preprocessing, which determine the vocabulary of terms which a system uses .
  - implementation of postings lists.
  - extended postings list data structure that supports faster querying
  - building postings data structures suitable for handling phrase and proximity queries

Document delineation and character sequence decoding

- **Obtaining the character sequence in a document**
- Digital documents that are the input to an indexing process are typically bytes in a file or on a web server.
- The first step of processing is to convert this byte sequence into a linear sequence of characters.
- For the case of plain English text in ASCII encoding, this is trivial
- Challenge 1
  - The sequence of characters may be encoded by one of various single byte or multibyte encoding schemes, such as Unicode UTF-8.We need to determine the correct encoding.
  - Once the encoding is determined, we decode the byte sequence to a character sequence.
- Challenge 2
- The characters may have to be decoded out of some binary representation like MicrosoftWord DOC files and/or a compressed format such as zip files.
  - We must determine the document format, and then an appropriate decoder has to be used.
  - In XML documents character entities, such as `&amp`, need to be decoded to give the correct character, namely `&` for `&amp`.
- Challenge 3 :
  - Ignoring markup(consisting of a set of symbols inserted in a text document to control its structure)



An example of a vocalised Arabic word The writing is from right to left and letters undergo complex mutations as they are combined.



In languages that are written right-to-left, such as Hebrew and Arabic, it is quite common to also have left-to-right text interspersed, such as numbers and dollar amounts.

Choosing a document unit

- Answering the question "what is a document?" is not trivial and requires some design decisions.
- Documents are fixed units for the purposes of indexing.
- Each file in a folder as a document
- A traditional Unix (mbox-format) email file stores a sequence of email messages (an email folder) in one file, but you might wish to regard each email message as a separate document.
- Many email messages now contain attached documents, and you might then want to regard the email message and each contained attachment as separate documents.
- If an email message has an attached zip file, you might want to decode the zip file and regard each file it contains as a separate document.
- You might want to combine multiple files into a single document.

- For a collection of books, it would usually be a bad idea to index an entire book as a document. Instead, we may well wish to index each chapter or paragraph as a mini-document.
- If the units get too small, we are likely to miss important passages because terms were distributed over several mini-documents, while if units are too large we tend to get spurious matches and the relevant information is hard for the user to find.
- An IR system should be designed to offer choices of granularity. For this choice to be made well, the person who is deploying the system must have a good understanding of the document collection, the users, and their likely information needs and usage patterns.

Determining the vocabulary of terms

Tokenization:

- Given a character sequence and a defined document unit, tokenization is the task of chopping it up into pieces, called *tokens*, at the same time throwing away certain characters, such as punctuations.

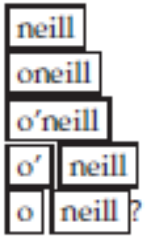
Input: Friends, Romans, Countrymen, lend me your ears;
Output: | Friends | Romans | Countrymen | lend | me | your | ears |

  - These tokens are referred to as terms or words
  - A *token* is an instance of a sequence of characters in some particular document that are grouped
  - A *type* is the class of all TERM tokens containing the same character sequence.
  - A *term* is a (perhaps normalized) type that is included in the IR system's dictionary.

For example, if the document to be indexed is *to sleep perchance to dream*, then there are 5 tokens, but only 4 types. However, if *to* is omitted from the index, then there will be only 3 terms: *sleep*, *perchance*, and *dream*.

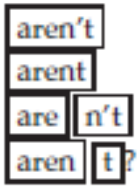
- The major question of the tokenization phase is what are the correct tokens to use?

For *O'Neill*, which of the following is the desired tokenization?



A simple strategy is to just split on all non-alphanumeric

And for *aren't*, is it:



Tokenization problems: One word or two? (or several)

- Hewlett-Packard
- State-of-the-art
- co-education
- the hold-him-back-and-drag-him-away maneuver
- data base
- San Francisco
- Los Angeles-based company
- cheap San Francisco-Los Angeles fares
- York University vs. New York University

These issues of tokenization are language-specific. It thus requires the language of the document to be known.

