Lecture 2: Data structures and Indexing

Information Retrieval
Computer Science Tripos Part II

Helen Yannakoudakis[1]

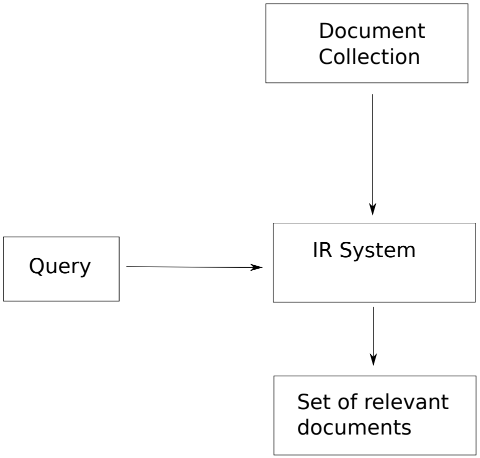
Natural Language and Information Processing (NLIP) Group
UNIVERSITY OF CAMBRIDGE

helen.yannakoudakis@cl.cam.ac.uk

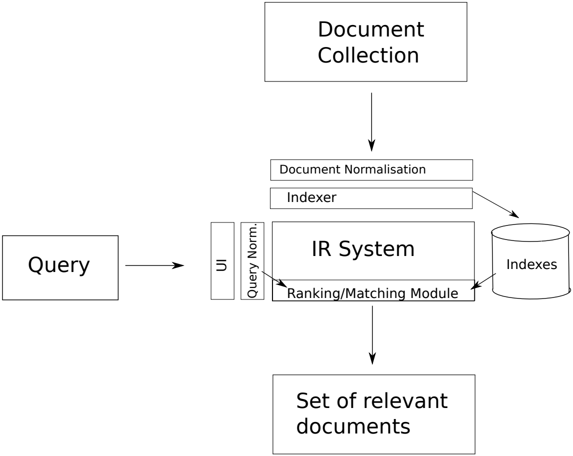
2018

[1] Based on slides from Simone Teufel and Ronan Cummins

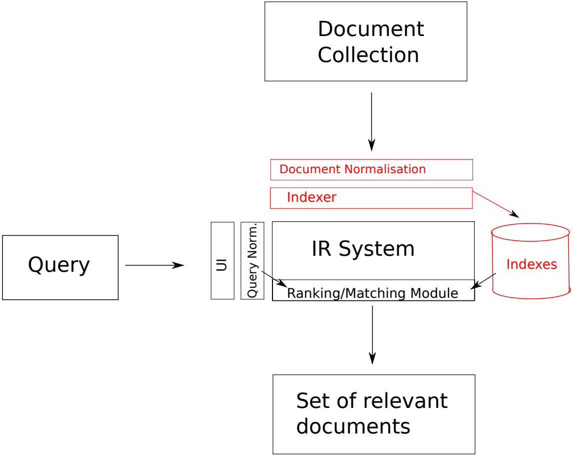
IR System Components



IR System Components



IR System Components



Today: The indexer

Definitions

So far, we've been talking about words...

- We call any unique word a type (*the* is a word type)
- We call an instance of a type a token (e.g., 13721 *the* tokens in Moby Dick)
- We call the type that is included in the IR system's dictionary a term (usually a "normalised" type – e.g., case, morphology, spelling etc.)

Consider the document to be indexed:
to sleep perchance to dream

Here we have *5 tokens*, *4 types*, *3 terms* (latter if we choose to omit *to* from the index).

Index construction

The major steps in inverted index construction:

- Collect the documents to be indexed.
- Tokenize the text.
- Perform linguistic pre-processing of tokens.
- Index the documents that each term occurs in.

Overview

1. Data structures and indexing
   - Posting lists and skip lists
   - Positional indexes
2. Documents, Terms, and Normalisation
   - Documents
   - Terms
   - Reuter RCV1 and Heap's Law

Example: index creation by sorting

Doc 1:
I did enact Julius Caesar: I was killed i' the Capitol;Brutus killed me.

⟹ Tokenisation

Doc 2:
So let it be with Caesar. The noble Brutus hath told you Caesar was ambitious.

