V.2 Index Compression

Heap's law (empirically observed and postulated):
Size of the vocabulary (distinct terms) in a corpus

$$E[\,distinct\ \ terms\ in\ corpus\,] \approx \alpha \cdot n^{\beta}$$

with total number of term occurrences n, and constants α, β ($\beta < 1$),
classically $\alpha \approx 20$, $\beta \approx 0.5$ ($\rightarrow$ ~ 3 Mio terms for 20 Bio docs)

Zipf's law (empirically observed and postulated):
Relative frequencies of terms in the corpus

$$P[k^{th}\ most\ \ popular\ \ term\ \ has\ \ rel.\ freq.\ x] \propto \left(\frac{1}{k}\right)^{\theta}$$

with parameter θ, classically set to 1

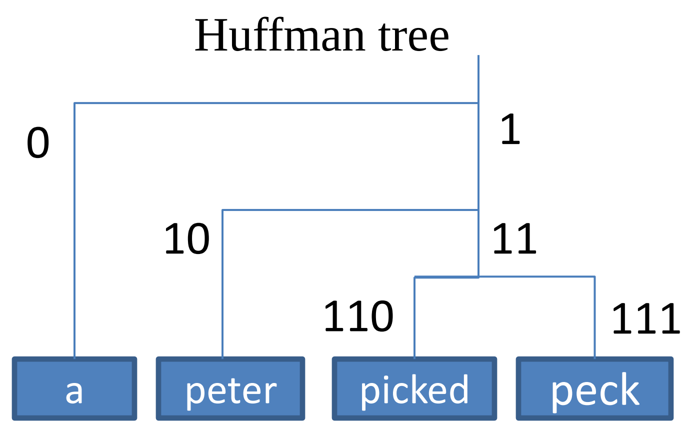
$\rightarrow$ Both laws strongly suggest opportunities for compression!

Recap: Huffman Coding

Variable-length unary code based on frequency analysis of the underlying distribution of symbols (e.g., words or tokens) in a text.

Key idea: choose shortest unary sequence for most frequent symbol.

Symbol x	Frequency f(x)	Huffman Encoding
a	0.8	0
peter	0.1	10
picked	0.07	110
peck	0.03	111



Let f(x) be the probability (or relative frequency) of the x-th symbol in some text d. The **entropy** of the text (or the underlying prob. distribution f) is: $H(d) = \sum_x f(x) \log_2 \frac{1}{f(x)}$

H(d) is a lower bound for the average (i.e., expected) amount of *bits per symbol* needed with optimal compression. Huffman comes close to H(d).

Overview of Compression Techniques

- Dictionary-based encoding schemes:
  - **Ziv-Lempel: LZ77**
    (entire family of Zip encodings: GZip, BZIP2, etc.)

- Variable-length encoding schemes:
  - **Variable-Byte encoding** (byte-aligned)
  - **Gamma**, **Golomb/Rice** (bit-aligned)
  - **S16** (byte-aligned, actually creates entire 32- or 64-bit words)
  - **P-FOR-Delta**
    (bit-aligned, with extra space for "exceptions")
  - **Interpolative Coding (IPC)**
    (bit-aligned, can actually plug in various schemes for binary code)

Ziv-Lempel Compression

LZ77 (Adaptive Dictionary) and further variants:

- Scan text & identify in a ***lookahead window*** the longest string that occurs repeatedly and is contained in a ***backward window***.
- Replace this string by a "pointer" to its previous occurrence.

Encode text into list of triples ***<back, count, new>*** where

- ***back*** is the backward distance to a prior occurrence of the string that starts at the current position,
- ***count*** is the length of this repeated string, and
- ***new*** is the next symbol that follows the repeated string.

Triples themselves can be further encoded (with variable length).

Better variants use explicit dictionary with statistical analysis (need to scan text twice).