Tokenization problems: Numbers

- 3/20/91
- 20/3/91
- Mar 20, 1991
- B-52
- 100.2.86.144
- (800) 234-2333
- 800.234.2333
- Older IR systems may not index numbers . . .
- . . . but generally it's a useful feature.

Problems in tokenization for other languages, e.g., no whitespace in Chinese

莎拉波娃现在居住在美国东南部的佛罗里达。今年４月９日，莎拉波娃在美国第一大城市纽约度过了１８岁生日。生日派对上，莎拉波娃露出了甜美的微笑。

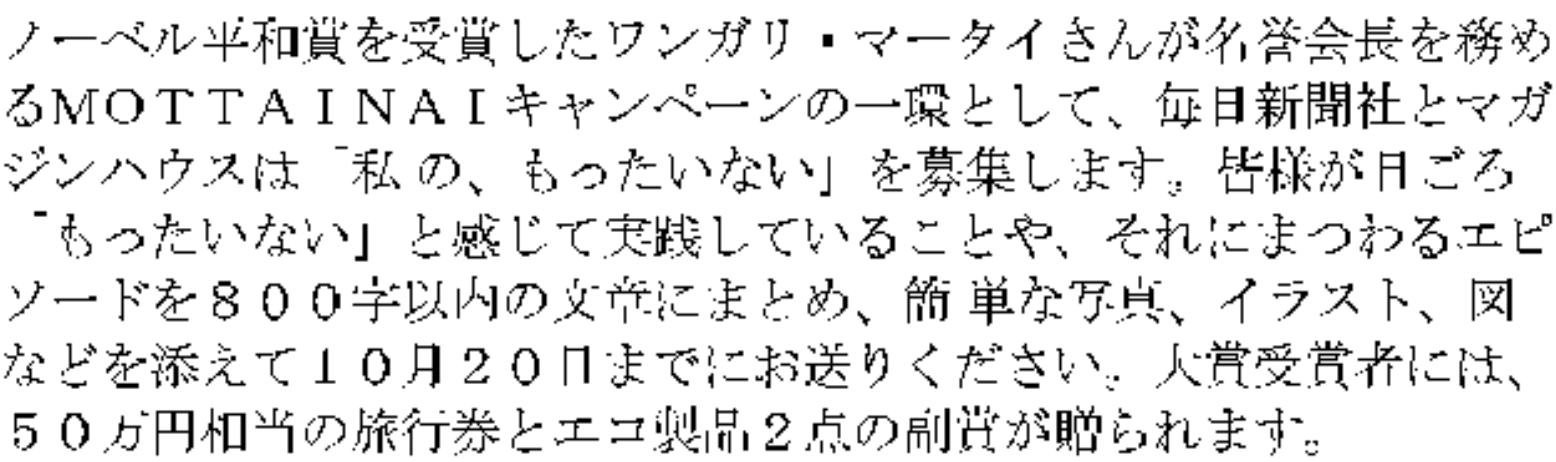
Ambiguous segmentation in Chinese

The two characters can be treated as one word meaning ‘monk’ or as a sequence of two words meaning ‘and’ and ‘still’.

Other cases of "no whitespace"

