Information Retrieval

WS 2017 / 2018

Lecture 4, Tuesday November 14th, 2017
(Compression, Codes, Entropy)

Prof. Dr. Hannah Bast
Chair of Algorithms and Data Structures
Department of Computer Science
University of Freiburg

Overview of this lecture

- Organizational
  - Your experiences with ES3 Efficient List Intersection
- Compression
  - Motivation saves space **and** query time
  - Codes Elias, Golomb, Variable-Byte
  - Entropy Shannon's famous theorem
  - **Exercise Sheet 4:** three nice proofs → part of Shannon's theorem + optimality of Golomb + size of inverted index

    We take a break from implementation work this week

Experiences with ES3 1/2

- Summary / excerpts
  - Very interesting topic and exercise (performance tuning)
  - First-time contact with Java or C++ for some
  - Large variations in runtimes between runs, especially in Java

    Reasons: JIT Compiler, Garbage Collection, Caching effects
  - It was surprisingly (for many) hard to beat the baseline

    "Whenever I tried an improvement I got a worse result"

    That is, in fact, an important lesson: it can be very hard to beat a well-implemented baseline, even with an algorithm that is asymptotically much better (but also more complex)
  - "The dude never actually bowls in the whole movie"

Experiences with ES3 2/2

Query film+bowling+rug:

The Big Lebowski

Fact "that rug really tied the room together"

mentioned four times

- Results
  - Three inverted lists of different lengths

film	1,983,510 postings
bowling	131,068 postings
rug	5,132 postings

  - Query **film+rug**, list length ratio **386**

    Any of galloping, skip ptrs, bin. search give large speedup
  - Query **bowling+rug**, list length ratio **25**

    Skipping helps, but not too much
  - Query **film+bowling**, list length ratio **15**

    Skipping costs more than it helps, switch to tuned baseline

Compression 1/6

- Motivation
  - Inverted lists can become very large

    Recall: the length of an inverted list of a word = total number of occurrences of that word in the collection

  - In a web-scale collection, the size can be millions, tens of millions, or hundreds of millions

    Just try a few single-keyword queries on Google and note the (estimated) number of results

algorithm	about 161,000,000 results
retrieval	about 661,000,000 results

  - Compression potentially saves space **and** time … the next slides will explain why

Compression 2/6

- Index in **memory**
  - Then compression saves memory (obviously)
  - Also: the index might be too large to fit into memory without compression, and with compression it does
  - Fitting in memory is good because reading from memory is much much **much** faster than reading from disk

    Transfer rate from memory ≈ 2 – 6 GB / sec

    Transfer rate from disk ≈ 50 – 500 MB / sec

    On a single disk, more like 50 MB / sec, for much higher bandwidths parallel disks are needed

Compression 3/6

- Index on **disk**:
  - Then compression saves disk space (obviously)
  - But it also saves query time, here is a realistic example:

Disk transfer time:	50 MB / second	
Compression rate:	Factor 5	
Decompression time:	30 MB / second	
Inverted list of size:	50 MB	
Reading uncompressed:	1.0 seconds	→ 50 MB
Reading compressed:	0.2 seconds	→ 10 MB
Decompressing:	0.3 seconds	→ 50 MB

Reading compressed + decompression **twice faster** compared to reading uncompressed

Compression 4/6

ids $> 2^{32} \approx 4B$
documents

- Gap encoding
  - Example inverted list (doc ids only):

    3, 17, 21, 24, 34, 38, 45, ..., 11876, 11899, 11913, ...
  - Numbers small in the beginning and large in the end, using an **int** requires **4 bytes** or even **8 bytes** per id
  - Alternative: store differences from one item to next:

    +3, +14, +4, +3, +10, +4, +7, ..., +12, +23, +14, ...
  - This is called **gap encoding**
  - Works as long as we process the lists from left to right
  - Now we have a sequence of mostly (but not always) small numbers ... how do we store these in little space?

