Information Retrieval

WS 2012 / 2013

Lecture 4, Wednesday November 14th, 2012

(Compression and Entropy)

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

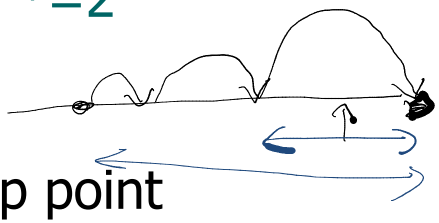
Overview of this lecture

- Organizational
  - Your results and experiences with ES#3 (List Intersection)
- Compression
  - Important to save (index) space
  - But also to save query time !
  - We will see some compression schemes relevant for IR
  - Analyze entropy = information content
  - Shannon's source coding theorem / optimal codes
  - **Exercise Sheet 4:**

    Prove a variety of interesting properties about coding schemes and their entropy

Experiences with ES#3 (list intersection)

- Summary / excerpts   last checked November 14, 16:08
  - Confusion about random list generation ... sorry!
  - Stuff about code optimization was very interesting
  - Math.pow in java is *really* slow ... just use gap *=2
  - Some of the Java people had space problems
  - Binary search can start from last exp-search jump point
  - Some were excited about their code improvements
  - Others were frustrated that they didn't achieve more
  - Live coding was nice again / unit test was helpful
  - Thanks again for great tutor feedback
  - How to autocomplete in Vim ... see my .vimrc on the Wiki



Results for ES#3 (list intersection)

- Main observations + discussion
  - For R=5 hard to make ExpBin faster

    most: ExpBin a bit slower; few: much slower; few: faster
  - For R=50 ExpBin usually faster

    most: somewhat faster; few: a lot faster; few: slower
  - **Reason:** ExpBin algorithmically better, but more complex code (larger constant factor + harder to optimize m.code)
  - Also Simple is faster for R=50 than for R=5
  - **Reason:** long runs can be skipped in single while loop
  - Some Java codes were faster than the fastest C++ codes
  - **Reason:** are you sure your code is correct?

Compression

- Motivation
  - Index lists can be very large for large text collections
  - May have to be stored on disk
  - Compression then obviously saves space
  - But also query time:

    Reading an inverted list from disk takes time

    Typical disk read rate: 50 – 100 MB / second
  - Assume 50 MB / sec and an inverted list of size 50 MB

    Then reading that list from disk takes 1 second

    If we compress it to 10 MB, reading takes 0.2 second

    We need to decompress it then, but even if that takes 0.3 seconds, we have still gained a factor of two !

Compressing inverted lists

- Example of an inverted list of document ids

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

  - Numbers can become very large ... so we need 4 bytes to store each, for web search even more
  - But we can also store the list like this

  +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 small numbers
  - We need a scheme to store small numbers in few bits

Universal encoding

- Encode small (positive) integers in few bits
  - **Ideally:** use $\log_2 x$ bits to encode $x \in N$
  - We certainly can't do better than that

    $\log_2 n$ bits needed to differentiate between n numbers
  - We can't even get $\leq 2 \cdot \log_2 x$ bits ... see Exercise 4.3 !

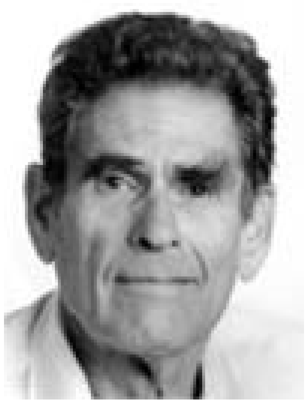
Prefix-free codes

- For our purposes, codes should be **prefix-free**
  - That is: no encoding of a symbol must be a prefix of an encoding of some other symbol
  - Assume the following code (which is not prefix-free)
    - A encoded by 1, B encoded by 11
    - now what does the sequence 1111 encode?
    - could be AAAA or ABA or BAA or AAB or BB
  - For a prefix-free code, decoding is unambiguous
  - And so are all the codes we will consider in this lecture

Elias encodings 1/2

- Elias-Gamma encoding, from 1975
  - Write x in binary, and prepend floor($\log_2 x$) zeros
  - Prefix-free, intuitively because the initial zeros tell us how long the binary representation of x is ... Exercise 4.1
  - Code for x uses ≈ 2 $\log_2 x$ bits ... exact length: Exercise 4.2
  - Let's look at the Elias-Gamma codes of 1, 2, 3, 4, 5, ...

