Index compression

INVERTED INDEX

Inverted index consists of two principal components

- Dictionary
- Posting lists

UNCOMPRESSED INVERTED INDEX

Uncompressed inverted index for a given document can be very large

	Collection Size		Index Size	
	Uncompressed	Compressed	Uncompressed	Compressed
Shakespeare	7.5 MB	2.0 MB	10.5 MB (139%)	2.7 MB (36%)
TREC45	1904.5 MB	582.9 MB	2331.1 MB (122%)	533.0 MB (28%)
GOV2	425.8 GB	79.9 GB	328.3 GB (77%)	62.1 GB (15%)

COMPRESSED INVERTED INDEX

Advantages of compressed inverted index

- Less storage
- Fast query retrieval time
- can index large collections

GENERAL PURPOSE DATA COMPRESSION

Compression algorithm takes the data and converts into another data which requires fewer bits to store and transfer.

- Encoder: converts original data A to B

  sizeof(A) > sizeof(B)

- Decoder: takes B and converts into C
  - lossy: C can be approximation of A (JPEG,MP3)
  - lossless: C is exact copy of A

Symbolwise Data Compression

- Data compression techniques can treat the information(M) as sequence of symbols.
- Not all symbols in M appear with the same frequency
- Symbols can be depend on the previous symbols ex: ‘q’ and ‘u’

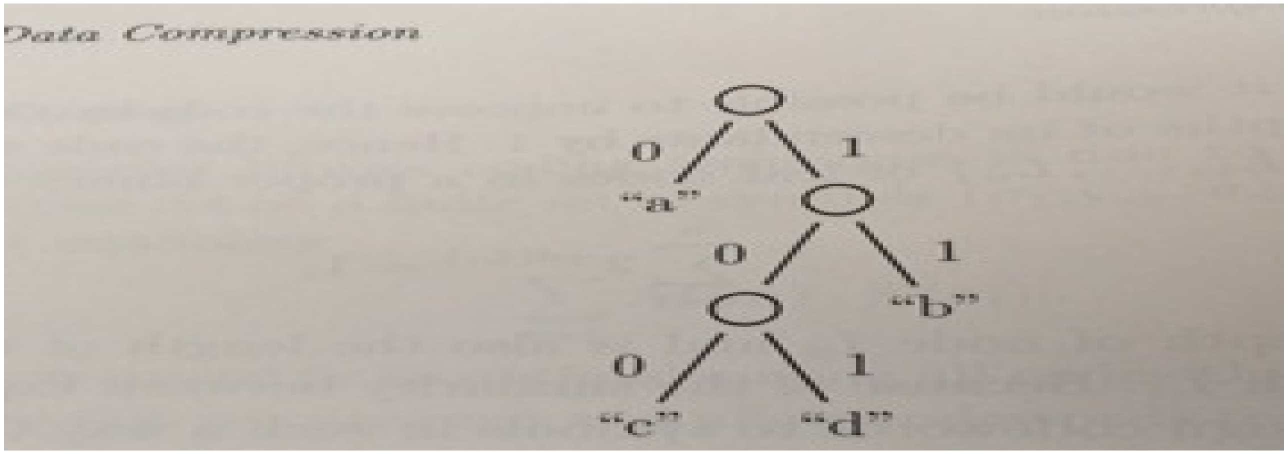
Modeling and Coding

Symbolwise compression methods work in two phases

- Modeling: a probability distribution M is computed that maps symbols to their probability of occurrence
- Coding: symbols in the message M are reencoded according to a code C

Ex: Huffman algorithm calculates probability of the frequency of characters and using that to find the code for each character