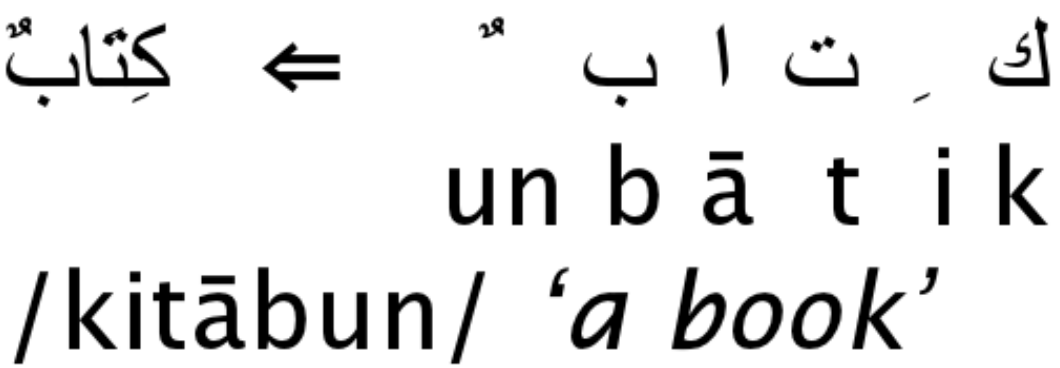
- Compounds in Dutch, German, Swedish
- Computerlinguistik → Computer + Linguistik
- Lebensversicherungsgesellschaftsangestellter
- → leben + versicherung + gesellschaft + angestellter
- Inuit: tusaatsiarunnanngittualuujunga (I can't hear very well.)
- Many other languages with segmentation difficulties: Finnish, Urdu, . . .
- CamelCase in social media

Japanese


ノーベル平和賞を受賞したワンガリ・マータイさんが名誉会長を務めるMOTTAINAIキャンペーンの一環として、毎日新聞社とマガジンハウスは「私の、もったいない」を募集します。皆様が日ごろ「もったいない」と感じて実践していることや、それにまつわるエピソードを８００字以内の文章にまとめ、簡単な写真、イラスト、図などを添えて１０月２０日までにお送りください。大賞受賞者には、５０万円相当の旅行券とエコ製品２点の副賞が贈られます。

4 different “alphabets”: Chinese characters, hiragana syllabary for inflectional endings and functional words, katakana syllabary for transcription of foreign words and other uses, and latin. No spaces (as in Chinese). End user can express query entirely in hiragana!

Arabic script



Arabic script: Bidirectionality

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

← → ← → ← START

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

Bidirectionality is not a problem if text is coded in Unicode.

Dropping common terms: stop words

- Some extremely common words which would appear to be of little value in helping select documents matching a user need are excluded from the vocabulary entirely. These words are called STOP WORDS.
- Strategy for determining a stop list is to sort the terms by *collection frequency* and then to take the most frequent terms.

a	an	and	are	as	at	be	by	for	from
has	he	in	is	it	its	of	on	that	the
to	was	were	will	with					

▶ **Figure 2.5** A stop list of 25 semantically non-selective words which are common

- Using a stop list significantly reduces the number of postings that a system has to store
- And not indexing stop words does little harm
- But, in the meaning of *flights to London* is likely to be lost if the word to is removed.
- The general trend in IR systems over time has been from standard use of quite large stop lists (200–300 terms) to very small stop lists (7–12 terms) to no stop list whatsoever.
- Web search engines generally do not use stop lists.
- Most modern IR systems, the additional cost of including stop words is not that big – neither in terms of index size nor in terms of query processing time.

Token Normalization

- Tokens in the query must just match tokens in the token list of the document.
- But, in may cases two character sequences are not quite the same but we would like a match to occur

  Example: We want to match *U.S.A.* and *USA*
- *Token normalization* is the process of canonicalizing (**one representation into a standard approved format)** TOKEN tokens so that matches occur despite superficial differences in the character sequences of the tokens.
- We commonly implicitly define equivalence classes of terms – efficient - there are more postings to store and merge
- Eg. tokens *anti-discriminatory* and *antidiscriminatory* are both mapped onto the term antidiscriminatory, in both the document text and queries, then searches for one term will retrieve documents that contain either.
- An alternative to creating equivalence classes is to maintain relations between unnormalized tokens. This method can be extended to hand-constructed lists of synonyms such as *car* and *automobile*,

- The usual way to implement is to
  - index unnormalized tokens and maintain a query expansion list of multiple vocabulary entries to consider for a certain query term.
  - The alternative is to perform the expansion during index construction. When the document contains automobile, we index it under car as well

Accents and diacritics

- Accents: résumé vs. resume (simple omission of accent)
- Most important criterion: How are users likely to write their queries for these words?
- Even in languages that standardly have accents, users often do not type them. (Polish?)