Example: Ziv-Lempel Compression

peter_piper_picked_a_peck_of_pickled_peppers

<0, 0, p>	for character 1:	p
<0, 0, e>	for character 2:	e
<0, 0, t>	for character 3:	t
<-2, 1, r>	for characters 4-5:	er
<0, 0, _>	for character 6:	_
<-6, 1, i>	for characters 7-8:	pi
<-8, 2, r>	for characters 9-11:	per
<-6, 3, c>	for charaters 12-13:	_pic
<0, 0, k>	for character 16	k
<-7,1,d>	for characters 17-18	ed

…

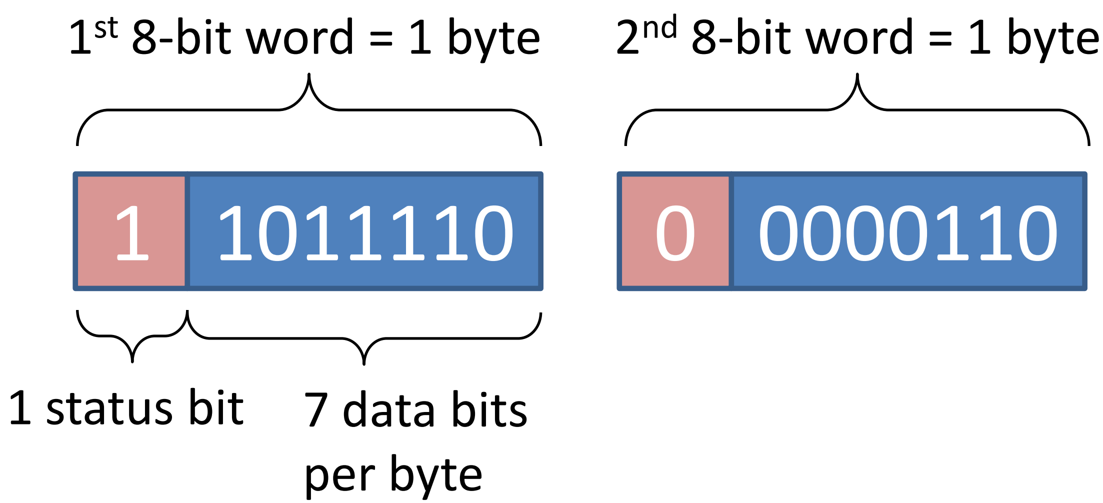
great for text, but not appropriate for index lists

Variable-Byte Encoding

- Encode sequence of numbers into variable-length bytes using one status bit per byte indicating whether the current number expands into next byte.

<u>Example:</u>

To encode the decimal number 12038, write:



Thus needs 2 bytes instead of 4 bytes (regular 32-bit integer)!

Gamma Encoding

Delta-encode **gaps** in inverted lists (successive doc ids):

Unary coding:
gap of size x encoded by:
$\log_2(x)$ times 0 followed by one 1
($\log_2(x) + 1$ bits)

good for short gaps

Binary coding:
gap of size x encoded by
binary representation of number x
($\log_2 x$ bits)

good for long gaps

Gamma (γ) coding:
length:= floor($\log_2$ x) in unary, followed by
offset := x – 2^(floor($\log_2$ x)) in binary
Results in (1 + $\log_2$ x + $\log_2$ x) bits per input number x

→ generalization: **Golomb/Rice code** (optimal for geometr. distr. x)
→ still need to pack variable-length codes into bytes or words

Example: Gamma Encoding

Number x	Gamma Encoding
$1 = 2^0$	1
$5 = 2^2 + 2^0$	00101
$15 = 2^3+2^2+2^1+2^0$	0001111
$16 = 2^4$	000010000

<u>Particularly useful when:</u>

- Distribution of numbers (incl. largest number) is not known ahead of time
- Small values (e.g., delta-encoded docIds or low TF*IDF scores) are frequent

Golomb/Rice Encoding

For a tunable parameter M, split input number x into:

- Quotient part q := floor(x/M) stored in unary code (using q x 1 + 1 x 0)
- Remainder part r:= (x mod M) stored in binary code
  - If M is chosen as a power of 2,
    then r needs $\log_2(M)$ bits (→ **Rice encoding**)
  - else set b := ceil($\log_2(M)$)
    - If $r < 2^b - M$, then r as plain binary number using b-1 bits
    - else code the number $r + 2^b - M$ in plain binary representation using b bits

M=10

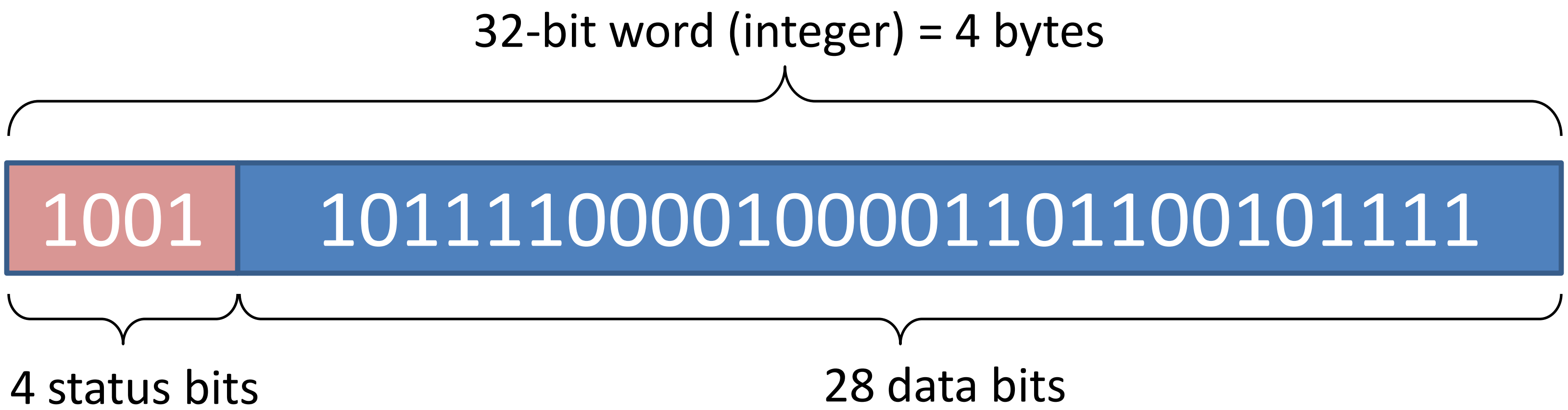
b=4

Example:

Number x	q	Output bits q	r	Binary (b bits)	Output bits r
0	0	0	0	0000	000
33	3	1110	3	0011	011
57	5	111110	7	1101	1101
99	9	1111111110	9	1111	1111

S9/S16 Compression

[Zhang, Long, Suel: WWW'08]