1	1
010	2
011	3
00100	4
00101	5
00110	6



*1923 New Jersey
†2001 Massachusetts

Elias encodings 2/2

- Elias-Delta encoding, also from 1975
  - Write x in binary, prepend Elias-Gamma code of $\text{floor}(\log_2 x) + 1$
  - Prefix-free for basically the same reason ... Exercise 4.1
  - This requires $\log_2 x + O(\log \log x)$ bits
  - Let's look at the Elias-Delta codes of 1, 2, 3, 4, 5, ...

1 1	1
0 1 0 1 0	2
0 1 0 1 1	3
0 1 1 1 0 0	4
0 1 1 1 0 1	5
0 1 1 1 1 0	6

Entropy encoding

- What if the numbers are not in sorted order
  - Or not numbers at all but just symbols

    C C B A D B B A B B C B B C B D

    0101100010010 . . . .
  - Give each number a code corresponding to its **frequency**
  - Frequencies in our example: A: 2 B: 8 C: 4 D: 2
  - A prefix-free code: B → 1 C → 01 D → 0010 A → 0001

    Requires: $8 \cdot 1 + 4 \cdot 2 + 2 \cdot 4 + 2 \cdot 4 = 32$ bits

    That is: 2 bits / symbol on average

    Better than the obvious 3-bit code
  - How do we know if / when we have reached the optimum?

Entropy 1/7

$H(x) = \sum_i p_i \log_2 \frac{1}{p_i}$

$p_i \leq 1 \Rightarrow H(x) \geq 0$

- Definition
  - **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 $p_i = \text{Prob}(X = i)$

$x \log_2 x \xrightarrow[x \to 0]{} 0$

EXAMPLE 1: $p_i = \frac{1}{m} \; \forall i$ $H(x) = \log_2 m$ equi-distr.

EXAMPLE 2: $p_j \Rightarrow 1$ for one j $H(x) = 0$

$p_i \to 0$ for other i

Entropy 2/7

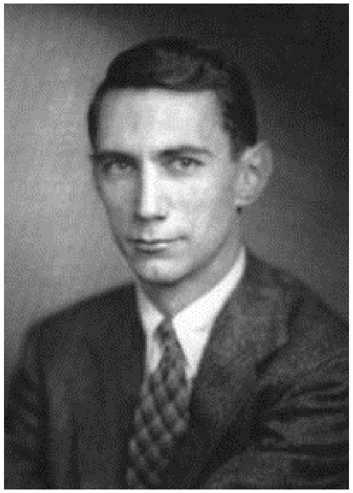
- Shannon's famous source coding theorem (1948)
  - Let X be a random variable with finite range
  - For an arbitrary prefix-free (**PF**) encoding C let $L_C(x)$ be the length of the code for $x \in \text{range}(X)$

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

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

    where $\mathbf{E}$ denotes the expectation
  - **Intuitively:** no code can be better than the entropy, and there always is a code which is almost as good

*1916 Michigan
†2001 Massachusetts



Entropy 3/7

- Proof of the source coding theorem
  - Denote by Li the length of the code for the i-th symbol
  - The following lemma is key for the source coding theorem:

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

    (2) Given Li with $\Sigma_i 2^{-Li} \leq 1$ ⇨ exists PF code with length Li

    $\Sigma_i 2^{-Li} \leq 1$ is known as "Kraft's inequality"

Entropy 4/7

1 ←

1 1 1

- Proof of Lemma, part (1)

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

Consider the following random experiment:

101101...