⟹ Tokenisation

Term	docID
I	1
did	1
enact	1
julius	1
caesar	1
I	1
was	1
killed	1
i'	1
the	1
capitol	1
brutus	1
killed	1
me	1
so	2
let	2
it	2
be	2
with	2
caesar	2
the	2
noble	2
brutus	2
hath	2
told	2
you	2
caesar	2
was	2
ambitious	2

⟹ Sorting

Term (sorted)	docID
ambitious	2
be	2
brutus	1
brutus	2
capitol	2
caesar	1
caesar	2
caesar	2
did	1
enact	1
hath	1
I	1
I	1
i'	1
it	2
julius	1
killed	1
killed	2
let	2
me	1
noble	2
so	2
the	1
the	2
told	2
you	2
was	1
was	1
with	2

Index creation; grouping step ("uniq")

Term	doc. freq.		Postings list
ambitious	1	→	2
be	1	→	2
brutus	2	→	1 → 2
capitol	1	→	1
caesar	2	→	1 → 2
did	1	→	1
enact	1	→	1
hath	1	→	2
I	1	→	1
i'	1	→	1
it	1	→	2
julius	1	→	1
killed	1	→	1
let	1	→	2
me	1	→	1
noble	1	→	2
so	1	→	2
the	2	→	1 → 2
told	1	→	2
you	1	→	2
was	2	→	1 → 2
with	1	→	2

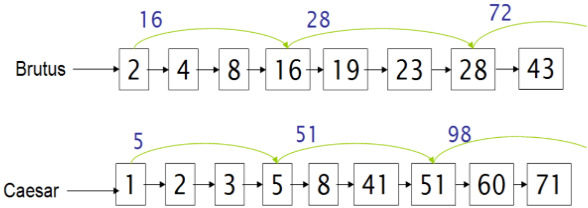
- Primary sort by term (dictionary)
- Secondary sort (within postings list) by document ID
- Document frequency (= length of postings list):
  - for more efficient Boolean searching
  - for term weighting (lecture 4)
- keep Dictionary in memory
- Postings List (much larger) traditionally on disk

Data structures for Postings Lists

Need variable-size postings lists:

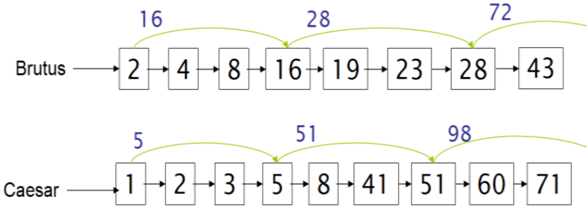
- On disk:
  - store as contiguous block without explicit pointers
  - minimises the size of postings lists and number of disk seeks
- In memory:
  - Linked list
    - Allow cheap insertion of documents into postings lists (e.g., when re-crawling)
    - Naturally extend to *skip lists* for faster access (skip pointers / shortcuts to avoid processing unnecessary parts of the postings list)
  - Variable length array
    - Better in terms of space requirements (no pointers)
    - Also better in terms of time requirements if memory caches are used, as they use contiguous memory

Optimisation: Skip Lists



- Recall basic algorithm
- More efficient way?
- Yes (given that index doesn't change too fast)
- Augment postings lists with skip pointers (at indexing time)
- If skip-list pointer present, skip multiple entries
  - E.g., after we match 8, $16 < 41$: skip to item after skip pointer
- Heuristic: for postings lists of length L, use $\sqrt{L}$ evenly-spaced skip pointers

Tradeoff Skip Lists



- Number of items skipped vs. frequency that skip can be taken
- More skips: each pointer skips only a few items, but we can frequently use it, but many comparisons.
- Fewer skips: each skip pointer skips many items, but we can not use it very often, but fewer comparisons.
- Skip pointers used to help a lot, but with modern harware, they may not.

Phrase Queries

- We want to answer a query such as [cambridge university] – as a phrase.
- The Duke of Cambridge recently went for a term-long course to a famous university should not be a match
- About 10% of web queries are phrase queries (double-quotes syntax).
- Consequence for inverted indexes: no longer sufficient to store docIDs in postings lists.
- Two ways of extending the inverted index:
  - biword index
  - positional index

Biword indexes

- Index every consecutive pair of terms in the text as a phrase.

Friends, Romans, Countrymen

Generates two biwords:

- friends romans
- romans countrymen

- Each of these biwords is now a dictionary term.
- Two-word phrases can now easily be answered.

Longer phrase queries

- A long phrase like cambridge university west campus can be broken into the Boolean query

cambridge university AND university west AND west campus

- False positives – we need to do post-filtering of hits to identify subset that actually contains the 4-word phrase.

