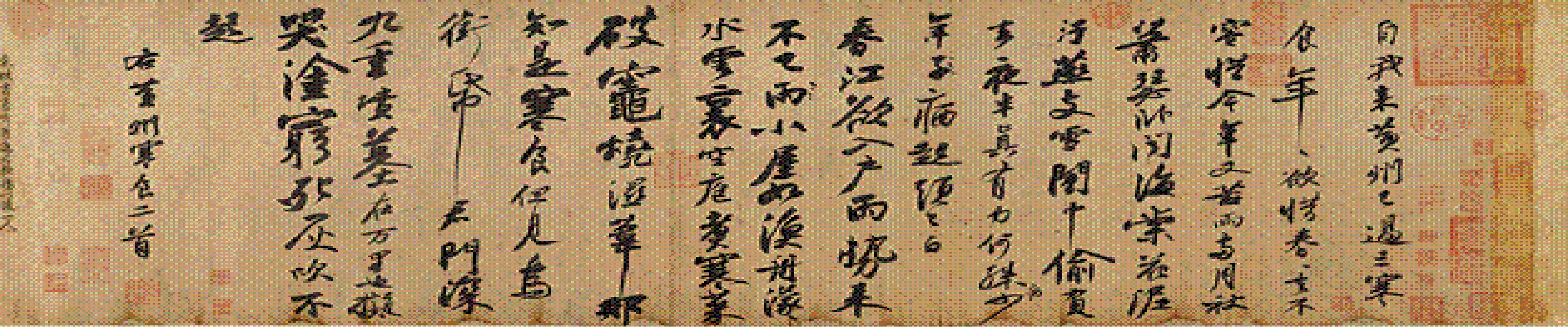


信息检索
Information Retrieval

教师：孙茂松

Tel:62781286

Email:sms@tsinghua.edu.cn

TA：胡锦毅

Email:hu-jy21@mails.tsinghua.edu.cn

郑重声明

- 此课件仅供选修清华大学计算机系本科生课《信息检索》(40240372)的学生个人学习使用，所以只允许学生将其下载、存贮在自己的电脑中。未经孙茂松本人同意，任何人不得以任何方式扩散之。否则，由此可能引起的一切涉及知识产权的法律责任，概由该人负责。
- 此课件仅限孙茂松本人讲课使用。除孙茂松本人外，凡授课过程中，PTT文件显示此《郑重声明》之情形，即为侵权使用。

第二章 信息检索系统的基本框架（Part 3）

2.4 对倒排文件的进一步考察

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

Term	N docs	Tot Freq
ambitious	1	1
be	1	1
brutus	2	2
capitol	1	1
caesar	2	3
did	1	1
enact	1	1
hath	1	1
I	1	2
i'	1	1
it	1	1
julius	1	1
killed	1	2
let	1	1
me	1	1
noble	1	1
so	1	1
the	2	2
told	1	1
you	1	1
was	2	2
with	1	1

Doc #	Freq
2	1
2	1
1	1
2	1
1	1
1	1
2	2
1	1
1	1
2	1
1	2
1	1
2	1
1	1
1	2
2	1
1	1
2	1
2	1
1	1
2	1
2	1
2	1
1	1
2	1
2	1

The file is commonly split into a *Dictionary* and a *Postings List*

2.4 对倒排文件的进一步考察

For the Dictionary

- How big is the term vocabulary?

  That is, how many distinct words are there?
- In practice, the vocabulary will keep growing with the collection size

Vocabulary vs. collection size

- Heaps' law: $M = kT^b$
- M is the size of the vocabulary, T is the number of tokens in the collection
- Typical values: $30 \leq k \leq 100$ and $b \approx 0.5$
- In a log-log plot of vocabulary size M vs. T, Heaps' law predicts a line with slope about ½
  - It is the simplest possible relationship between the two in log-log space
  - An empirical finding ("empirical law")