Case folding

- Usually: Reduce all letters to lower case
- Possible exceptions: capitalized words in mid-sentence
- MIT vs. mit
- Fed vs. fed
- It's often best to lowercase everything since users will use lowercase regardless of correct capitalization

Stemming and Lemmatization

- Goal of both same: reduce inflectional forms (**a changed spelling that shows the way it is used in sentences**) and derivationally related forms (Terms in different syntactic categories that have the same root form and are semantically related) to a common base form.
- Inflectional Form : Eg. "Finds" and "found" are inflected forms of "find."
- Derivationally related forms: Eg. The verb "to butter" has the same root form as the noun "butter", they are different syntactic categories (verb vs. noun), and they're clearly semantically related.
- Stemming refers to a heuristic process that chops off the ends of words in the hope of achieving the goal correctly most of the time
- Lemmatization implies doing "proper" reduction to dictionary headword form (the lemma - the basic form of a word that is shown in a dictionary entry, for example the singular form of a noun or the infinitive form - **First, find the word to. Second, if the word after to is a verb, then it is an infinitive**. of a verb), using dictionary and morphological analysis of words

- Token *saw*, stemming might return just *s*, whereas lemmatization would attempt to return either *see* or *saw* depending on whether the use of the token was as a verb or a noun.
- Stemming most commonly collapses derivationally related words, whereas lemmatization commonly only collapses the different inflectional forms of a lemma.
- Linguistic processing for stemming or lemmatization is often done by an additional plug-in component to the indexing process

Lemmatization

- Reduce inflectional/variant forms to base form
- Example: *am, are, is → be*
- Example: *car, cars, car's, cars' → car*
- Example: *the boy's cars are different colors → the boy car be different color*
- Inflectional morphology (*cutting → cut)* vs. derivational morphology (*destruction → destroy)*
- After lemmatization, we will be getting a valid word that means the same thing.

Stemming

- Heuristic process that chops off the ends of words in the hope of achieving what "principled" lemmatization attempts to do with a lot of linguistic knowledge.
- Language dependent
- Often inflectional and derivational
- Example for derivational: *automate, automatic, automation* all reduce to *automat*
- It is just like cutting down the branches of a tree to its stems. For example, the stem of the words ***eating, eats, eaten*** is ***eat***.

Porter algorithm

- Most common algorithm for stemming English
- Results suggest that it is at least as good as other stemming options
- Conventions + 5 phases of reductions
- Phases are applied sequentially
- Each phase consists of a set of commands.
  - Sample command: Delete final *ement* if what remains is longer than 1 character
  - replacement → replac
  - cement → cement
- Sample convention: Of the rules in a compound command, select the one that applies to the longest suffix.

Porter stemmer: A few rules

Rule	Example
SSES → SS	caresses → caress
IES → I	ponies → poni
SS → SS	caress → caress
S →	cats → cat

- PORTER STEMMING ALGORITHM –

Three stemmers: A comparison

Sample text: Such an analysis can reveal features that are not easily visible from the variations in the individual genes and can lead to a picture of expression that is more biologically transparent and accessible to interpretation

Porter stemmer: such an analysi can reveal featur that ar not easili visibl from the variat in the individu gene and can lead to pictur of express that is more biolog transpar and access to interpret

Lovins stemmer: such an analys can reve featur that ar not eas vis from th vari in th individu gen and can lead to a pictur of expres that is mor biolog transpar and acces to interpres

Paice stemmer: such an analys can rev feat that are not easy vis from the vary in the individ gen and can lead to a pict of express that is mor biolog transp and access to interpret

Does stemming improve effectiveness?