BITWISE CODING

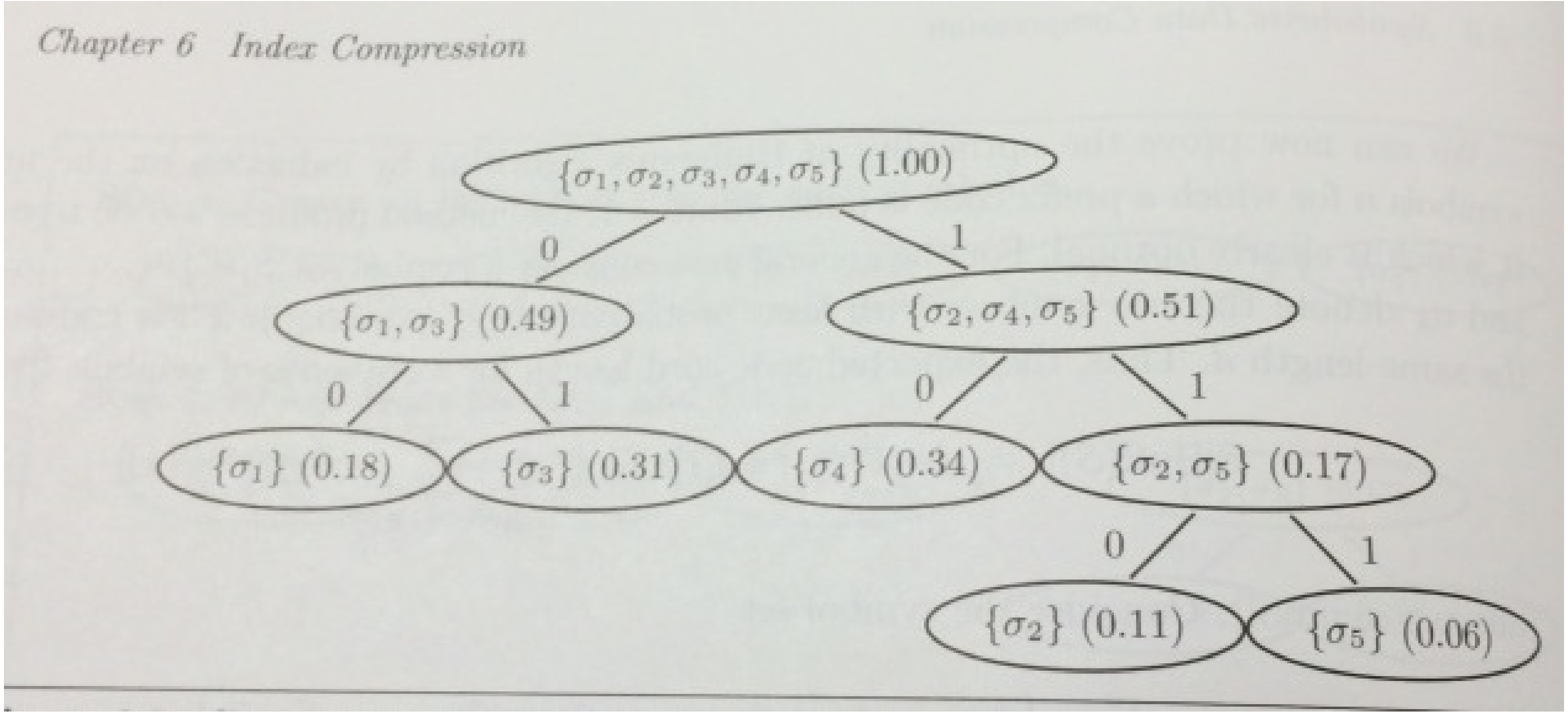


- Prefix property: no code word is an initial substring of any other code word
- a-0, b-11, c-100, d-101

HUFFMAN CODING ALGORITHM

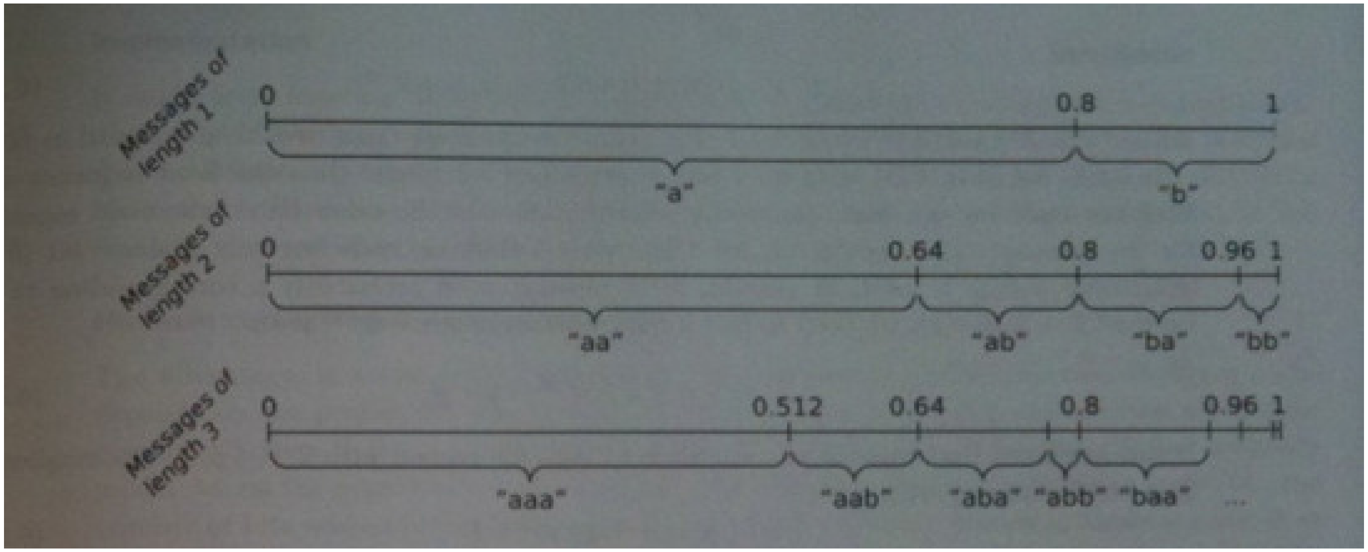
- Generate probability of occurrences of each character
- Create each individual node with the probability
- Find two minimum nodes and combined them into one node with sum of their probabilities
- Two minimum nodes become left and right nodes.
- Repeat it until ends with a single node

