Techniques for Inverted Index Compression

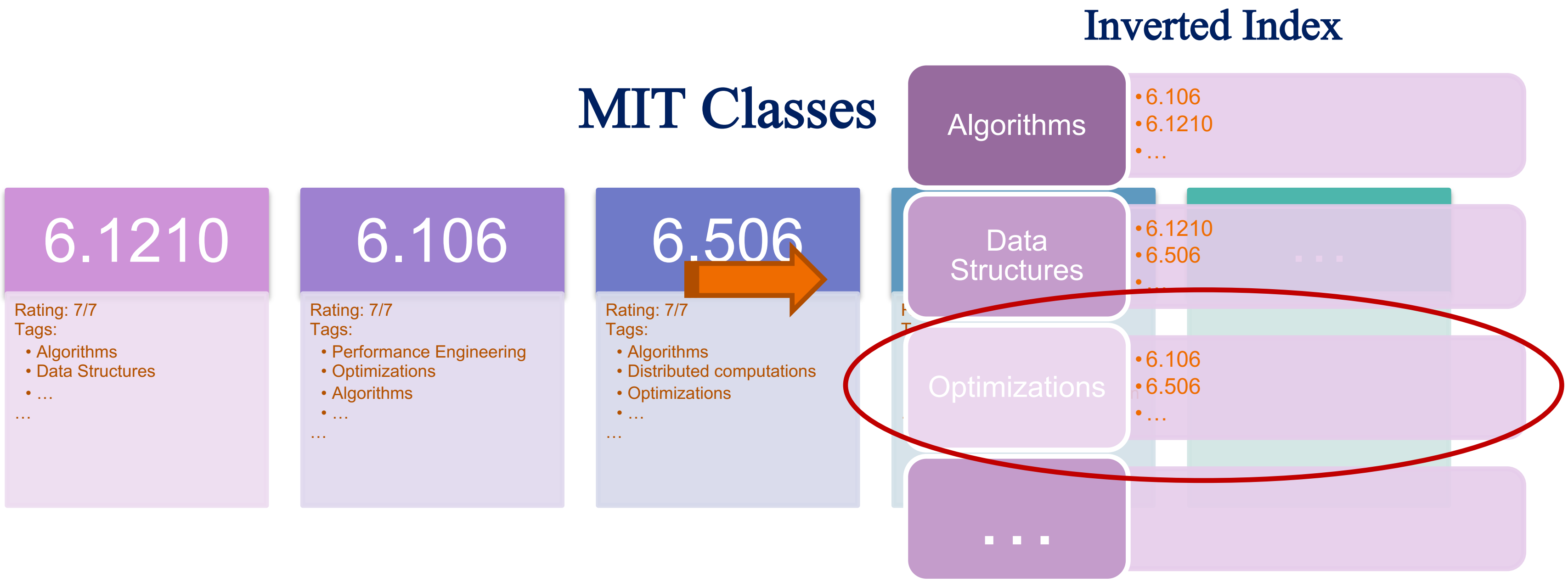
Giulio Ermanno Pibiri, Rossano Venturini

Presented by: Giorgi Kldiashvili

What Is An Inverted Index?

- A data structure used in information retrieval systems to efficiently retrieve documents or web pages containing a specific term or set of terms.
- In an inverted index, the index is organized by terms, and each term points to a list of documents or web pages that contain that term.
- Typically used to optimize efficiency of data retrieval queries.
- Has a good structure for optimizations.
- Used in variety of applications:
    - Search engines
    - Document retrieval systems
    - Recommendation systems
    - Social networks
    - Bioinformatics
    - Database management systems
    - etc

Inverted Index Example



What is the highest rated class about Optimizations?

Problem:
Inverted Index can be very large!

- Google Search index contains hundreds of bil[illegible] of [illegible] and i[illegible] we[illegible] over 100,000,000 gigabytes in size[1].
- The posts index alone in Faceb[illegible]k G[illegible]ph Search [illegible]ouses over 700 TB of data and includes over [illegible] factor[illegible] for surfacing the most relevant content[2].

Compress Them!!!

[1] https://www.google.com/search/howsearchworks/how-search-works/organizing-information/
[2] https://www.searchenginewatch.com/2013/10/25/facebook-on-graph-search-posts-index-700-tb-of-data-100-ranking-factors/

Goals

- Survey encoding algorithms suitable for Inverted Index Compression
- Characterize their performance through experimentations
- Evaluate them using space and memory usage

Overview

- High level definition of compression techniques split into three subgroups.
- Description of the evaluation methodology.
- Experiment results and final thoughts.