Heaps' Law

For RCV1, the dashed line

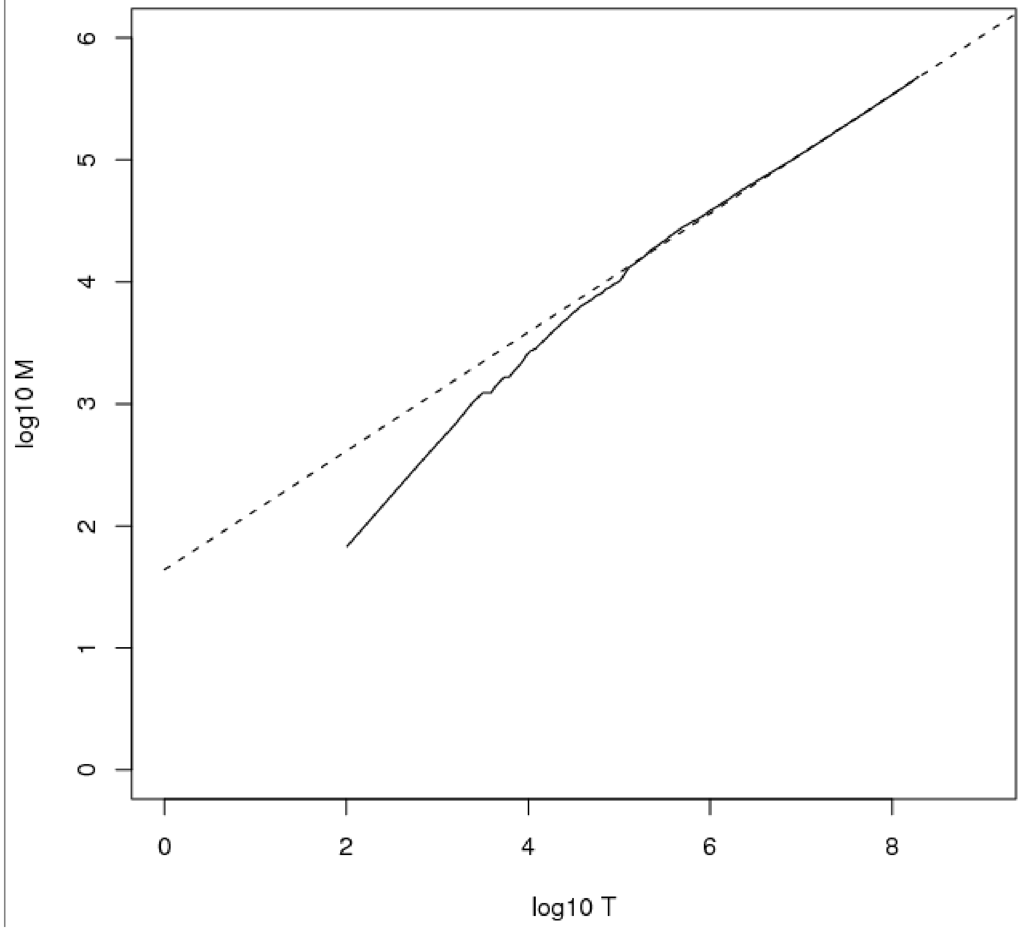
$\log_{10}M = 0.49 \log_{10}T + 1.64$

is the best least squares fit.

Thus, $M = 10^{1.64}T^{0.49}$ so $k = 10^{1.64} \approx 44$ and $b = 0.49$.

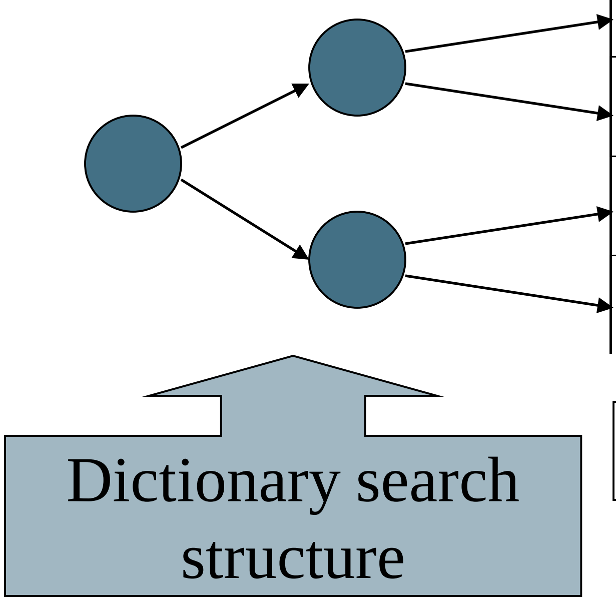
Good empirical fit for Reuters RCV1 !

For first 1,000,020 tokens, law predicts 38,323 terms; actually, 38,365 terms



Dictionary storage - first cut

- Array of fixed-width entries
  - ~400,000 terms; 28 bytes/term = 11.2 MB.



Terms	Freq.	Postings ptr.
a	656,265	
aachen	65	
….	….	
zulu	221	

20 bytes

4 bytes each

Fixed-width terms are wasteful

- Most of the bytes in the **Term** column are wasted – we allot 20 bytes for 1 letter terms.
  - And we still can't handle ***supercalifragilisticexpialidocious*** or ***hydrochlorofluorocarbons***.
- Written English averages ~4.5 characters/word.
  - Exercise: Why is/isn't this the number to use for estimating the dictionary size?
- Ave. dictionary word in English: ~8 characters
  - How do we use ~8 characters per dictionary term?
- Short words dominate token counts but not type average.