- In general, stemming increases effectiveness for some queries, and decreases effectiveness for others.
- Queries where stemming is likely to help: [tartan sweaters], [sightseeing tour san francisco] (equivalence classes: {sweater,sweaters}, {tour,tours})
- Porter Stemmer equivalence class *oper* contains all of *operate operating operates operation operative operatives operational.*
- Queries where stemming hurts: [operational AND research], [operating AND system], [operative AND dentistry]

Stemming v/s Lemmatization

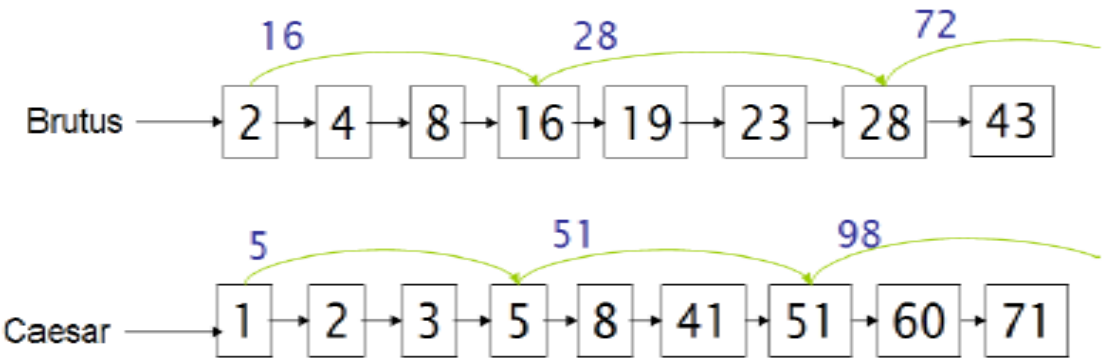
Stemming	Lemmatization
Stemming is a process that stems or removes last few characters from a word, often leading to incorrect meanings and spelling.	**Lemmatization** considers the context and converts the word to its meaningful base form, which is called Lemma.
For instance, stemming the word '**Caring**' would return '**Car**'.	For instance, lemmatizing the word '**Caring**' would return '**Care**'.
Stemming is used in case of large dataset where performance is an issue.	Lemmatization is computationally expensive since it involves look-up tables .

Faster postings list intersection via skip pointers

- The basic postings list intersection: we walk through the two postings lists simultaneously, in time linear in the total number of postings entries. If the list lengths are m and n, the intersection takes $O(m + n)$ operations. Can we do better than this?
- One way to do this is to use a *skip list* by augmenting SKIP LIST postings lists with skip pointers (at indexing time)
- Skip pointers are effectively shortcuts that allow us to avoid processing parts of the postings list that will not figure in the search results.

Two issues : 1. where to place skip pointers and
2. how to do efficient merging using skip pointers.

- Merging:



Note: that the presence of skip pointers only helps for AND queries, not for OR queries.

```
INTERSECTWITHSKIPS(p1, p2)
 1  answer ← ⟨ ⟩
 2  while p1 ≠ NIL and p2 ≠ NIL
 3  do if docID(p1) = docID(p2)
 4        then ADD(answer, docID(p1))
 5             p1 ← next(p1)
 6             p2 ← next(p2)
 7        else if docID(p1) < docID(p2)
 8                then if hasSkip(p1) and (docID(skip(p1)) ≤ docID(p2))
 9                     then while hasSkip(p1) and (docID(skip(p1)) ≤ docID(p2))
10                          do p1 ← skip(p1)
11                     else p1 ← next(p1)
12                else if hasSkip(p2) and (docID(skip(p2)) ≤ docID(p1))
13                     then while hasSkip(p2) and (docID(skip(p2)) ≤ docID(p1))
14                          do p2 ← skip(p2)
15                     else p2 ← next(p2)
16  return answer
```

▶ **Figure 2.10** Postings lists intersection with skip pointers.

