Introduction to Information Retrieval

Index Compression (Posting)

Postings Compression

Postings compression

- The postings file is much larger than the dictionary, factor of at least 10.
- Key desideratum: store each posting compactly.
- A posting for our purposes is a docID.
- For Reuters (800,000 documents), we would use 32 bits per docID when using 4-byte integers.
- Alternatively, we can use log2 800,000 ≈ 20 bits per docID.
- Our goal: use far fewer 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 log2 1M ~ 20 bits.
- A term like **the** occurs in virtually every doc, so 20 bits/posting is too expensive.
    - Prefer 0/1 bitmap vector in this case

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

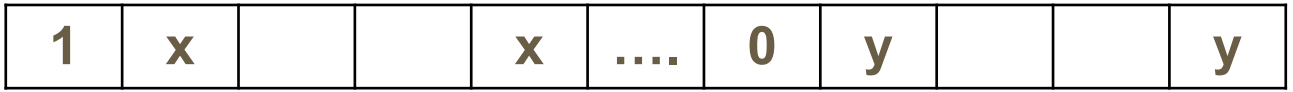
- Aim:
  - For **arachnocentric**, we will use ~20 bits/gap entry.
  - For **the**, we will use ~1 bit/gap entry.
- If the average gap for a term is G, we want to use ~log2G bits/gap entry.
- Key challenge: encode every integer (gap) with about as few bits as needed for that integer.
- This requires a **variable length encoding**
- Variable length codes achieve this by using short codes for small numbers

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

Variable Byte (VB) codes

- If G ≤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 824 = 1100111000 = **00000**110 10111000

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 prefix-decodable.

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

- Convert to Binary.
- Group the bits in 7s
- For LSB put a 1 in the 8^{th} bit
-
- For all other put a 0

- 156781 = 100110010001101101
- 00001001 01001000 11101101
- Waste – 4 bits. - this 4 bits is not a big deal when the number is big but big wastage when the number is small.

Other variable unit codes

- Instead of bytes, we can also use a different “unit of alignment”: 32 bits (words), 16 bits, 4 bits (nibbles).
- Variable byte alignment wastes space if you have many small gaps – nibbles do better in such cases.
- Variable byte codes:
  - Used by many commercial/research systems
  - Good low-tech blend of variable-length coding and sensitivity **to computer memory alignment matches** (vs. bit-level codes, which we look at next).

Unary code

- Represent n as n 1s with a final 0.
- Unary code for 3 is 1110.
- Unary code for 40 is

1111111111111111111111111111111111111110 .

- Unary code for 80 is:

11111111111111111111111111111111111111111111111111111111111111111111111111111110

- This doesn’t look promising, but….

Gamma codes

- We can compress better with bit-level codes
    - The Gamma code is the best known of these.
- Represent a gap G as a pair length and offset
- Offset is G in binary, with the leading bit cut off
    - For example 13 → 1101 → 101

Gamma codes

- length is the length of offset
    - For 13 (offset 101), this is 3.
- We encode length with unary code: 1110.
- Gamma code of 13 is the concatenation of length and offset: 1110101

Gamma code examples – G – $2\log_2(G)$

number	length	offset	γ-code
0			none
1	0		0
2	10	0	10,0
3	10	1	10,1
4	110	00	110,00
9	1110	001	1110,001
13	1110	101	1110,101
24	11110	1000	11110,1000
511	111111110	11111111	111111110,11111111
1025	11111111110	0000000001	11111111110,0000000001

- 111101010110111110110 – Can I tell how many numbers are there, if I know that this is encode by gamma code

- Gamma code is taking almost similar space to a normal code but still you are getting the decipherable property.

