# Information Retrieval

## WS 2013 / 2014

### Lecture 4, Tuesday November 12th, 2013
### (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
  - Motivation: saves space (obviously), but also query time
  - Concrete schemes: Elias, Golomb, Variable-Byte
  - Shannon's theorem: optimal compression = entropy
  - **Exercise Sheet 4:** prove optimality of Golomb encoding for gap-encoded inverted lists + verify experimentally

# Experiences with ES#3 (list intersection)

- Summary / excerpts     last checked November 12, 15:00
  - Gallop not hard to understand + exercise quite feasible
  - Many of you spent most of their time on debugging

    Don't worry, this will get better with practice !
  - Surprised that the (theoretically optimal) Gallop is so slow

# Results for ES#3 (list intersection)

- Main observations + discussion
  - Let R be the ratio between the two list lengths
  - For R=2 (university german), simple is unbeatable
  - For R=13 (university berlin), simple is still hard to beat
  - For R=198 (university freiburg), gallop is faster
  - Reason: gallop asymptotically faster than simple, but more complex code = larger constant factors in the running time

# Compression 1/4

- Motivation
  - A search engine index can become very large

    Understand: total number of index items = total size of the text collection in words
  - Index in **memory**:

    Then compression saves memory (obviously)

    Also note that an index might be to large to fit into memory without compression, and with compr. it does

    Fitting in memory is good because reading from memory is (much) faster than reading from disk

# Compression 2/4

- Motivation
  - Index on **disk**:

    Then compression saves disk space (obviously)

    But is also saves query time:

    Reading an inverted list from disk takes a lot of time

    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 !

# Compression 3/4

- Compressing inverted lists

  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

# Compression 4/4

- 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

10100111001

- **Elias-Gamma** encoding, from 1975
  - Write $\lfloor \log_2 x \rfloor$ zeros, ~~then 1~~, then $x$ in binary
  - Prefix-free, because the number of initial zeros tells us exactly how many bits of the code come afterwards
  - Code for $x$ uses $2 \cdot \lfloor \log_2 x \rfloor + 1$ bits … **verify yourself !**
  - Let's look at the Elias-Gamma codes of 1, 2, 3, 4, 5, …

1 1
010 2
011 3
00100 4
00101 5

*1923 New Jersey
†2001 Massachusetts

# Elias encodings 2/2

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

- **Elias-Delta** encoding, also from 1975
  - Write $\lfloor \log_2 x \rfloor + 1$ in Elias-Gamma, then $x$ in binary
  - Prefix-free, because the initial Elias-Gamma code (which is itself prefix-free) tells us exactly how many bits of the code come afterwards … **again, verify this yourself !**
  - This requires $\lfloor \log_2 x \rfloor + 2 \log_2 \log_2 x + O(1)$ bits … **verify !**
  - Let's look at the Elias-Delta codes of 1, 2, 3, 4, 5, ...

| | | |
|---|---|---|
| 1 1 | 1 | 1 = Elias-Gamma for 1 |
| 0 1 0 1 0 | 2 | 010 = —"— for 2 |
| 0 1 0 1 1 | 3 | 011 = —"— for 3 |
| 0 1 1 1 0 0 | 4 | ⋮ |
| 0 1 1 1 0 1 | 5 | |

# Entropy 1/7

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

Example: $p_i = \frac{1}{m}$ i.e. $\Pr(X = i) = \frac{1}{m}$

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

$$= \log_2 m \quad \blacksquare$$

# Entropy 2/7

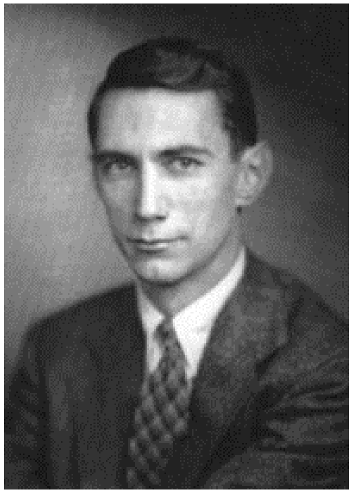
- Shannon's famous source coding theorem (1948)
  - Let $X$ be a random variable with finite range
  - For an arbitrary prefix-free (**PF**) encoding, let $L(x)$ be the length of the code for $x \in \text{range}(X)$

    (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 $\mathbf{E}$ denotes the expectation

    **Remember:** no code can be better than the entropy, and there is always a code which is almost as good



*1916 Michigan
†2001 Massachusetts

# Entropy 3/7

$L_1 = 1$
$L_2 = 1$

$\sum_i 2^{-L_i} = \frac{1}{2} + \frac{1}{2} = 1$

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

    (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

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

# Entropy 4/7

01001...

$\rightarrow$ 010

01011 $\leftarrow$

UNI FREIBURG

- Proof of central lemma, part (1)

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

  – Consider the following random experiment:

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

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

  – Let $C_i$ be the event that code $i$ is generated

$$\underbrace{\Pr(C_1 \cup \ldots \cup C_m)}_{\underbrace{\Pr(C_1) + \cdots + \Pr(C_m)}_{= \sum_{i=1}^{m} 2^{-L_i}}} \le 1$$

$$\Pr(C_i) = \left(\tfrac{1}{2}\right)^{L_i} = 2^{-L_i}$$

# Entropy 5/7

not fully containing a previous path

- Proof of central lemma, part (2)