Issues with biword indexes

- Why are biword indexes rarely used?
- False positives, as noted above
- Index blowup due to very large dictionary / vocabulary
    - Searches for a single term?
    - Infeasible for more than bigrams

Positional indexes

- Positional indexes are a more efficient alternative to biword indexes.
- Postings lists in a non-positional index: each posting is just a docID
- Postings lists in a positional index: each posting is a docID and a list of positions (offsets)

Positional indexes: Example

Query: "to be or not to be"

to, 993427:
< 1: < 7, 18, 33, 72, 86, 231>;
2: <1, 17, 74, 222, 255>;
4: <8, 16, 190, 429, 433>;
5: <363, 367>;
7: <13, 23, 191>;
. . . . . . >

be, 178239:
< 1: < 17, 25>;
4: < 17, 191, 291, 430, 434>;
5: <14, 19, 101>;
. . . . . . >

Document 4 is a match – why?
(As always: term, doc freq, docid, offsets)

Proximity search

- We just saw how to use a positional index for phrase searches.
- We can also use it for proximity search.

employment /4 place

- Find all documents that contain employment and place within 4 words of each other.
- HIT: Employment agencies that place healthcare workers are seeing growth.
- NO HIT: Employment agencies that have learned to adapt now place healthcare workers.

Note that we want to return the actual matching positions, not just a list of documents.

Proximity intersection

```
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
```

Combination scheme

- Biword indexes and positional indexes can be profitably combined.
- Many biwords are extremely frequent: Michael Jackson, Britney Spears etc
- For these biwords, increased speed compared to positional postings intersection is substantial.
- Combination scheme: Include frequent biwords as vocabulary terms in the index. Do all other phrases by positional intersection.
- Williams et al. (2004) evaluate a more sophisticated mixed indexing scheme. Faster than a positional index, at a cost of 26% more space for index.
- For web search engines, positional queries are much more expensive than regular Boolean queries.

Overview

1. Data structures and indexing
   - Posting lists and skip lists
   - Positional indexes

2. Documents, Terms, and Normalisation
   - Documents
   - Terms
   - Reuter RCV1 and Heap's Law

Definitions – reminder

- We call any unique word a type (*the* is a word type)
- We call an instance of a type a token (e.g., 13721 *the* tokens in Moby Dick)
- We call the type that is included in the IR system's dictionary a term (usually a "normalised" type – e.g., case, morphology, spelling etc.)

Documents

- Up to now, to build an inverted index, we assumed that:
  - We know what a document is.
  - We can “machine-read” each document
  - Each token is a candidate for a postings entry.
- More complex in reality

Parsing a document

Convert byte sequence into a linear sequence of characters, but . . .

- We need to determine the correct character encoding
- We need to determine format to decode the byte sequence into a character sequence
  - MS word, zip, pdf, latex, xml (e.g., `&amp`). . .
- Each of these is a statistical classification problem
- Alternatively we can use heuristics

Language

Text is not just a linear sequence of characters (e.g., diacritics above and below letters in Arabic)

- What language is it in?
- Writing system conventions?
- Documents or their components can contain multiple languages/format; for instance a French email with a Spanish pdf attachment
- A single index usually contains terms of several languages

Indexing granularity

What is the *document* unit for indexing?

- a file in a folder?
- a file containing an email thread?
- an email?
- an email with 5 attachments?
- individual sentences?

- Answering the question "What is a document?" is not trivial
- Precision/recall tradeoff: smaller units raise precision, drop recall

Tokenisation

Given a character sequence (and a defined document unit), we now need to determine our tokens...

...but, what are the correct tokens to use?

Mr. O'Neill thinks that the boys' stories about Chile's capital aren't amusing.

neill	aren't
oneill	arent
o'neill	are n't
o' neill	aren t
o neill	
?	?

The choices determine which queries will match.

Tokenisation 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

Numbers

20/3/91
3/20/91
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.