HUFFMAN CODING ALGORITHM



ARITHMETIC CODING

- improve upon the Huffman code for single symbols by taking pairs of symbols and making the Huffman code for them
- Ex: “aa”, “ab”, “ba”, “bb”,…



CONTD…

- Find the sub intervals of the sequences of symbols and then find its binary representation and encode the message using those
- Ex: aaa => [0,0.512) => 0, aab => [0.512, 0.64) => 0.10011, aba => [0.64,0.75) => 0.11
- Decodes as soon as it sees 0 to“aaa” .10 to “aab” and so on…

POSTING LISTS

- Majority of data in an inverted index are postings data.
- A posting list consists of a sequence of integers giving the doc id's of the document that contained a particular word.

  L= (3, 7, 11, 23, 29, 37, 41,…)
- List can be very large and each element occurs single time
- Standard compression methods like Huffman coding is not feasible

COMPRESSING POSTING LISTS: Δ-VALUES

Transformed into an equivalent sequence of difference between consecutive elements(Δ-values)
Δ (L) = (3, 4, 4, 12, 6, 8, 4, …)

- Elements are smaller and can be encoded using fewer bits.
- Elements can occur multiple times
- Nonparametric Gap Compression : does not consider the actual Δ-gap distribution (**γ** Codes)
- Parametric Gap Compression : conducts an analysis of some statistical properties of the list to be compressed (Golomb/Rice codes)

γ CODES

- Offsets in a posting list: (100000, 100005,100011,...)
- Gap compression: (1: 5, 6...)
- To compress these small numbers: We write (number of bits -1) we want in unary with 0's, followed by a 1, followed by the number in binary.
- Unary representation for 1=>0, 2=> 00, 3=>000….

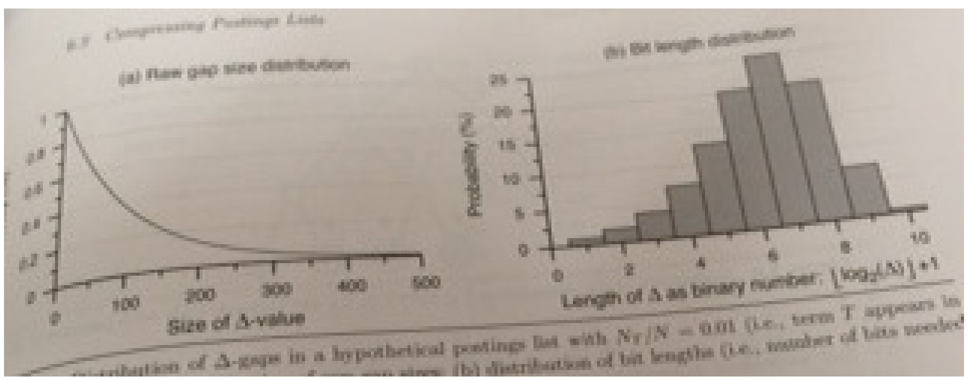
k	selector(k)	body(k)
1	1	1
5	001	101
7	001	111
16	00001	10000

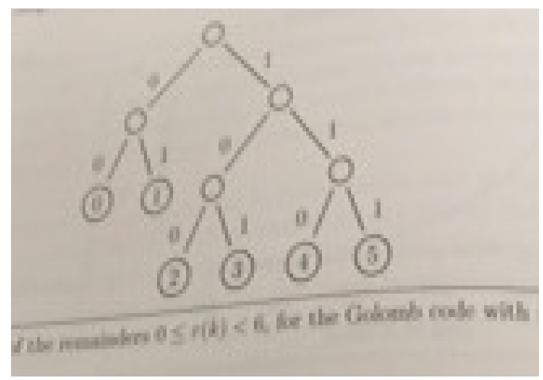
- High-order bit of the binary code for the number is redundant given that we known the length of the number, so we can drop this bit to get the actual encoding.