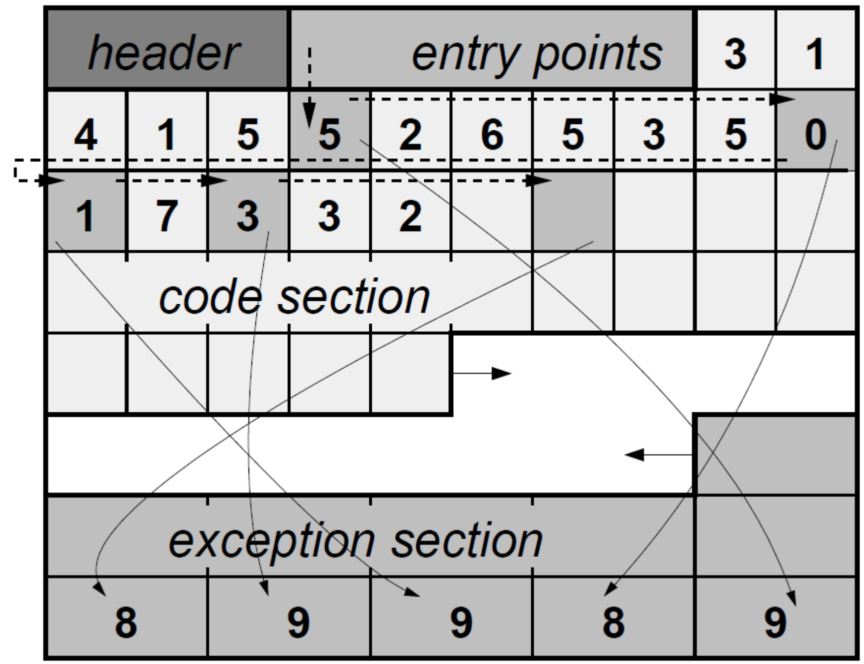
- Byte aligned encoding (32-bit integer words of fixed length)
- 4 status bits encode 9 or 16 cases for partitioning the 28 data bits
  - <u>Example:</u> If the above case 1001 denotes 4 x 7 bit for the data part, then the data part encodes the decimal numbers: 94, 8, 54, 47
- Decompression implemented by case table or by hardcoding all cases
- High cache locality of decompression code/table
- Fast CPU support for bit shifting integers on 32-bit to 128-bit platforms

P-FOR-Delta Compression

[Zukowski, Heman, Nes, Boncz: ICDE'06]

For "Patched Frame-of-Reference" w/Delta-encoded Gaps

- Key idea: encode individual numbers such that "most" numbers fit into **b bits**.
- Focuses on encoding an entire block at a time by choosing a value of b bits such that **[high$_{coded}$, low$_{coded}$] is small**.
- Outliers ("exceptions") stored in extra **exception section** at the end of the block in reverse order.

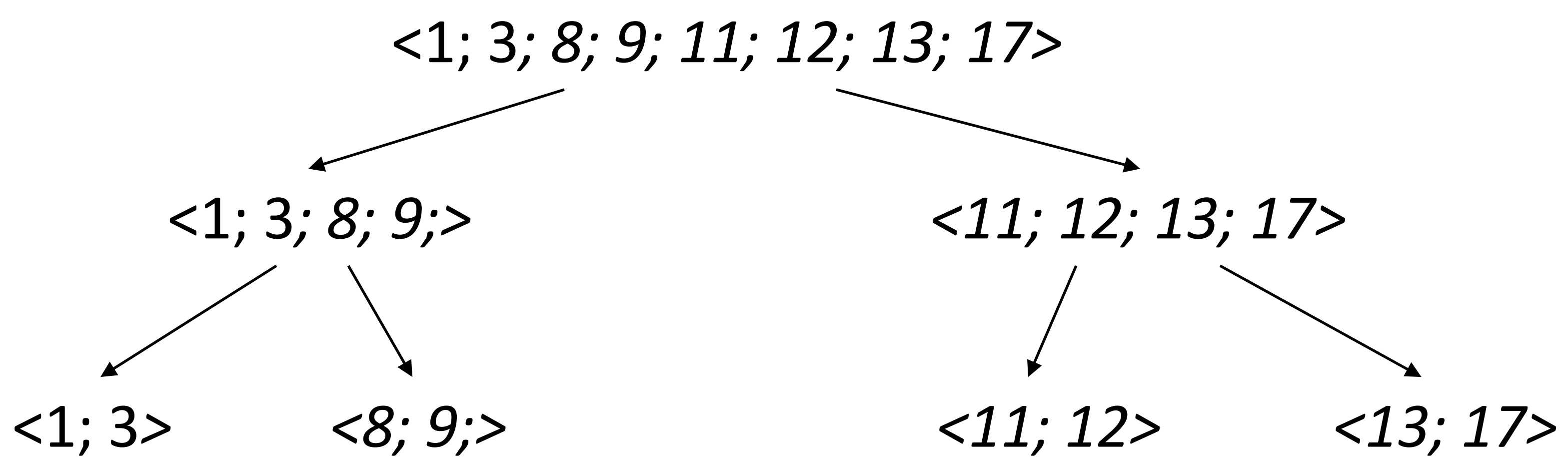


Encoding of **31415926535897932** using b=3 bitwise coding blocks for the code section.