0 or 1 with prob. $\frac{1}{2}$ until I get a code OR no more code possible

This is well-defined! (only for PF codes)

C_i = the event that I get code i.

$$(*) = \Pr(C_1 \text{ or} \ldots \text{ or } C_m)$$

$$= \sum_{i=1}^{m} \Pr(C_i) = \sum_{i=1}^{m} 2^{-L_i} \leq 1$$

↑ because (*) is a probability

Entropy 5/7

- Proof of Lemma, part (2)

  Given Li with $\sum_i 2^{-L_i} \leq 1$ ⇨ exists PF code with length Li

For example $\underset{L_1}{3}, \underset{L_2}{3}, \underset{L_3}{1}, \underset{L_4}{2}$

Check that

$\sum 2^{-L_i} = 1$

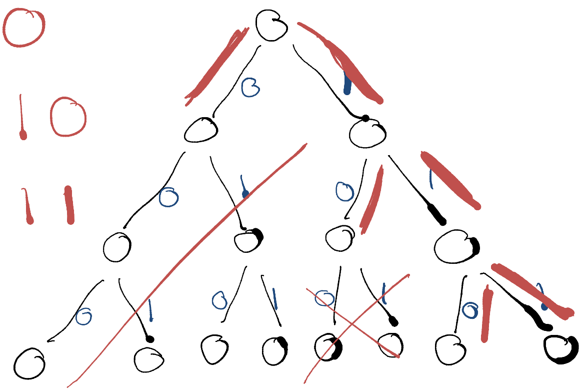
$\underset{1/8}{2^{-3}} + \underset{1/8}{2^{-3}} + \underset{1/2}{2^{-1}} + \underset{1/4}{2^{-2}}$

$C_3 = 0$

$C_4 = 10$

$C_2 = 110$

$C_1 = 111$



This is the principle behind Huffman Encoding

Entropy 6/7

Correction 26 Nov 12 20:47
* tidier write-up
* use equality for side constraint, by introducing variable s

- Proof of source coding theorem, part (1)

For any PF encoding C it holds: **E** $L_C(X) \geq H(X)$

By def. of expectation: $E\,L(X) = \sum_{i=1}^{m} p_i \cdot L_i$

By Kraft's inequality: $\sum_{i=1}^{m} 2^{-L_i} = s \leq 1$

LAGRANGE AGAIN:

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

$$\frac{\partial \mathcal{L}}{\partial L_j} = p_j - \lambda \cdot \ln 2 \cdot 2^{-L_j} \overset{!}{=} 0$$

$$\Longrightarrow p_j = \lambda \cdot \ln 2 \cdot 2^{-L_j}$$

$$\Longrightarrow 1 = \sum_{i=1}^{m} p_i = \lambda \cdot \ln 2 \cdot \sum_{i=1}^{m} 2^{-L_i} \Rightarrow \lambda \cdot \ln 2 = 1/s$$

$$\Longrightarrow p_j = \underbrace{1/s}_{\geq 1} \cdot 2^{-L_j} \Longrightarrow L_j \geq \log_2 \frac{1}{p_j}$$

$$\frac{\partial^2 \mathcal{L}}{\partial^2 L_j} > 0 \Rightarrow \text{MIN} \Rightarrow \sum_{i=1}^{m} p_i \cdot L_i \geq \sum_{i=1}^{m} p_i \cdot \log_2 \frac{1}{p_i} = H(X) \blacksquare$$

$$[e^x]' = e^x$$

$$2^x = e^{\ln 2 \cdot x}$$

$$\Rightarrow [2^{-x}]' = -\ln 2 \cdot e^{-\ln 2 \cdot x} = -\ln 2 \cdot 2^{-x}$$

Entropy 7/7

$$H(X) = \sum p_i \cdot \log_2 \frac{1}{p_i}$$

- Proof of source coding theorem, part (2)