Compression 5/6

- Binary representation
  - We can write number x in binary using $\lfloor \log_2 x \rfloor + 1$ bits

x	binary	number of bits	
1	1	1	$\lfloor \log_2 1 \rfloor + 1 = 1$
2	10	2	$\lfloor \log_2 2 \rfloor + 1 = 2$
3	11	2	$\lfloor \log_2 3 \rfloor + 1 = 2$
4	100	3	$\lfloor \log_2 4 \rfloor + 1 = 3$
5	101	3	

  - This encoding is optimal in a sense … see later slides
  - So why not just (gap-)encode like this and concatenate:

    +3, +14, +4, … → 11, 1110, 100, … → 111110100…

Compression 6/6

- Prefix-free codes, definition
  - Decode bit sequence from the last slide: 111110100

    This could be: +3, +14, +4 → 11, 1110, 100

    Could also be: +7, +6, +4 → 111, 110, 100

    Or: +3, +3, +2, + 4 → 11, 11, 10, 100
  - Problem: we have no way to tell where one code ends and the next code begins

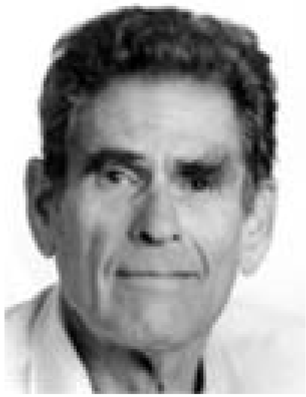
    Equivalently: some codes are prefixes of other codes
  - In a **prefix-free code**, no code is a prefix of another

    Then decoding from left to right is unambiguous !

Codes 1/5

- Elias-Gamma ... from 1975
  - Write $\lfloor \log_2 x \rfloor$ zeros, then x in binary like on slide 9
  - Prefix-free, because the number of initial zeros tells us exactly how many bits of the code come afterwards
  - Code for x has a length of exactly $2 \cdot \lfloor \log_2 x \rfloor + 1$ bits

1	1
2	010
3	011
4	00100
⋮	
10	0001010



Peter Elias
1923 – 2001

Codes 2/5

+1 because there is
no code for 0 in Elias-Gamma

- Elias-Delta ... also from 1975
  - Write $\lfloor \log_2 x \rfloor + 1$ in Elias-Gamma, followed by x in binary (like on slide 9) but **without** the leading 1
  - Prefix-free because the (prefix-free) Elias-Gamma code tells us exactly how many bits of the code come afterwards
  - Code for x has $\lfloor \log_2 x \rfloor + 2 \log_2 \log_2 x + O(1)$ bits

1	1
2	010 0
3	010 1
4	011 00
⋮	
10	00100 010

ELIAS-GAMMA

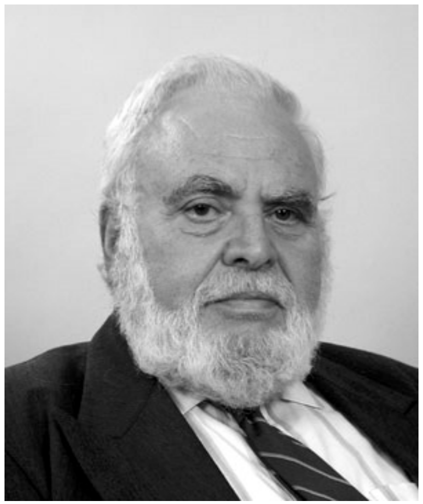
1	1
2	010
3	011
4	00100
⋮	⋮

Codes 3/5

q = quotient r = remainder

- Golomb (not Gollum) ... from 1966
  - Comes with an integer parameter M, called **modulus**
  - Write x as q · M + r, where q = x div M and r = x mod M
  - The code for x is then the concatenation of:
    - q written in unary with 0s $\lfloor \frac{x}{M} \rfloor$ bits
    - a single 1 (as a delimiter) 1 bit
    - r written in binary $\lceil \log_2 M \rceil$ bits

Solomon Golomb
1932 – 2016



$M = 16$, $x = 42$

$q = 42$ div $16 = 2 = 00$ in unary

$r = 42$ mod $16 = 10 = 1010$ in binary

Golomb code is 0 0 1 1 0 1 0

Codes 4/5

- Variable-Byte (VB)
  - Idea: use **whole bytes**, in order to avoid the (expensive) bit fiddling needed for the previous schemes