Gamma code properties

- G is encoded using 2 | log G | + 1 bits
    - Length of offset is | log G | bits
    - Length of length is | log G | + 1 bits
- All gamma codes have an odd number of bits
- Almost within a factor of 2 of best possible, $\log_2 G$

Gamma code properties

- Gamma code is uniquely prefix-decodable, like VB
- Gamma code can be used for any distribution
- Gamma code is parameter-free

Gamma seldom used in practice

Gamma seldom used in practice

- Machines have word boundaries – 8, 16, 32, 64 bits
- Operations that cross word boundaries are slower
- Compressing and manipulating at the granularity of bits can be slow
- Variable byte encoding is aligned and thus potentially more efficient
- Regardless of efficiency, variable byte is conceptually simpler at little additional space cost

RCV1 compression

Data structure	Size in MB
dictionary, fixed-width	11.2
dictionary, term pointers into string	7.6
with blocking, k = 4	7.1
with blocking & front coding	5.9
collection (text, xml markup etc)	3,600.0
collection (text)	960.0
Term-doc incidence matrix	40,000.0
postings, uncompressed (32-bit words)	400.0
postings, uncompressed (20 bits)	250.0
postings, variable byte encoded	116.0
postings, γ–encoded	101.0

Index compression summary

- We can now create an index for highly efficient Boolean retrieval that is very space efficient
- Only 4% of the total size of the collection
- Only 10-15% of the total size of the text in the collection
- However, we've ignored positional information
- Hence, space savings are less for indexes used in practice
    - But techniques substantially the same.

Index parameters vs. what we index

(docid, number of times the term, pos1, pos2)

size of	word types (terms)			non-positional postings			positional postings		
	dictionary			non-positional index			positional index		
	Size (K)	Δ%	cumul %	Size (K)	Δ %	cumul %	Size (K)	Δ %	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

Exercise: give intuitions for all the '0' entries. Why do some zero entries correspond to big deltas in other columns?

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.
- Prune postings entries that are unlikely to turn up in the top k list for any query.
  - Almost no loss quality for top k list.

Vocabulary vs. collection size

- How big is the term vocabulary?
  - That is, how many distinct words are there?
- Can we assume an upper bound?
  - Not really: At least 7020 = 1037 different words of length 20
- In practice, the vocabulary will keep growing with the collection size
  - Especially with Unicode

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 $b \approx 0.5$ T = 100 M = o(10), T = 10000, M= o(100)
- 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

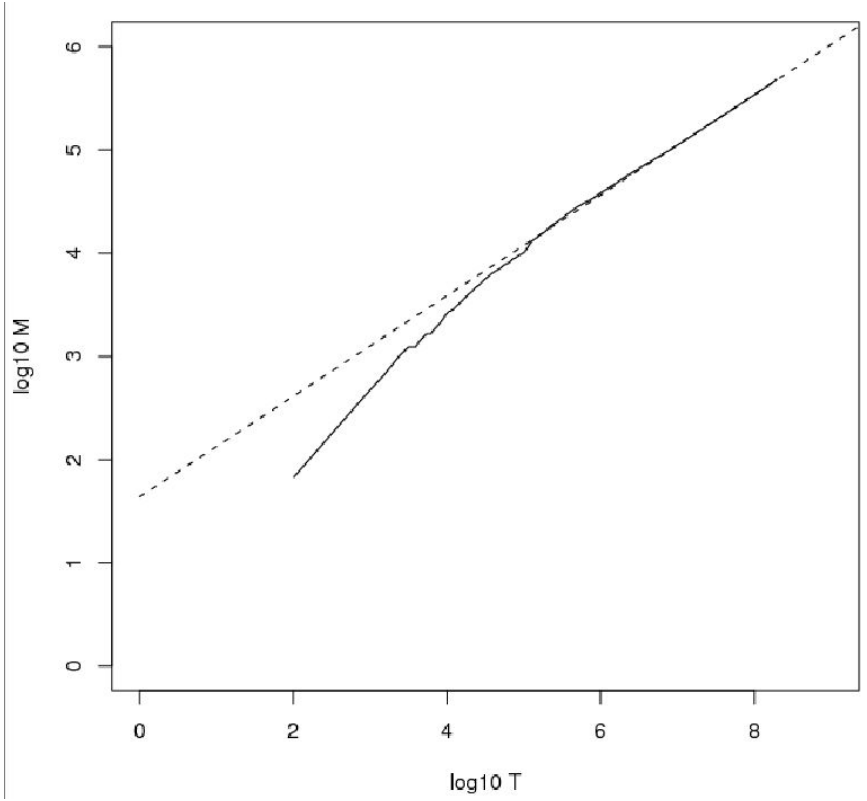
For RCV1, the dashed line $\log_{10}M = 0.49 \log_{10}T + 1.64$ is the best least squares fit.

