Information Retrieval

WS 2015 / 2016

Lecture 4, Tuesday November 10th, 2015
(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
  - Assistant: Björn → Elmar
- Compression
  - Motivation — saves space **and** query time
  - Codes — Elias, Golomb, Variable-Byte
  - Entropy — Shannon's theorem
  - **Exercise Sheet 4:** prove the optimality of some of the codes + investigate the limits of what can be achieved

    We take a break from implementation work this week

Experiences with ES3 1/2

- Summary / excerpts
  - Interesting exercise, many liked performance tweaking
  - Submissions are getting less … not unusual though
  - Sentinels already implemented in C++, but not in Java
  - Not so easy to improve on a tuned baseline
  - Complex improvements can cost more than they help

    E.g. galloping always slower than binary search for some
  - Skip pointers can be simulated via a simple offset

    Like a low-budget version of galloping search
  - Seemingly insignificant code changes can affect runtime

Experiences with ES3 2/2

- Results
  - Three inverted lists of different lengths

film	171,951 postings	repeated 100 times
comedy	27,706 postings	repeated 100 times
2015	285 postings	repeated 100 times

  - Query film+2015, list length ratio = 603

    Any of galloping, skip ptrs, bin. search give large speedup
  - Query comedy+2015, list length ratio = 97

    Skipping helps, but not too much
  - Query film+comedy, list length ratio = 6

    Skipping costs more than it helps, switch to tuned baseline

Compression 1/6

- Motivation
  - Inverted lists can become very large

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

    For example, in the English Wikipedia:

    film: 1,667,147 occurrences
    year: 2,052,964 occurrences
    one: 4,022,417 occurrences

  - Compression potentially saves space **and** time

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 GB / second

    Transfer rate from disk ≈ 50 MB / second

Compression 3/6

when combined with zipping, can skip compressed parts with- and even decompressing them

- 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

for web-scale even 8 bytes / id

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

    3, 17, 21, 24, 34, 38, 45, ..., 11876, 11899, 11913, ...

  - Numbers small in the beginning, large in the end, using an int for each id would be **4 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	$= \lfloor \overset{=0}{\log_2 1} \rfloor + 1$
2	10	2	$= \lfloor \overset{=1}{\log_2 2} \rfloor + 1$
3	11	2	$= \lfloor \overset{=1}{\log_2 3} \rfloor + 1$
4	100	3	$= \lfloor \overset{=2}{\log_2 4} \rfloor + 1$
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 1/4

needs $\lfloor \log_2 x \rfloor + 1$ bits

- 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

$1 \rightarrow 1$

$2 \rightarrow 010$

$3 \rightarrow 011$

$4 \rightarrow 00100$

$\vdots$

$10 \rightarrow 0001010$

Peter Elias
1923 – 2001

Codes 2/4

+1 because there is no EG code for 0

- 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
  - Elias-Delta is also prefix-free and the length of the code length is $\lfloor \log_2 x \rfloor + 2 \log_2 \log_2 x + O(1)$ bits

Pr. in Exercise 4.1

$\lfloor \underbrace{\log_2 4}_{=2} \rfloor + 1 = 3$ in EG

Elias-Gamma

1 → 1
2 → 010
3 → 011
4 → 00100

1 → 1
2 → 0100 0
3 → 0101 1
4 → $\boxed{011}$ $\boxed{00}$ — 4 in binary w/o leading 1
⋮
10 → 00100 010

Codes 3/4

q = "quotient"
r = "remainder"

prefix free

ALSO PF
think about it

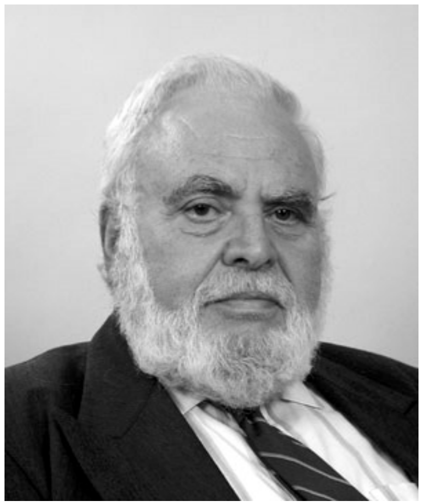
- Golomb (not Gollum) ... from 1966
  - Comes with an integer parameter M, called **modulus**
  - Write x as $q \cdot M + r$, where $q = x \text{ div } M$ and $r = x \bmod M$
  - The code for x is then the concatenation of:
    - q written in unary with 0s — q bits
    - a single 1 (as a delimiter) —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

$M = 16$, $x = 70 = \underset{q}{4} \cdot 16 + \underset{r}{6}$

Code for x is 0000 1 $\underbrace{0110}$

FIXED LENGTH for given M

Solomon Golomb
1932 – still alive



Codes 4/4

7 bits of „information" per byte

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

    VB often used in practice, for exactly that reason
  - Use one bit of each byte to indicate whether this is the last byte in the current code or not
  - VB is also used for UTF-8 encoding … see later lecture

$x = 521 = 4 \cdot 128 + 9$

Byte 1 Byte 2

VB-code for x $\boxed{1|0000100}$ $\boxed{0|0001001}$

0000100 : 0001001

would be 521 in bin.

