Search Engines

WS 2009 / 2010

Lecture 4, Thursday November 12th, 2009

(IO- and Cache-Efficiency, Compression)

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

UNI FREIBURG

Goals

UNI FREIBURG

- Learn why compression is important
  - cache efficiency: sequential vs. random access to memory
  - IO-efficiency: sequential vs. random access to disk
- Learn about various compression schemes
  - Elias, Golomb, Variable Byte, Simple-9
- Learn some important theoretical foundations
  - entropy = optimal compressibility of data
  - which scheme is optimal under which circumstances?
  - how fast can one compress / decompress?

IO-Efficiency (IO = Input / Output)

UNI FREIBURG

- Sequential vs. random access to disk
  - typical disk transfer rate is 50 MB / second
  - typical disk seek time is 5 milliseconds
- This means that
  - .. when we write code for data is so large that it does not completely fit into memory (as is typical for search apps) …
  - then we must take great care that whenever we read from (or write to) disk, we actually read (or write) big *blocks of data*
  - an algorithm which does that is called IO-efficient
  - more formally → next slide

IO-Efficiency — Formalization

UNI FREIBURG

- Standard RAM model (RAM = random access machine)
  - count the number of operations
    - one operation = an arithmetic operation, read/write a single memory cell, etc.
  - for example, sorting n integers needs $O(n \cdot \log n)$ operations
- IO-Model or: External memory model
  - data is read / written in blocks of B contiguous bytes
  - count the number of block reads / write
  - ignore all other operations / computational costs
  - for example, the IO-efficiency of sorting is $O(n/B \cdot \log_{M/B} n/B)$

    were M is the size of the main memory briefly explain bound

Cache-Efficiency

- Basic functioning of a cache

draw interactively

UNI FREIBURG

Comparison: Disk / Memory

UNI FREIBURG

- Both have in common that
  - sequential access is much more efficient than random access
- The ratios are very different though
  - disk
    - ~ 50 MB / second transfer rate
    - ~ 5 milliseconds seek time
    - this implies an optimal block size of ~ 250 KB
  - memory
    - ~ 1 GB / second sequential read
    - ~ 100 nanoseconds for a random access / cache miss
    - this implies an optimal block size of ~ 100 bytes

IO / Cache Efficiency

UNI FREIBURG

- Understand
  - considering IO and cache efficiency is *key* for the efficiency of any program you write that deals with non-trivial amounts of data
  - an algorithm that tries to access data in blocks as much as possible is said to have good locality of access
- In the exercise …
  - … you will write two programs that compute exactly the same function, but with very different locality of access
  - let's see which running time differences you measure

Compression

UNI FREIBURG

- Now we can understand why compression is important
  - for information retrieval / search
  - or for any application that deals with large amounts of data
- Namely
  - consider a query
  - assume that query needs 50 MB of data from disk
  - then it takes 1 second just to read this data
  - assume the data is stored compressed using only 5 MB
  - now we can read it in just 0.1 seconds
  - assume it takes us 0.1 seconds to decompress the data
  - then we are still 5 times faster then without compression

    (assuming that the internal computation is << 0.1 seconds)

Compressing inverted lists

UNI FREIBURG

- Example of an inverted list of document ids

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

  – numbers can become very large
  – 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
  – need a scheme which stores small numbers in few bits
  – such a scheme is called universal encoding → next slide

Universal encoding

- Goal of universal encoding
  - store a number in x in $\sim \log_2 x$ bits
  - less is not possible, why? → Exercise
- Elias code
  - write the number x to be encoded in binary
  - prepend z zeroes, where $z = \text{floor}(\log_2 x)$
  - examples
    -
    -
    -

Prefix Freeness

UNI FREIBURG

- For our purposes, codes should be prefix free
  - prefix free = 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
  - the Elias code is prefix-free → Exercise
  - and so are all the codes we will consider in this lecture

Elias-Gamma Encoding

UNI FREIBURG

- Elias encoding
  - uses $\sim 2 \log_2 x$ bits to encode a number x
  - that is about a factor of 2 off the optimum
  - the reason is the prepended zeroes (unary encoding)
- Elias-Gamma encoding
  - encode the prepended zeroes with Elias
  - show example

  - now $\log_2 x + 2 \cdot \log_2 \log_2 x$ bits
  - this can be iterated → $\log2\, x + 2 \log_2^{(k)} x$ bits
  - what is the optimal k? → Exercise

Entropy Encoding

UNI FREIBURG

- 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
- Entropy encoding
  - give each number a code corresponding to its frequency
  - frequencies in our example: A: 2 B: 8 C: 4 D: 2
  - prefix-free codes: 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
  - but is it the best we can get?