Chinese: No Whitespace

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 1 8 岁生日。生日派对上，莎拉波娃露出了甜美的微笑。

- Need to perform word segmentation
- Use a lexicon or supervised machine-learning

Chinese: Ambiguous segmentation

和尚

- As one word, means “monk”
- As two words, means “and” and “still”

Other cases of “no whitespace”: Compounding

Compounding in Dutch, German, Swedish

German

Lebensversicherungsgesellschaftsangestellter
leben+s+versicherung+s+gesellschaft+s+angestellter

Other cases of “no whitespace”: Agglutination

“Agglutinative” languages do this not just for compounds:

Inuit

tusaatsiarunnangittualuujunga
(= “I can’t hear very well”)

Finnish

epäjärjestelmällistyttämättömyydellänsäkäänköhän
(= “I wonder if – even with his/her quality of not having been made unsystematized”)

Turkish

Çekoslovakyalılaştıramadıklarımızdanmşçasına
(= “as if you were one of those whom we could not make resemble the Czechoslovacian people”)

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。

- Different scripts (alphabets) might be mixed in one language.
- Japanese has 4 scripts: kanja, katakana, hiragana, Romanji
- no spaces

Normalisation – equivalence classes

- Need to normalise tokens to get document–query matches
- Example: We want to match U.S.A. to USA
- We most commonly implicitly define equivalence classes of terms.
- Useful as searches for one term will retrieve documents that contain either.
- Advantage of using mapping rules is that the equivalence classing to be done is implicit

Alternative

- Alternatively, we could do asymmetric expansion where we maintain relations between un-normalized tokens.

  Example of asymmetric expansion of *query terms* that can usefully model users' expectations:

  window $\rightarrow$ window, windows
  windows $\rightarrow$ Windows, windows, window
  Windows $\rightarrow$ Windows

- Either at query time, or at index time
- Potentially more powerful, but less efficient than equivalence classing
  - e.g., query expansion dictionary and more processing at query-time

Normalisation: Accents and diacritics

- résumé vs. resume
- Universität
- Meaning-changing in some languages:

> peña = cliff, pena = sorrow (Spanish)

- Main question: will users apply it when querying?

Normalisation: Case Folding

- Reduce all letters to lower case
- Even though case can be semantically distinguishing

> Fed vs. fed
> March vs. march
> Turkey vs. turkey
> US vs. us

- Best to reduce to lowercase because users will use lowercase regardness of correct capitalisation.

Normalisation: More equivalence classing

- Thesauri: semantic equivalence, car = automobile
- Soundex: phonetic equivalence, Muller = Mueller; lecture 3

Lemmatisation

- Reduce inflectional/variant forms to base form

am, are, is → be
car, car's, cars', cars → car
the boy's cars are different colours → the boy car be different color

- Lemmatisation implies doing "proper" reduction to dictionary headword form (the lemma)
- Inflectional morphology (cutting → cut)
- vs. derivational morphology (destruction → destroy)

Stemming

- Stemming is a crude heuristic process that chops off the ends of words in the hope of achieving what "principled" lemmatisation attempts to do with a lot of linguistic knowledge.
- language-specific rules, but fast and space-efficient
- does not require a stem dictionary, only a suffix dictionary
- Often both inflectional and derivational

automate, automation, automatic → automat

- Root changes (deceive/deception, resume/resumption) aren't dealt with, but these are rare

Porter Stemmer

- M. Porter, “An algorithm for suffix stripping”, Program 14(3):130-137, 1980
- Most common algorithm for stemming English
- Results suggest it is at least as good as other stemmers
- Syllable-like shapes + 5 phases of reductions
- Phases are applied sequentially
- Each phase consists of a set of commands
- Of the rules in a compound command, select the top one and exit that compound (this rule will have affected the longest suffix possible, due to the ordering of the rules).

Stemming: Representation of a word

[C] (VC){m}[V]

C : one or more adjacent consonants
V : one or more adjacent vowels

[ ] : optionality
( ) : group operator
{**x**} : repetition x times
m : the "measure" of a word

shoe	$[sh]_C[oe]_V$	m=0
Mississippi	$[M]_C([i]_V[ss]_C)([i]_V[ss]_C)([i]_V[pp]_C)[i]_V$	m=3
ears	$([ea]_V[rs]_C)$	m=1

Notation: measure *m* is calculated on the word **excluding** the suffix of the rule under consideration

Porter stemmer: selected rules

SSES $\rightarrow$ SS
IES $\rightarrow$ I
SS $\rightarrow$ SS
S $\rightarrow$ $\emptyset$

caresses $\rightarrow$ caress
cares $\rightarrow$ care

(m>0) EED $\rightarrow$ EE

feed $\rightarrow$ feed
agreed $\rightarrow$ agree
BUT: freed, succeed

Porter Stemmer: selected rules

$$(*V*)\ \mathrm{ED} \rightarrow \emptyset$$

plastered $\rightarrow$ plaster
bled $\rightarrow$ bled

Three stemmers: a comparison

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 a 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 it for others.