↑ indicates that code continues

4 in binary

↑ indicates that last byte of code

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

$H(X) \geq 0$

$H(X) = 0$ iff one $p_i = 1$ all other zero

Note: for H, we define $0 \cdot \log_2 0 = 0$

- Entropy
  - **Intuitively:** the information content of a message = the optimal number of bits to encode that message
  - **Formally:** defined for a discrete random variable X

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

    Think of X as generating the symbols of the message

    Then the **entropy** of X is written and defined as

    $H(X) = - \Sigma_i \, p_i \log_2 p_i$ where $p_i = Prob(X = i)$

    E.g. m symbols, each equally likely $\Rightarrow p_i = \frac{1}{m}$

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

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 $L(x)$ be the length of the code for $x \in \text{range}(X)$

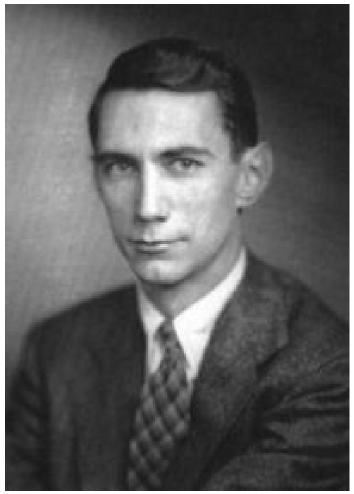
    (1) For any PF encoding it holds: $\mathbf{E}\, L(X) \geq H(X)$

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

    where **E** denotes the expectation

    In words: no code can be better than the entropy, and there is always a code as good

    almost



Claude Shannon
1916 – 2001

Entropy 4/12

Intuitively: not all L_i can be small (small $L_i \rightarrow$ large 2^{-L_i})

E.g. $\underset{2^{-1}}{L_1 = 1}$, $\underset{2^{-1}}{L_2 = 1}$, $\underset{2^{-1}}{L_3 = 1}$, ... NOT POSSIBLE

- **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"

UNI FREIBURG

Entropy 5/12

01101 $L_i = 5$

$\underbrace{(\frac{1}{2}) \cdot \ldots (\frac{1}{2})}_{5 \text{ terms}} = 2^{-L_i}$

- Proof of central lemma, part (1)
  - To show: given a PF code with lengths L_i ⇨ $\Sigma_i\, 2^{-L_i} \leq 1$
  - Consider the following random experiment:

    Generate a random binary sequence, and pick each bit independent from all other bits 0110 STOP

    Stop when you have a valid code, or when no more code is possible … well-defined for PF codes only !

    code for symbol i
  - Let C_i be the event that code i is generated

$\Pr(C_1 \dot\cup C_2 \dot\cup \ldots \dot\cup C_m) \leq 1$

$= \Pr(C_1) + \Pr(C_2) + \ldots + \Pr(C_m) = \sum_{i=1}^{m} 2^{-L_i}$

$\Pr(C_i) = 2^{-L_i}$

Entropy 6/12

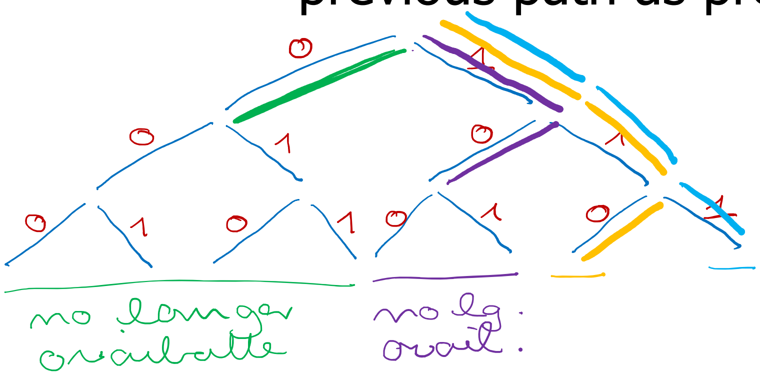
GIVEN:
$L_1 = 1,\ L_2 = 2,\ L_3 = 3,\ L_4 = 3$
$\sum 2^{-L_i} = \frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \frac{1}{8} = 1$ o.k.