Definition of Entropy

UNI FREIBURG

- Entropy
  - defined for a discrete random variable X

    (that is, for a probability distribution with finite range)
  - assume w.l.o.g that X is from the range {1, …, m}
  - let p_i = Prob(X = i)
  - then the entropy of X is written and defined as

    $H(X) = - \sum_{i=1,\ldots m} p_i \cdot \log p_i$
- Examples
  - equidistribution: $p_i = 1/n \quad \rightarrow \quad H = \log_2 n$
  - deterministic: $p_i = 1$, all others 0 $\quad \rightarrow \quad H = 0$

intuitively: entropy = average #bits to encode a symbol

Source Coding Theorem

- By Claude Shannon, 1948
  - let X be random variable with finite range
  - let C be a (binary) code for the possible values

    $C(x)$ = code for value x from the range

    $L(x)$ = length of that code
  - Then

    $E[\, L(X) \,] \geq H(X)$

    $E[\, L(X) \,] \leq H(X) + 1$

Claude Shannon
*1916 Michigan
†2001 Massachusetts

Proof of Source Coding Theorem

- Prove lower bound, give hints on upper bound
- Key: the Kraft inequality

Entropy Encoding ↔ Universal Encoding

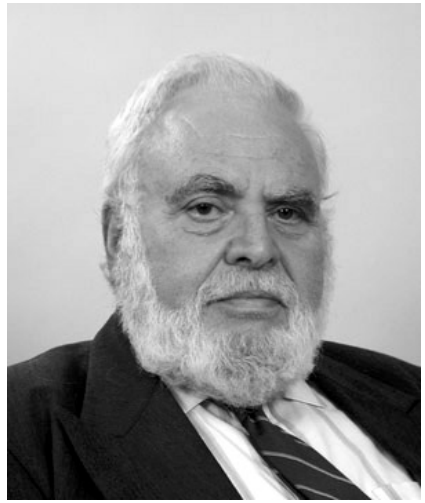
UNI FREIBURG

- Recall
  - entropy-optimal encoding gives a code with $\log_2 1/p(x)$ bits to a symbol which occurs with probability $p(x)$
  - optimal universal encoding gives a code with $c \cdot \log_2 x + O(1)$ bits to a positive integer x
- Therefore, by the source code theorem
  - universal encoding is the entropy-optimal code when number x occurs with probability $\sim 1 / x^c$
  - for example, the Elias code is optimal when number x occurs with probability $\sim 1 / x^2$

Golomb encoding

UNI FREIBURG

- By Solomon Golomb, 1966
  - comes with a parameter M (modulus)
  - write positive integer x as $q \cdot M + r$
  - where $q = x \text{ div } M$ and $r = x \text{ mod } M$
  - the codeword for x is then the concatenation of
    - the quotient q written in unary with 0s
    - a single 1 (as a delimiter)
    - the remainder r written in binary
  - examples



Solomon Golomb
*1932 Maryland

Golomb Encoding — Analysis

UNI FREIBURG

- Show that Golomb encoding is optimal
  - for gap-encoding inverted lists
  - assuming the doc ids in a list of size m are a random subset of size m of all doc ids $1..n$

Simpler encodings — Variable Byte

- Variable byte encoding
  - always use $8 \cdot x$ bits → codes aligned to byte boundaries
  - most significant bit of byte indicates whether code continues
  - examples

  - advantages:
    - simple
    - faster to decompress than non-byte aligned code

Simpler encodings — Simple9

UNI FREIBURG

- Simple-9 Encoding (Anh and Moffat, 2005)
  - align to full machine words (used to be: 4-byte ints)
  - each int is split into two parts x (4 bits) and y (28 bits)
  - x says how y is to be interpreted
  - depending on y, x is interpreted as
    - 14 (small) numbers of 2 bits each, or
    - 9 (small) numbers of 3 bits each, or
    - …
    - 1 number of 28 bits
  - advantage: decompression of a whole 4-byte int can be hard-coded for each possible x
  - this gives a super fast decompression
  - compression ratio is not optimal but ok