Interpolative Coding (IPC)

[Moffat, Stuiver: IR'00]

- IPC directly encodes docIds rather than gaps.
- Specifically aims at bursty/clustered docId's of similar range.
- Recursively splits input sequence into low-distance ranges.

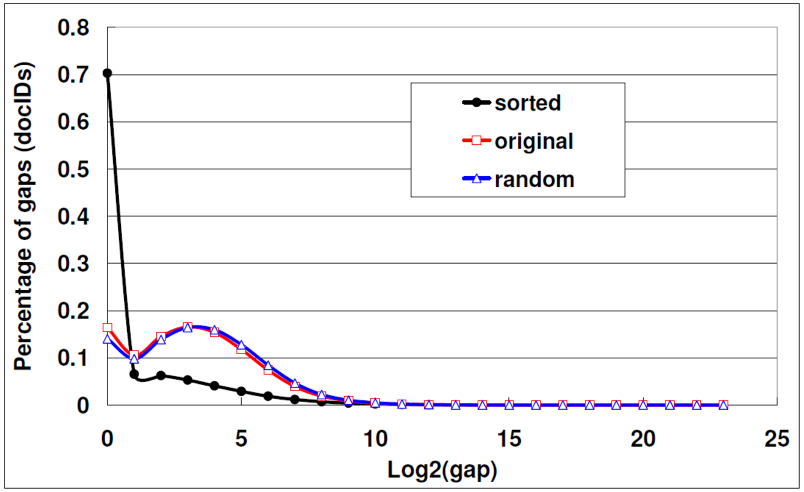


- Requires ceil($\log_2(\text{high}_i - \text{low}_i + 1)$) bits per number for bucket i in binary!
- <u>But:</u> → Requires the decoder to know all $\text{high}_i/\text{low}_i$ pairs.

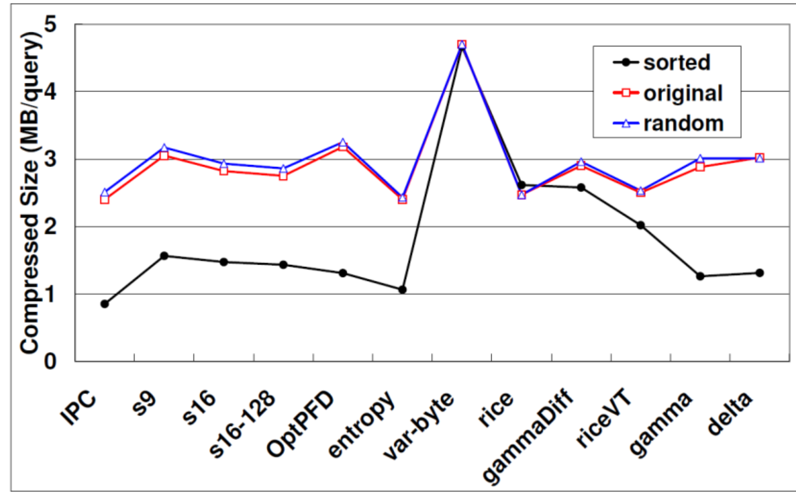
  → Need to know large blocks of the input sequence in advance.