  Given $L_i$ with $\Sigma_i\ 2^{-L_i} \le 1$ ⇨ exists PF code with length $L_i$

  – Consider a complete + nice binary tree of depth max $L_i$

  – Marks all left edges 0, and all right edges 1

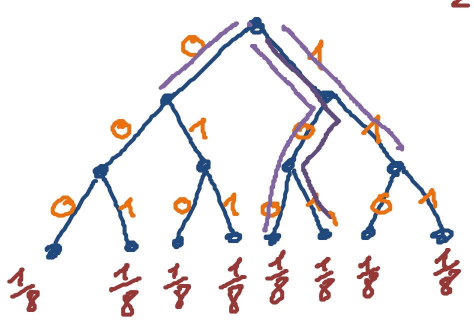
  – Consider the code lengths $L_i$ in sorted order, smallest first

  – Then iterate: pick a path of length $L_i$ from the root, ~~disjoint from all previous path~~ … this gives a PF code for symbol i

GIVEN : 1 , 2 , 3 , 3

$L_1$, $L_2$, $L_3$, $L_4$

$2^{-L_i}$: $\frac{1}{2}$, $\frac{1}{4}$, $\frac{1}{8}$, $\frac{1}{8}$




1 : 0

2 : 1 1

3 : 1 0 0

4 : 1 0 1

VERIFY: $\sum_{i=1}^{4} 2^{-L_i}$

$= \frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \frac{1}{8} = 1 \le 1$ ✓

This is called Huffman coding

# Entropy 6/7

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

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

  **Verify yourself** … good exam preparation exercise !

$$\sum_{i=1}^{m} 2^{-L_i} = \sum_{i=1}^{m} 2^{-\log_2 \frac{1}{p_i}}$$
$$= \sum_{i=1}^{m} 2^{\log_2 p_i}$$
$$= \sum_{i=1}^{m} p_i = 1$$

# Entropy 7/7

$$\sum_{i=1}^{m} 2^{-\lceil \log_2 1/p_i \rceil} \le \sum_{i=1}^{m} 2^{+\log_2 p_i} = \sum_{i=1}^{m} p_i = 1$$

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

    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:

$$E\, L(X) = \sum_{i=1}^{m} p_i \cdot \underbrace{\left\lceil \log_2 \frac{1}{p_i} \right\rceil}_{\le \log_2 \frac{1}{p_i} + 1}$$

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

# Entropy-optimal encodings 1/2

- Definition
  - A PF code for $X$ is entropy-optimal if

    $\mathbf{L_i \leq \log_2 1/p_i + O(1)}$

    where $L_i$ = code length for symbol $i$, and $p_i = Pr(X = i)$
  - **Understand from previous slides:**

    Then $\mathbf{E}\ L(X) \leq H(X) + O(1)$ ... and there cannot be a much better PF code for $X$, since always $\mathbf{E}\ L(X) \geq H(X)$
  - If all positive integers can be encoded, such a code is called **universal**

# Entropy-optimal encodings 1/2

- Elias-Gamma is a universal code
  - 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 $\underline{L_i \leq \log_2 1/p_i + 1}$.
  - 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 $1 / i^2$

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

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

$\approx 0.64$

$\Rightarrow p_1 \approx 0.36$

$p_2 = \frac{1}{4}$

$p_3 = \frac{1}{9}$

$p_4 = \frac{1}{16}$

$\vdots$

for $i \geq 2$

$$\Rightarrow L_i = 2\lfloor \log_2 i \rfloor + 1$$
$$\leq 2 \cdot \log_2 i + 1$$
$$= \log_2 i^2 + 1$$
$$= \log_2 \frac{1}{p_i} + 1$$

for $i = 1$

$$L_1 = 1$$
$$\leq \log_2 \frac{1}{p_1} + 1$$

# Golomb encoding 1/2

- A slightly more involved encoding from 1966
  - Comes with a parameter M, called modulus
  - Write positive integer x as $q \cdot M + r$
  - Where $q = x \text{ div } M$ and $r = x \bmod M$
  - 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)

    write $\lceil \log_2 M \rceil$ bits

    (3) the remainder r written in binary

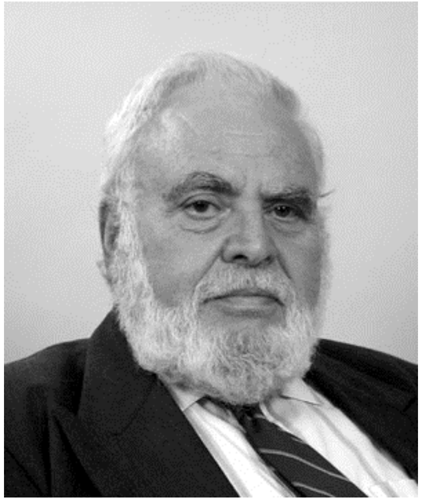
$M = 16$ , $x = 37$

$37 = \underbrace{2}_{q} \cdot 16 + \underbrace{5}_{r}$

$00 \qquad 0101 \implies \underbrace{00}_{q}1\underbrace{0101}_{r}$

fixed length, unlike for Elias

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 4.1 and 4.2
  - However, the implementation of Golomb requires quite a lot of "bit fiddling", which costs time in decompression

    Implement and evaluate it yourself in Exercise 4.3

    **NOTE:** we have prepared quite a lot of working code for you already (for both Java and C++) so that this exercise is not too much work … see also next slide

# 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

    We have fully implemented VB encoding for you (in both Java and C++), including **timing**, **tests** and **everything**

    Use this as a template for your GolombEncoding code !
  - VB is also used for UTF-8 encoding ... later lecture

byte

$\underbrace{\boxed{1|0011000}\ \boxed{1|0010001}\ \boxed{0|1111001}}_{\text{3-byte code}}$ $\underbrace{\boxed{1|0000001}\ \boxed{0|0100001}}_{\text{2-byte code}}$

codepoint 001100000100011111001

$= 24 + 17 \cdot 128 + 121 \cdot 128^2$

$= 1.984.664$

codepoint 00000010100001

# 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