  There is a PF encoding C with: **E** $L_C(X) \le H(X) + 1$

$$E\,L(X) = \sum_{i=1}^{m} p_i L_i \le \sum_{i=1}^{m} p_i \left(\log_2 \frac{1}{p_i} + 1\right)$$

Just set $L_i = \lceil \log_2 \frac{1}{p_i} \rceil \Rightarrow \sum_{i=1}^{m} 2^{-L_i} \le \sum_{i=1}^{m} 2^{-\log_2 \frac{1}{p_i}}$

$= \sum_{i=1}^{m} p_i = 1$

Kraft's inequality

$\Rightarrow \exists$ PF code

$$\Rightarrow = \underbrace{\sum_{i=1}^{m} p_i \log_2 \frac{1}{p_i}}_{H(X)} + \underbrace{\sum_{i=1}^{m} p_i}_{=1} = H(X) + 1 \quad \blacksquare$$

Optimality of Elias-Gamma

- Elias encoding
  - Elias code lengths satisfy $Li \leq 2 \log_2 i + 1$
  - 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 $1 / i^2$
  - Recall **E** $L(X) = \Sigma_i\ p_i\ Li$ and $H(X) = -\Sigma_i\ p_i \log_2 p_i$
  - Then we have **E** $L(X) \leq H(X) + 1$

Golomb encoding 1/2

- A slightly more involved encoding from 1966
  - Comes with a parameter M, called modulus
  - Write positive integer x as q · M + r
  - Where q = x div M and r = x mod M
  - The code for x is then the concatenation of:

    (1) the quotient q written in unary with 0s

    (2) a single 1 (as a delimiter)

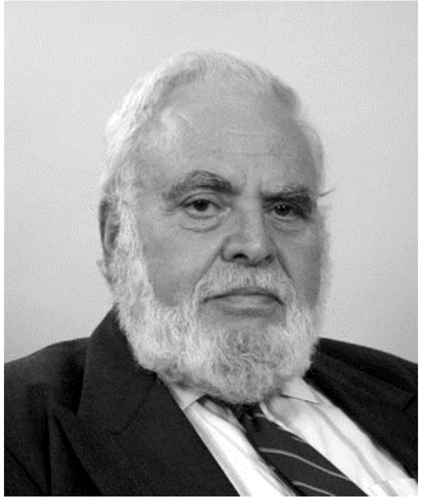
    (3) the remainder r written in binary in $\lceil \log_2 M \rceil$ bits

$M = 10$, $x = 37$

$q = 3$; $r = 7$

code : $000.1.\underbrace{0111}_{\lceil \log_2 M \rceil \text{ bits}}$

Solomon Golomb
*1932 Maryland



Golomb encoding 1/2

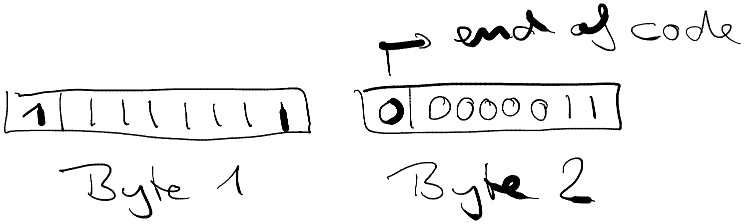
- Analysis
  - Golomb codes are optimal for gap-encoding inverted list

    You should prove this yourself in Exercise 4.4
  - Typically not used in practice, however !
  - **Reason:** the additional decompression effort usually does not outweigh the slight improvement in space, compared to simpler schemes

Variable-Byte Encoding

- A very simple scheme often used in practice
  - Use **whole bytes**, in order to avoid the (computationally expensive) bit fiddling needed for the previous schemes
  - Use one bit of each byte to indicate, whether this is the last byte in the current code or not
  - This is also used for the UTF-8 encoding ... later lecture

511 = 111111111



References

- In the Raghavan/Manning/Schütze textbook

  Section 5: Index compression

  Section 5.3: Postings file compression ... (some codes only)

- Relevant Wikipedia articles

  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