k	selector(k)	body(k)
1	1	
5	01 1	01
7	01 1	11
16	001 01	0000

Golomb/Rice codes

- Compress a list whose Δ-values follow a geometric distribution $Pr[\Delta=k]=(1-p)^{k-1}p$.
- Arbitary Modulus M (Golomb)
- M is a power of 2 (Rice)





- Determine an appropriate modulus
- Split each value into two components:
  - quotient q(k)
  - remainder r(k)

Where $q(k)=\lfloor k-1/M \rfloor, r(k)=(k-1) \bmod M$

Table 6.4 Encoding positive integers (e.g., Δ-gaps) using parameterized Golomb/Rice codes. The first part of each codeword corresponds to the quotient $q(k)$. The second part corresponds to the remainder $r(k)$.

Integer	Golomb Codes			Rice Codes	
	$M=3$	$M=6$	$M=7$	$M=4$	$M=8$
1	1 0	1 00	1 00	1 00	1 000
2	1 10	1 01	1 010	1 01	1 001
3	1 11	1 100	1 011	1 10	1 010
4	01 0	1 101	1 100	1 11	1 011
5	01 10	1 110	1 101	01 00	1 100
6	01 11	1 111	1 110	01 01	1 101
7	001 0	01 00	1 111	01 10	1 110
8	001 10	01 01	01 00	01 11	1 111
9	001 11	01 100	01 010	001 00	01 000
31	00000000001 0	000001 00	00001 011	00000001 10	0001 110

BYTE-ALIGNED CODES

- vByte(variable-byte coding): Splits the binary representation of each Δvalue into 7-bit chunk + 1 bit continuation flag

  L=(1624, 1650,1876,1972, …)

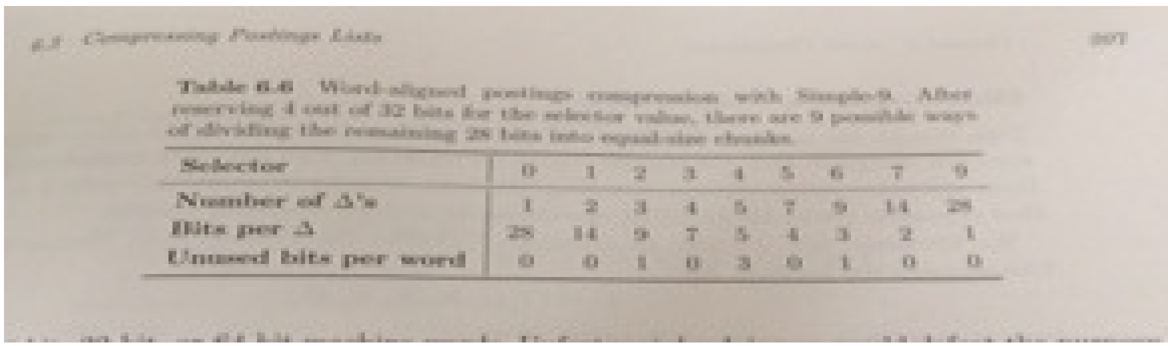
  Δ(L) = (1624, 26, 226, 96, 384, …)

  1 1011000 0 0001100 0 0011010 1 1100010 0 0000001 0 1100000 1 0000000 0 0000011…
- 0 at the beginning of the chunk indicates the end of the current code word. ($88+12 * 2^7 =1624$)

WORD-ALIGNED CODES(SIMPLE-9)

- Inspects Δvalues in a postings sequence and insert as many as possible into a 32-bit.
- Reserve 4 bits for selector

0001 00011100101100 00000000011001 0010 011100001 001011111 101111111 U



6.3 Compressing Postings Lists

Table 6.6 Word-aligned postings compression with Simple-9. After reserving 4 out of 32 bits for the selector value, there are 9 possible ways of dividing the remaining 28 bits into equal-size chunks.

Selector	0	1	2	3	4	5	6	7	8
Number of Δ's	1	2	3	4	5	7	9	14	28
Bits per Δ	28	14	9	7	5	4	3	2	1
Unused bits per word	0	0	1	0	3	0	1	0	0

References

- Stefan, B., Clarke , C., & Cormack, G. (2010). Information retrieval - Implementing and Evaluating Search Engines . Cambridge, Massachusetts: MIT Press.