Inverted Index Compression Technique Types

Integer Compressors

- Unary and Binary
- Gamma and Delta
- Golomb
- Rice
- Zeta
- Fibonacci
- Variable-Byte
- SC-Dense

List Compressors

- Binary packing
- Simple
- PForDelta
- Elias-Fano
- Interpolative
- Directly-addressable
- Hybrid
- Entropy encodings

Entire Index Compressors

- Clustered
- ANS-based
- Dictionary-based

Timeline of Compression Techniques

1949	Shannon-Fano [32, 93]
1952	Huffman [43]
1963	Arithmetic [1][1]
1966	Golomb [40]
1971	Elias-Fano [30, 33]; Rice [87]
1972	Variable-Byte and Nibble [101]
1975	Gamma and Delta [31]
1978	Exponential Golomb [99]
1985	Fibonacci-based [6, 37]
1986	Hierarchical bit-vectors [35]
1988	Based on Front Coding [16]
1996	Interpolative [65, 66]
1998	Frame-of-Reference (For) [39]; modified Rice [2]
2003	SC-dense [11]
2004	Zeta [8, 9]

2005	Simple-9, Relative-10, and Carryover-12 [3]; RBUC [60]
2006	PForDelta [114]; BASC [61]
2008	Simple-16 [112]; Tournament [100]
2009	ANS [27]; Varint-GB [23]; Opt-PFor [111]
2010	Simple8b [4]; VSE [96]; SIMD-Gamma [91]
2011	Varint-G8IU [97]; Parallel-PFor [5]
2013	DAC [12]; Quasi-Succinct [107]
2014	Partitioned Elias-Fano [73]; QMX [103]; Roaring [15, 51, 53]
2015	BP32, SIMD-BP128, and SIMD-FastPFor [50]; Masked-VByte [84]
2017	Clustered Elias-Fano [80]
2018	Stream-VByte [52]; ANS-based [63, 64]; Opt-VByte [83]; SIMD-Delta [104]; general-purpose compression libraries [77]
2019	DINT [79]; Slicing [78]

Integer Compressors

Integer Encoding Goals

- Map each integer to unique binary string codeword.
- Ideally $|C(x)| \approx \log_2(1/\mathbb{P}(x))$.
- Good decoding and encoding performance.
- Low overhead for storing the encoding details.

Prefix-free Code

- No codeword is a prefix of another codeword.
- Can be rearranged so that lexicographical ordering stays intact.
- In this lexicographical ordering, codewords with same lengths will end up in consecutive order.
- Can be uniquely decoded.
- Lexicographical ordering can be exploited to increasing encoding and decoding performance.

Prefix-free Encodings

(a)

x	Codewords	*Lengths*	*Values*
1	0	1	0
2	100	3	64
3	101	3	80
4	11000	5	96
5	11001	5	100
6	11010	5	104
7	11011	5	108
8	1110000	7	112
–	–	–	127

(b)

Lengths	*First*	*Values*
1	1	0
2	2	64
3	2	64
4	4	96
5	4	96
6	8	112
7	8	112
–	9	127

Prefix-free Encodings

Encode(x) :

determine ℓ such that $\mathit{first}[\ell] \leq x < \mathit{first}[\ell + 1]$

$\mathit{offset} = x - \mathit{first}[\ell]$

$\mathit{jump} = 1 \ll (M - \ell)$

$\text{Write}((\mathit{values}[\ell] + \mathit{offset} \times \mathit{jump}) \gg (M - \ell), \ell)$

Decode() :

determine ℓ such that $\mathit{values}[\ell] \leq \mathit{buffer} < \mathit{values}[\ell + 1]$