Comparison of Compression Techniques

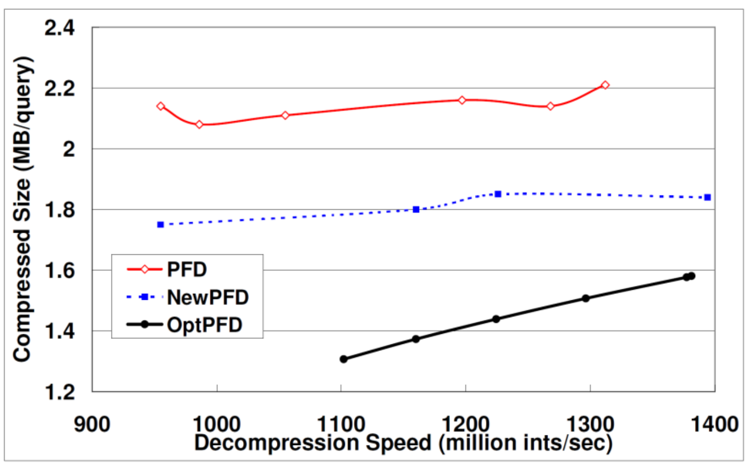
[Yan, Ding, Suel: WWW'09]



Distribution of docID-gaps on TREC GOV2 (~25 Mio docs) reporting averages over 1,000 queries



Compressed docId sizes (MB/query) on TREC GOV2 (~25 Mio docs) reporting averages over 1,000 queries

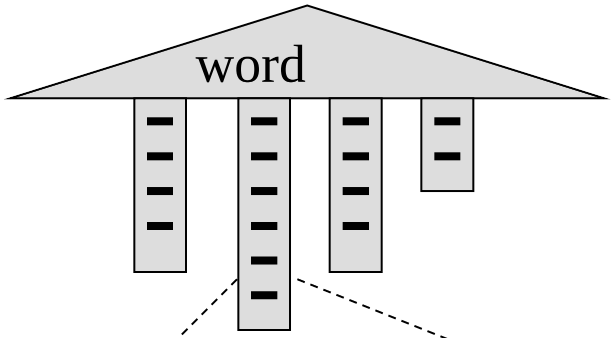


Decompression speed (MB/query) for TREC GOV2, 1,000 queries

- Variable-length encodings usually win by far in (de-) compression speed over dictionary & entropy-based schemes, at comparable compression ratios!

Layout of Index Postings

[J. Dean: WSDM 2009]



word	skip table	block 1	…	block N

one block
(with n postings):

delta to last docId in block	header
#docs in block: n	header
n-1 docId deltas: Rice_M encoded	docId postings
tf values : Gamma encoded	payload (of postings)
term attributes: Huffman encoded	payload (of postings)
term positions: Huffman encoded	payload (of postings)

layout allows incremental decoding

Additional Literature for Chapters V.1-2

Indexing with inverted files:

- J. Zobel, A. Moffat: Inverted Files for Text Search Engines, Comp. Surveys 2006
- S. Brin, L. Page: The Anatomy of a Large-Scale Hypertextual Web Search Engine, WWW 1998
- L.A. Barroso, J. Dean, U. Hölzle: Web Search for a Planet: The Google Cluster Architecture. IEEE Micro 2003
- J. Dean, S. Ghemawat: MapReduce: Simplified Data Processing in Large Clusters, OSDI 2004
- X. Long, T. Suel: Three-Level Caching for Efficient Query Processing in Large Web Search Engines, WWW 2005
- H. Yan, S. Ding, T. Suel: Compressing Term Positions in Web Indexes, SIGIR 2009
- J. Dean: Challenges in Building Large-Scale Information Retrieval Systems, Keynote, WSDM 2009, http://videolectures.net/wsdm09_dean_cblirs/

Inverted index compression:

- Marvin Zukowski, Sándor Héman, Niels Nes, Peter A. Boncz: Super-Scalar RAM-CPU Cache Compression. ICDE 2006
- Jiangong Zhang, Xiaohui Long, Torsten Suel: Performance of compressed inverted list caching in search engines. WWW 2008
- Alistair Moffat, Lang Stuiver: Binary Interpolative Coding for Effective Index Compression. Inf. Retr. 3(1): 25-47 (2000)
- Hao Yan, Shuai Ding, Torsten Suel: Inverted index compression and query processing with optimized document ordering. WWW 2009