Compressing the term list: Dictionary-as-a-String

- Store dictionary as a (long) string of characters:
  - Pointer to next word shows end of current word
  - Hope to save up to 60% of dictionary space.

….systilesyzygeticsyzygialsyzygyszaibelyiteszczecinszomo….

Freq.	Postings ptr.	Term ptr.
33		
29		
44		
126		

Total string length = 400K x 8B = 3.2MB

Pointers resolve 3.2M positions: $\log_2 3.2M$ = 22bits = 3bytes

Space for dictionary as a string

- 4 bytes per term for Freq.
- 4 bytes per term for pointer to Postings.

- 3 bytes per term pointer
- Avg. 8 bytes per term in term string

Now avg. 11 bytes/term, not 20.

- 400K terms x 19 $\Rightarrow$ 7.6 MB (against 11.2MB for fixed width)

2.4 对倒排文件的进一步考察

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

Term	N docs	Tot Freq
ambitious	1	1
be	1	1
brutus	2	2
capitol	1	1
caesar	2	3
did	1	1
enact	1	1
hath	1	1
I	1	2
i'	1	1
it	1	1
julius	1	1
killed	1	2
let	1	1
me	1	1
noble	1	1
so	1	1
the	2	2
told	1	1
you	1	1
was	2	2
with	1	1

Doc #	Freq
2	1
2	1
1	1
2	1
1	1
1	1
2	2
1	1
1	1
2	1
1	2
1	1
2	1
1	1
1	2
2	1
1	1
2	1
2	1
1	1
2	1
2	1
2	1
1	1
2	1
2	1

The file is commonly split into a *Dictionary* and a *Postings List*

2.4 对倒排文件的进一步考察

For the postings file

- much larger than the dictionary
- A posting is a docID.
- For a document collection (1M documents), we would use 32 bits per docID when using 4-byte integers.
- Alternatively, we can use $\log_2 1M \approx 20$ bits per docID.
- Our goal: use a lot less than 20 bits per docID.

Postings: two conflicting forces

- A term like ***arachnocentric*** occurs in maybe one doc out of a million – we would like to store this posting using $\log_2 1M \sim 20$ bits.
- A term like ***the*** occurs in virtually every doc, so 20 bits/posting is too expensive.

Postings file entry

- We store the list of docs containing a term in increasing order of docID.
  - ***computer***: 33,47,154,159,202 …
- Consequence: it suffices to store *gaps*.
  - 33,14,107,5,43 …
- Hope: most gaps can be encoded/stored with far fewer than 20 bits.

Three postings entries

	encoding	postings list									
THE	docIDs	...		283042		283043		283044		283045	...
	gaps				1		1		1		...
COMPUTER	docIDs	...		283047		283154		283159		283202	...
	gaps				107		5		43		...
ARACHNOCENTRIC	docIDs	252000		500100							
	gaps	252000	248100								

Variable length encoding

- Aim:
  - For ***arachnocentric***, we will use ~20 bits/gap entry.
  - For ***the***, we will use ~1 bit/gap entry.
- If the average gap for a term is *G*, we want to use ~$\log_2 G$ bits/gap entry.
- Key challenge: encode every integer (gap) with about as few bits as needed for that integer.
- This requires a *variable length encoding*
- Variable length codes achieve this by using short codes for small numbers

Uniquely Decodable Codes

A **variable length code** assigns a bit string (codeword) of variable length to every message value

e.g. `a = 1, b = 01, c = 101, d = 011`

What if you get the sequence of bits `1011` ?

Is it `aba, ca,` or, `ad`?

A **uniquely decodable code** is a variable length code in which bit strings can always be uniquely decomposed into its codewords.

Variable Byte (VB) codes

- For a gap value *G*, we want to use close to the fewest bytes needed to hold $\log_2 G$ bits
- Begin with one byte to store *G* and dedicate 1 bit in it to be a continuation bit *c*
- If $G \leq 127$, binary-encode it in the 7 available bits and set $c = 1$
- Else encode *G*'s lower-order 7 bits and then use additional bytes to encode the higher order bits using the same algorithm
- At the end set the continuation bit of the last byte to 1 ($c = 1$), and for the other bytes $c = 0$.

Example

docIDs	824	829	215406
gaps		5	214577
VB code	00000110 10111000	10000101	00001101 00001100 10110001

Postings stored as the byte concatenation
000001101011100010000101000011010000110010110001

Key property: VB-encoded postings are uniquely decodable codes.