Assignment 1

1. Word **global** is available in documents 2 3 and 4. Word **algorithm** in documents 1 3 and 5, Word **collection** in documents 2, 4 and 6. Word **complete** in documents 1 and 2 . Word **december** in documents 4 and 5. Draw the term-document incidence matrix. Given the query

global AND **collection** AND NOT **complete** determine the documents satisfying the query.

2.Given the words, Evaluation, Normalisation and Implementing, to Porter Stemming algorithm, determine the output. Write the complete steps involved.

3. Draw the inverted index that would be built for the following document collection.

- Doc 1 – branded cotton dress display
- Doc 2 - dress demand increases during december
- Doc 3 - cold weather in december
- Doc 4 – drastic weather change due to global warming

- 4.

Shown below is a portion of a positional index in the format: term: doc1: (position1, position2, ...); doc2: (position1, position2, ...); etc.

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: (16,36,736);

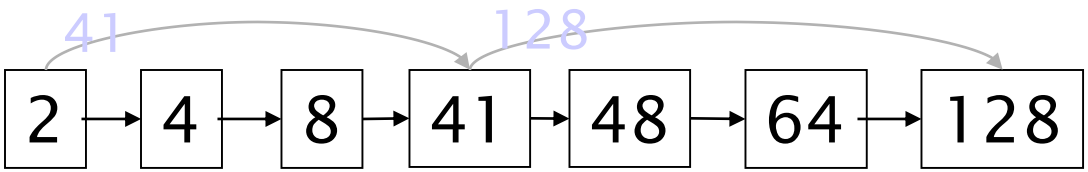
Which document(s) if any match each of the following queries, where each expression within quotes is a phrase query?

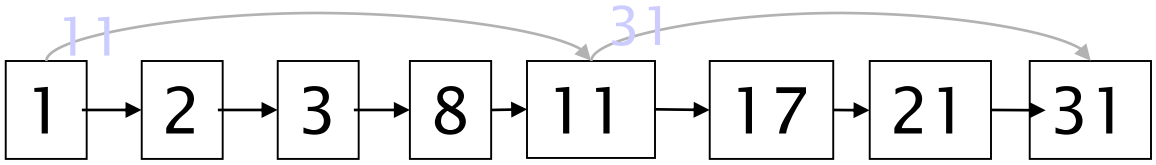
a. "fools rush in"

b. "fools rush in" AND "angels fear to tread"

- Where do we place skips?
- shorter skip spans – more skips - lots of comparisons to skip pointers, and lots of space storing skip pointers.
- long skip spans - Fewer skips – few pointer comparisons, but there will be fewer opportunities to skip.
- A simple heuristic for placing skips, which has been found to work well in practice, is that for a postings list of length P, use $\sqrt{P}$ evenly-spaced skip pointers.
- **Building effective skip pointers is easy if an index is relatively static; it is harder if a postings list keeps changing because of updates**

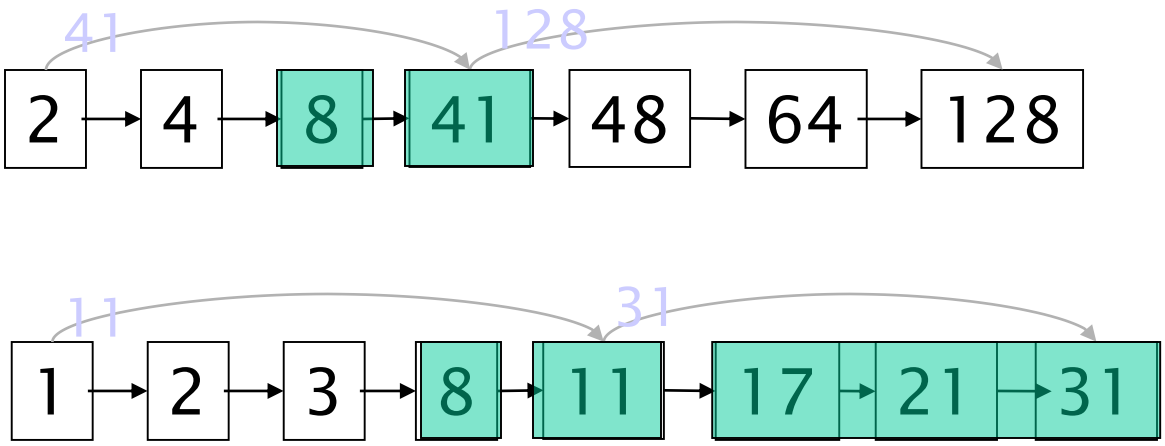
Augment postings with skip pointers (at indexing time)