Example queries where stemming helps

tartan sweaters → sweater, sweaters
sightseeing tour san francisco → tour, tours

Example queries where stemming hurts

operational research → "oper" = operates, operatives, operate, operation, operational, operative

operating system → operates, operatives, operate, operation, operational, operative

operative dentistry → operates, operatives, operate, operation, operational, operative

Stop words

- Extremely common words which are of little value in helping select documents matching a user need

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

- Used to be standard in older IR systems.
- Need them to search for

to be or not to be
prince of Denmark
bamboo in water

- Length of practically used stoplists has shrunk over the years.
- Most web search engines do index stop words.

Reuters RCV1 collection

- Shakespeare's collected works are not large enough to demonstrate scalable index construction algorithms.
- Instead, we will use the Reuters RCV1 collection.
- English newswire articles published in a 12-month period (1995/6)

N	documents	800,000
M	terms	400,000
T	tokens	100,000,000

Effect of pre-processing for Reuters

size of	terms dictionary			non-positional postings non-positional index			positional postings (word tokens) positional index		
	size	Δ	cml	size	Δ	cml	size	Δ	cml
unfiltered	484,494			109,971,179			197,879,290		
no numbers	473,723	-2	-2	100,680,242	-8	-8	179,158,204	-9	-9
case folding	391,523	-17	-19	96,969,056	-3	-12	179,158,204	-0	-9
30 stopw's	391,493	-0	-19	83,390,443	-14	-24	121,857,825	-31	-38
150 stopw's	391,373	-0	-19	67,001,847	-30	-39	94,516,599	-47	-52
stemming	322,383	-17	-33	63,812,300	-4	-42	94,516,599	-0	-52

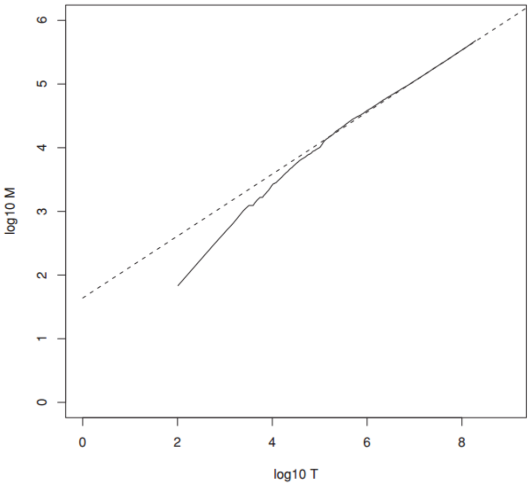
Δ: reduction in size from the previous line.[2]
cml: cumulative reduction from "unfiltered".

[2] Except for 30 and 150 stopw's that use "case folding" as their reference line.

How big is the vocabulary?

- That is, how many terms are there?
- Can we assume there is an upper bound?
- Not really: At least $70^{20} \approx 10^{37}$ different words of length 20.
- Vocabulary size M will keep growing with collection size.
- Heaps' law: $M = kT^b$
  - T is the number of tokens in the collection. Typical values for the parameters k and b are: $30 \leq k \leq 100$ and $b \approx 0.5$.
  - Dictionary size continues to increase with more documents
  - Dictionary size is quite large for large collections
- Heaps' law is linear in log–log space.
  - It is the simplest possible relationship between collection size and vocabulary size in log–log space.
  - Empirical law

Heaps' law for Reuters



Vocabulary size M as a function of collection size T (number of tokens) for Reuters-RCV1. For these data, the dashed line $\log_{10} M = 0.49 * \log_{10} T + 1.64$ is the best least squares fit. Thus, $M = 10^{1.64} T^{0.49}$ and $k = 10^{1.64} \approx 44$ and $b = 0.49$.

Empirical fit for Reuters

- Good, as we just saw in the graph.
- Example: for the first 1,000,020 tokens, Heaps' law predicts 38,323 terms:

$$44 \times 1{,}000{,}020^{0.49} \approx 38{,}323$$

- The actual number is 38,365 terms, very close to the prediction.
- Empirical observation: fit is good in general.

Take-away

- More complex indexes for phrases
- Understanding of the basic unit of classical information retrieval systems: terms and documents: What is a document, what is a term?
- Tokenization: how to get from raw text to terms (or tokens)
- Normalisation and equivalence classes

Reading

- MRS Chapter 2.2
- MRS Chapter 2.3
- MRS Chapter 2.4
- MRS Chapter 4.3