    VB is often used in practice, for exactly that reason

    In particular, for UTF-8 encoding … see Lecture 7
  - Use one bit of each byte to indicate whether this is the last byte in the current code or not

$x = 542$; x div $128 = 4 = 100$ in binary

x mod $128 = 30 = 11110$ in binary

1	0000100

Byte 1

says that there are more bytes to come

0	0011110

Byte 2

says that this is the last byte of the code

The "code point" is 0000100 : 0011110 which is 542 in binary

Codes 5/5

- Other codes
  - There is a huge variety of codes with different trade-offs wrt space reduction, compression speed, decompression speed
  - One particularly interesting recent coding scheme is called

    Asymmetric numeral systems (ANS) ... Duda et al, PCS'14
  - Close to optimal space reduction **and** fast (de)compression
  - It is actively used at Facebook, Apple, Google, ...
  - The basic idea is to encode a **whole message** (not just individual symbols) into a single (then fairly big) **integer**
  - It is an intricate scheme which, interestingly, is much easier to implement than to understand mathematically

    Check out the Wikipedia article if you are interested

Entropy 1/12

- Motivation
  - Which code compresses the best ?

    It depends !

    But on what ?
  - Roughly: it depends, on the relative frequency on the numbers / symbols we want to encode

    For example, in natural language, an "e" is much more frequent than a "z"

    So we should encode "e" with less bits than "z"
  - The next slides will make this more precise

Entropy 2/12

$H(X) = -\sum_{i=1}^{m} \frac{1}{m} \log_2 \frac{1}{m}$

$= -m \cdot \frac{1}{m} \cdot \log_2 \frac{1}{m} = \log_2 m$

- Definition
  - Entropy of a discrete random variable **X**

    Without loss of generality range of X = {1, ..., m}

    Think of X as generating the symbols of a message
  - Then the **entropy of X** is written and defined as

    $H(X) = - \Sigma_i \, p_i \log_2 p_i$ where $p_i = \mathrm{Prob}(X = i)$
  - We will see: H(X) is the optimal number of bits to encode a random symbol generated according to X
  - Example: all $p_i = 1/m$, then $H(X) = \log_2 m$

    If all m symbols are equally likely, the best encoding is the standard encoding with $\log_2$ m bits / symbol

Entropy 3/12

- Shannon's source coding theorem ... from 1948
  - Let X be a random variable with finite range
  - For an arbitrary prefix-free (**PF**) encoding, let L_i be the length of the code for $i \in \text{range}(X)$

    (1) For any PF encoding it holds: $\mathbf{E}\, L_X \geq H(X)$

    (2) There is a PF encoding with: $\mathbf{E}\, L_X \leq H(X) + 1$

    where $\mathbf{E}$ denotes the expectation
  - In words:

    No code can be better than the entropy, and there is always a code that is (almost) as good

Claude Shannon
1916 – 2001

Entropy 4/12

- Central Lemma ... to prove the source coding theorem
  - Denote by L_i the length of the code for the i-th symbol, then

    (1) Given a PF code with lengths L_i ⇨ $\Sigma_i\, 2^{-L_i} \leq 1$

    (2) Given L_i with $\Sigma_i\, 2^{-L_i} \leq 1$ ⇨ exists PF code with length L_i
  - Note: $\Sigma_i\, 2^{-L_i} \leq 1$ is known as "Kraft's inequality"
  - Intuitively: not all L_i can be small ... small L_i → large 2^{-L_i}

    For example, the lemma says that a prefix-free code where three $L_i = 1$ is not possible, because $2^{-1} + 2^{-1} + 2^{-1} > 1$

Entropy 5/12

- Proof of central lemma, part (1)
  - Show: given prefix-free code with lengths L_i then $\Sigma_i\ 2^{-L_i} \leq 1$
  - Generate a random binary sequence as follows:

    1. Pick one bit after the other, and independently

    2. Stop when you have a valid code, or when no more code is possible (= no code starts with that bit sequence)
  - Let C_i = the event that code i is generated
  - Observation: for prefix-free code, the C_i are **independent**
  - Thus $\underbrace{\Pr(C_1)}^{2^{-L_1}} + \ldots + \underbrace{\Pr(C_m)}^{2^{-L_m}} = \Pr(C_1 \cup \ldots \cup C_m) \leq 1$

    holds because $C_1, \ldots, C_m$ are independent
  - We also have $\Pr(C_i) = \underbrace{\frac{1}{2} \cdot \frac{1}{2} \cdot \ldots \cdot \frac{1}{2}}_{L_i \text{ times}} = 2^{-L_i}$
  - This proves $\Sigma_i\ 2^{-L_i} \leq 1$

Entropy 6/12

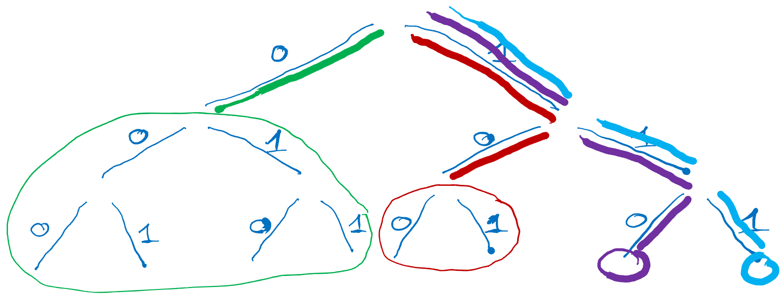
$L_1 = 1, \; L_2 = 2, \; L_3 = 3, \; L_4 = 3$

$\sum_{i=1}^{4} 2^{-L_i} = \frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \frac{1}{8} = 1$ ✓

THIS IS CALLED HUFFMAN ENCODING

- Proof of central lemma, part (2)
  - To show: L_i with $\Sigma_i \, 2^{-L_i} \leq 1$ ⇨ exists PF code with lengths L_i
  - Complete binary tree of depth $M = \max L_i$ ... has 2^M leaves

    in the example $M = 3$, $2^M = 8$
  - Mark all left edges 0, and all right edges 1
  - Consider the code lengths L_i in sorted order, smallest first
  - Then iterate: pick subtree with $2^{M - L_i}$ leaves that does not overlap with already picked subtrees ... path to that subtree gives code for symbol i and $\Sigma_i \, 2^{M - L_i} = 2^M \cdot \Sigma_i \, 2^{-L_i} \leq 2^M$



$L_1 = 1$, $2^{M-L_1} = 4$, Code $= 0$

$L_2 = 2$, $2^{M-L_2} = 2$, Code $= 10$

$L_3 = 3$, $2^{M-L_3} = 1$, Code $= 110$

$L_4 = 3$, $2^{M-L_4} = 1$, Code $= 111$

Entropy 7/12

- Proof of source coding theorem, part (1)
  - To show: for any PF encoding $\mathbf{E}\, L_X \geq H(X)$
  - By definition of expectation: $\mathbf{E}\, L_X = \Sigma_i\, p_i \cdot L_i$ (1)
  - By Kraft's inequality: $\Sigma_i\, 2^{-L_i} \leq 1$ (2)
  - Using Lagrange, it can be shown that, under the constraint (2), (1) is **min**imized for $L_i = \log_2 1/p_i$

    This is Exercise 1 from ES4 … it is perfect to practice Lagrangian optimization and to deepen understanding of the source coding theorem

  - Then $E\, L_X = \Sigma_i\, p_i \cdot L_i \geq \Sigma_i\, p_i \cdot \overbrace{\log_2 1/p_i}^{= -\log_2 p_i} = H(X)$

Entropy 8/12

- Proof of source coding theorem, part (2)
  - Show: there is a PF encoding with $\mathbf{E}\ L_X \le H(X) + 1$ $\quad \geq \log_2 \frac{1}{p_i}$ $\quad = \log_2 p_i$ $\quad = 1$
  - Let $L_i = \lceil \log_2 1/p_i \rceil$, then $\Sigma_i\ 2^{-L_i} = \sum_i 2^{-\lceil \log_2 \frac{1}{p_i} \rceil} \le \sum_i 2^{-\log_2 \frac{1}{p_i}} = \sum_i p_i$