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

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



Zipf's law

- Heaps' law gives the vocabulary size in collections.
- We also study the relative frequencies of terms.
- In natural language, there are a few very frequent terms and very many very rare terms.
- Zipf's law: The ith most frequent term has frequency proportional to 1/i .

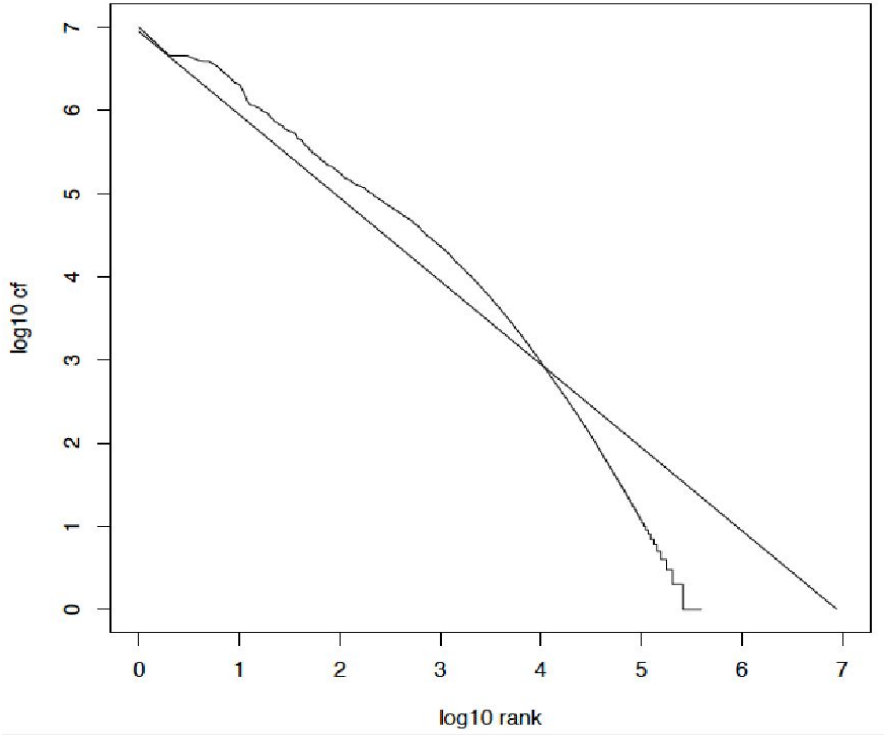
  cfi ∝ 1/i = K/i

  where K is a normalizing constant
- cfi is collection frequency: the number of occurrences of the term ti in the collection. 100, 50, 33, 25, 20, 17

Zipf consequences

- If the most frequent term (the) occurs cf1 times
  - then the second most frequent term (of) occurs cf1/2 times
  - the third most frequent term (and) occurs cf1/3 times ...
- Equivalent: **cfi = K/i** where K is a normalizing factor, so
  - l**og cfi = log K - log i**
  - Linear relationship between log cfi and log i
- Another power law relationship

Zipf's law for Reuters RCV1



- Exponentially decaying
- Sum of the rare words is small
- ==========IR system of Taglaq ==========
- Limited number of words.
- Accuracy of a search system which tackles only popular words – accuracy of the search system which tackles popular as well as rare words = DELTA which is SMALL
- Cost of a search system which tackles only popular words – Cost of the search system which tackles popular as well as rare words = GAMMA which is LARGE

==========

- Zipf’s Law
- Sum of the rare words is not small
- ==========IR system of Taglaq ==========
- Limited number of words.
- Accuracy of a search system which tackles only popular words – accuracy of the search system which tackles popular as well as rare words = DELTA which is LARGE
- Cost of a search system which tackles only popular words – Cost of the search system which tackles popular as well as rare words = GAMMA which is LARGE

==========