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

Query processing with skip pointers



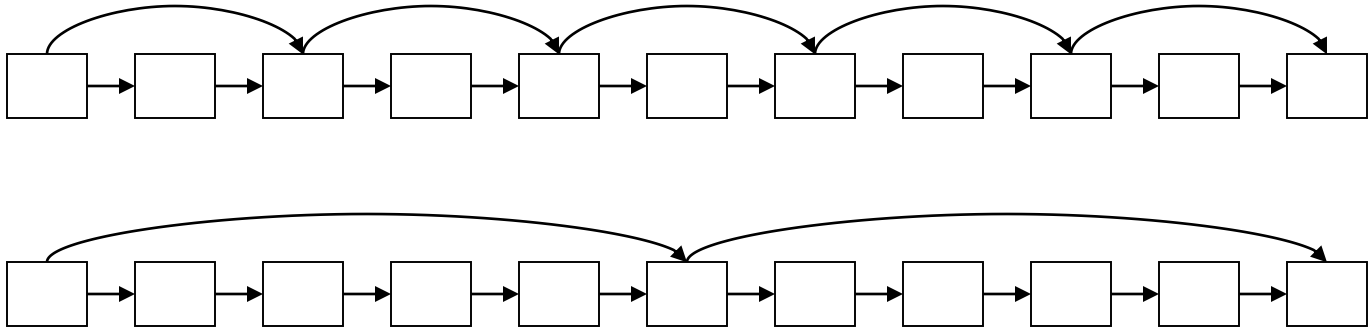
Suppose we've stepped through the lists until we process **8** on each list. We match it and advance.

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 → 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.
- 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

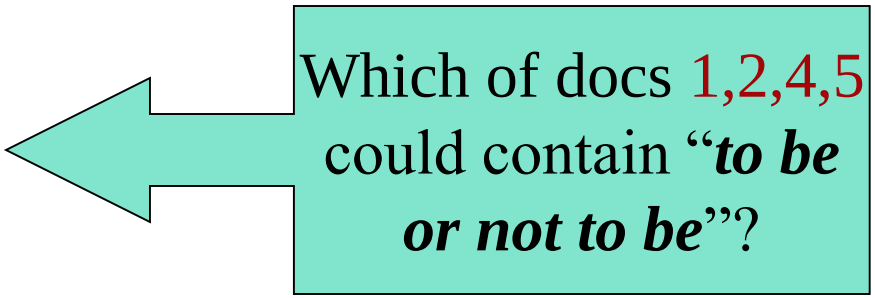
Solution 2: Positional indexes

- 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, …>



- 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

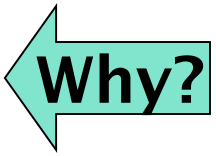
- LIMIT! /3 STATUTE /3 FEDERAL /2 TORT
  - Again, here, /*k* means "within *k* words of".
- Clearly, positional indexes can be used for such queries; biword indexes cannot.
- Exercise: Adapt the linear merge of postings to handle proximity queries. Can you make it work for any value of *k*?
  - This 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
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"***
- Williams et al. (2004) evaluate 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