    Note that the code length must be an integer, and we must round upwards, so that Kraft's inequality holds
  - By the central lemma, part (2), there then exists a PF code with code lengths L_i
  - By definition of expectation: $\mathbf{E}\ L_X = \Sigma_i\ p_i \cdot L_i$
  - Hence $\mathbf{E}\ L_X = \sum_i p_i \cdot \lceil \log_2 \frac{1}{p_i} \rceil \le \sum_i p_i \left(\log_2 \frac{1}{p_i} + 1 \right) = \underbrace{\sum_i p_i \log_2 \frac{1}{p_i}}_{= H(X)} + \underbrace{\sum_i p_i}_{= 1}$

Entropy 9/12

- Entropy-optimal codes
  - Consider a PF code with L_i = code length for symbol i and p_i = probability for symbol i
  - We say that the code is **optimal for distribution p_i** if

    $L_i \leq \log_2 1/p_i + 1$

  - This implies

    $\mathbf{E}\, L_X = \Sigma_i\, p_i \cdot L_i \leq \Sigma_i\, p_i \cdot (\log_2 1/p_i + 1) = H(X) + 1$

  - By Shannon's theorem this is the best we can hope for

    For the optimality proof from Exercise 2 from ES4, it suffices that you show $L_i \leq \log_2 1/p_i + \mathbf{O(1)}$

Entropy 10/12

- Universal codes
  - A prefix-free code is called **universal** if for **every** probability distribution over the symbols to be encoded

    $\mathbf{E}\, L_X = O(\, H(X)\,)$
  - That is, the expected code length is within a constant factor of the optimum for **any** distribution
  - Elias-Gamma, Elias-Delta, Golomb, and Variable-Byte are all universal in this sense
  - Note the difference to the stronger inequality from the previous slide for optimality for a particular distribution

    $\mathbf{E}\, L_X \le H(X) + 1$ versus $\mathbf{E}\, L_X = O(\, H(X)\,)$

Entropy 11/12

- Entropy-optimality of Elias-Gamma
  - Recall: code length for Elias-Gamma is $L_i = 2 \lfloor \log_2 i \rfloor + 1$
  - For which probability distribution is this entropy-optimal?
  - We need $L_i = 2 \lfloor \log_2 i \rfloor + 1 \leq \log_2 1/p_i + 1$
  - This suggests $p_i = 1 / i^2$ because:

    $p_i = 1 / i^2 \rightarrow \log_2 1/p_i = \log_2 i^2 = 2 \cdot \log_2 i$
  - We have to take care that the p_i sum to 1, however:

    $\Sigma_{i=1..\infty} 1 / i^2 = \pi^2 / 6 = 1.6449\ldots$
  - Hence let $p_i = 1 / i^2$ for $i \geq 2$, and p_1 such that $\Sigma_i\, p_i = 1$

    We have thus established a distribution, for which Elias-Gamma is entropy-optimal in the sense of slide 24

Entropy 12/12

- Entropy-optimality of Golomb
  - Consider the following random experiment for the generation of an inverted list L of length m :

    Include each document in L with probability p = m/n, independently of each other, where n = #documents

  - Let X be a fixed **gap** in this inverted list, then

    $\Pr(X = i) = (1 - p)^{i-1} \cdot p =: p_i$ for $i = 1, 2, 3, \ldots$

    Exercise 2 from ES4: Golomb is optimal for this distrib.

  - Bottom line: Golomb is optimal for gap-encoded lists

    But not too practical, because of the bit fiddling, see the definition of Golomb encoding on slide 13

References

- Textbook

  Section 5: Index compression

  Section 5.3: Postings file compression (some codes only)

- Wikipedia

  http://en.wikipedia.org/wiki/Elias_gamma_coding

  http://en.wikipedia.org/wiki/Elias_delta_coding

  http://en.wikipedia.org/wiki/Golomb_coding

  http://en.wikipedia.org/wiki/Variable-width_encoding

  http://en.wikipedia.org/wiki/Source_coding_theorem

  http://en.wikipedia.org/wiki/Kraft_inequality