- Proof of central lemma, part (2)

  ensures that there is always enough of the tree "left"

  - To show: L_i with $\Sigma_i\ 2^{-L_i} \leq 1$ ⇨ exists PF code with length L_i
  - Consider a complete binary tree of depth $\max L_i$ $= 3$
  - Mark all left edges 0, and all right edges 1
  - Consider the code lengths L_i in sorted order, smallest first
  - Then iterate: pick a path of length L_i from the root, with no previous path as prefix … this gives a PF code for symbol i

This principle is called Huffman encoding



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

Code for 1: 0
Code for 2: 10
Code for 3: 110
Code for 4: 111

Entropy 7/12

$p_1, \dots, p_m$ = prob. distr. over symbols

$\Sigma p_i = 1$

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

"average code length"

constraint

$$\Rightarrow \underline{E\,L(X)} = \Sigma_i\, p_i \cdot L_i \geq \Sigma_i\, p_i \cdot \log_2 1/p_i \underline{= H(X)}$$

LAGRANGE (sketch):

Exercise 4.4

$$\mathcal{L} = \sum_{i=1}^{m} p_i L_i + \lambda \left(1 - \sum_{i=1}^{m} 2^{-L_i}\right) \qquad 2^{-L_i} = e^{-\ln 2 \cdot L_i}$$

$$\frac{\partial \mathcal{L}}{\partial L_i} = p_i + \lambda \cdot \ln 2 \cdot 2^{-L_i} \stackrel{!}{=} 0 \quad \Rightarrow \dots \Rightarrow L_i = \log_2 \frac{1}{p_i}$$

Entropy 8/12

Given $p_1, \ldots, p_m$

- Proof of source coding theorem, part (2)
  - Show: there is a PF encoding with **E** $L(X) \leq H(X) + 1$
  - Let $L_i = \lceil \log_2 1/p_i \rceil$, then $\Sigma_i \, 2^{-L_i} \leq 1$

    $\log_2 1/p_i$ is the "ideal" code length

    Note that rounding is necessary because the code length must be an integer, and that we need to 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: **E** $L(X) = \Sigma_i \, p_i \cdot L_i$

$$E\,L(X) = \sum_{i=1}^{m} p_i \cdot \lceil \log_2 \tfrac{1}{p_i} \rceil \leq \underbrace{\sum_{i=1}^{m} p_i \cdot \log_2 1/p_i}_{= H(X)} + \underbrace{\sum_{i=1}^{m} p_i}_{= 1}$$

$$\lceil \log_2 \tfrac{1}{p_i} \rceil \leq \log_2 \tfrac{1}{p_i} + 1$$

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

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

    Then **E** $L(X) \leq H(X) + 1$ and by Shannon's theorem this is the best we can hope for

    For the optimality proofs from Exercise Sheet 4, 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 **universal** if for <u>every</u> probability distribution over the symbols to be encoded

    E L(X) = O(H(X))

    That is, the expected code length is within a constant factor of the optimum for <u>any</u> distribution

  - Elias-Gamma, Elias-Delta, Golomb, and Variable-Byte are all universal in this sense

    For a finer distinction, the definition of optimality from the previous slide is better

    E L(X) ≤ H(X) + 1 versus E L(X) = O(H(X))

Entropy 11/12

$p_i = 1/i^2$

$\Rightarrow \log_2 \frac{1}{p_i} = \log_2 i^2$

$= 2 \cdot \log_2 i$

- 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 something like $p_i \approx 1 / i^2$
  - Let $p_i = 1 / i^2$ for $i \geq 2$, and p_1 such that $\Sigma_i \, p_i = 1$

    That is, numbers $i \geq 2$ occur with probability $1 / i^2$

$$\sum_{i=1}^{\infty} \frac{1}{i^2} = \frac{\pi^2}{6} = 1.6\ldots$$

Entropy 12/12

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

    Include each document i in L with probability p = m/N, independently of each other, where N = #documents
  - Let G be a fixed **gap** in this inverted list, then

    $\Pr(G = i) = (1 - p)^{i-1} \cdot p =: p_i$ for $i = 1, 2, 3, \ldots$
  - Exercise 4.3: prove that Golomb is optimal for this distr.
  - Bottom line: Golomb is optimal for gap-encoded lists

    But not practical, because of the bit fiddling, see slide 14

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