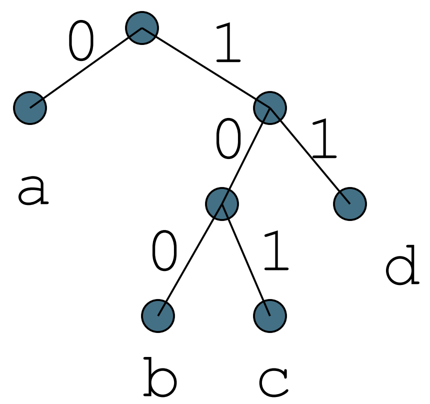
For a small gap (5), VB uses a whole byte.

Prefix Codes (Bit-based)

A **prefix code** is a variable length code in which no codeword is a prefix of another word

e.g a = 0, b = 110, c = 111, d = 10

Can be viewed as a binary tree with message values at the leaves and 0 or 1s on the edges.



Average Length

For a code *C* with associated probabilities *p(c)* the **average length** is defined as

$$l_a(C) = \sum_{c \in C} p(c) l(c)$$

We say that a prefix code *C* is **optimal** if for all prefix codes *C'*, $l_a(C) \leq l_a(C')$

$$l_a(C)\,?$$

Entropy (Shannon 1948)

For a set of messages *S* with probability *p(s), s* $\in$*S*, the **self information** of *s* is:

$$i(s) = \log \frac{1}{p(s)} = -\log p(s)$$

Measured in bits if the log is base 2.

The lower the probability, the higher the information

Entropy is the weighted average of self information.

$$H(S) = \sum_{s \in S} p(s) \log \frac{1}{p(s)}$$

Self-Information vs. $H(S)$

s	a	b	c	d

p	.25	.25	.25	.25
$i(s)$	2	2	2	2

$H(S)=2$

p	.7	.1	.1	.1
$i(s)$	.51	3.32	3.32	3.32

$H(S)=1.353$

Self-Information vs. $H(S)$

- Greater frequency <==> Less information
- Extreme case: $p(s_x) = 1$, $H(S) = 1*\lg(1) = 0$
- Why is H(S) a right formula?
- $1/p$ is the *average length of the gaps between recurrences of s*

.....*s*......*s*...........................*s*.........*s*........*s*.....

a *b* *c* *d*

Average of *a, b, c, d* … = $1/p$

Number of bits to specify a gap is about $\lg(1/p)$

Relationship to Entropy

Theorem (lower bound): *For any probability distribution p(S) with associated uniquely decodable code C,*

$$H(S) \leq l_a(C)$$

Theorem (upper bound): *For any probability distribution p(S) with associated optimal prefix code C,*

$$l_a(C) \leq H(S) + 1$$

Entropy and Compression

a	b	c	d
.7	.1	.1	.1
0	10	110	111

- No code taking only frequencies into account can be better than this entropy
- Average length for the code $=.7\cdot1+.1\cdot2+.1\cdot3+.1\cdot3 = 1.5$
- Entropy $= .7\cdot\lg(1/.7)+.1\cdot\lg(1/.1)+ .1\cdot\lg(1/.1)+.1\cdot\lg(1/.1) = 1.353$
- Lower entropy <=> More redundant <=> More compressible
- Higher entropy <=> Less redundant <=> Less compressible

2.4 对倒排文件的进一步考察

Blocking

Other Indices for Text

Suffix Trees and Suffix Arrays

Pat trees

Signature Files

……

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; ...

Positional index size

- You can compress position values/offsets
- Nevertheless, a positional index expands postings storage *substantially*
- Nevertheless, a positional index is now standardly used.

Lossless vs. lossy compression

- Lossless compression: All information is preserved.
  - What we mostly do in IR.
- Lossy compression: Discard some information
- Several of the preprocessing steps can be viewed as lossy compression: case folding, stop words, stemming, number elimination.

Reuters RCV1

symbol	statistic	value
N	documents	800,000
L	avg. # tokens per doc	200
M	terms (= word types)	~400,000
	avg. # bytes per token (incl. spaces/punct.)	6
	avg. # bytes per token (without spaces/punct.)	4.5
	avg. # bytes per term	7.5
	non-positional postings	100,000,000

Reuters RCV1

size of	word types (terms)			non-positional postings			positional postings		
	dictionary			non-positional index			positional index		
	Size (K)	∆%	cumul %	Size (K)	∆ %	cumul %	Size (K)	∆ %	cumul %
Unfiltered	484			109,971			197,879		
No numbers	474	-2	-2	100,680	-8	-8	179,158	-9	-9
Case folding	392	-17	-19	96,969	-3	-12	179,158	0	-9
30 stopwords	391	-0	-19	83,390	-14	-24	121,858	-31	-38
150 stopwords	391	-0	-19	67,002	-30	-39	94,517	-47	-52
stemming	322	-17	-33	63,812	-4	-42	94,517	0	-52