$\mathit{offset} = (\mathit{buffer} - \mathit{values}[\ell]) \gg (M - \ell)$

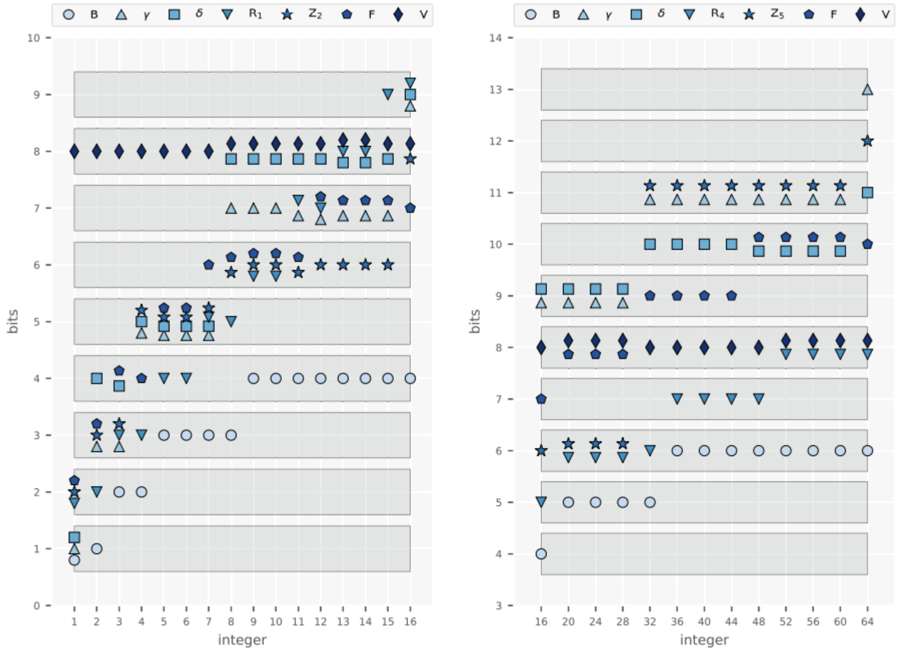
$\mathit{buffer} = ((\mathit{buffer} \ll \ell)\ \&\ \text{MASK}) + \text{Take}(\ell)$

return $\mathit{first}[\ell] + \mathit{offset}$

Integer Encoding

Encoding	Optimal when $\mathbb{P}(x) \approx$
Unary	$1/2^x$
Binary	$1/2^k$
Gamma	$1/(2x^2)$
Delta	$1/(2x(\log_2 x)^2)$
Golumb	$p(1-p)^{x-1}$
Rice	$p(1-p)^{x-1}$
Zeta	$1/(\zeta(\alpha)x^{\alpha})$
Fibonnaci	$1/(2x^{\frac{1}{log_2\phi}}) \approx 1/(2x^{1.44})$
VByte	$\sqrt[7]{1/x^8}$
SC-Dense	$(s+c)^{-k(x)}$

Codeword Length



Integer Encoding

Encoding	Optimal when $\mathbb{P}(x) \approx$
Unary	$1/2^x$
Binary	$1/2^k$
Gamma	$1/(2x^2)$
Delta	$\mathbf{1/(2x(\log_2 x)^2)}$
Golumb	$p(1-p)^{x-1}$
Rice	$p(1-p)^{x-1}$
Zeta	$1/(\zeta(\alpha)x^\alpha)$
Fibonnaci	$1/(2x^{\frac{1}{\log_2 \phi}}) \approx 1/(2x^{1.44})$
VByte	$\sqrt[7]{1/x^8}$
SC-Dense	$(s+c)^{-k(x)}$

Codeword Length

Unary Encoding

- Encode x as $1^{x-1}0$.
- $|C(x)| = x$.
- Optimal when $\mathbb{P}(x) \approx 1/2^x$.

x	U(x)
1	0
2	10
3	110
4	1110
5	11110
6	111110
7	1111110
8	11111110

Binary Encoding

- Encode x as $bin(x-1)$.
- $|C(x)| \approx \log_2(\max\{x\}) = k$.
- Optimal when $\mathbb{P}(x) \approx 1/2^k$.

x	$\mathrm{B}(x)$
1	0
2	1
3	10
4	11
5	100
6	101
7	110
8	111

Gamma Encoding

- Encode x as unary representation of $|bin(x)|$ followed by $(|bin(x)| - 1)$ bits from $bin(x)$.
- $|C(x)| = 2|bin(x)| - 1$.
- Optimal when $\mathbb{P}(x) \approx 1/(2x^2)$.

x	$\gamma(x)$
1	0.
2	10.0
3	10.1
4	110.00
5	110.01
6	110.10
7	110.11
8	1110.000

Delta Encoding

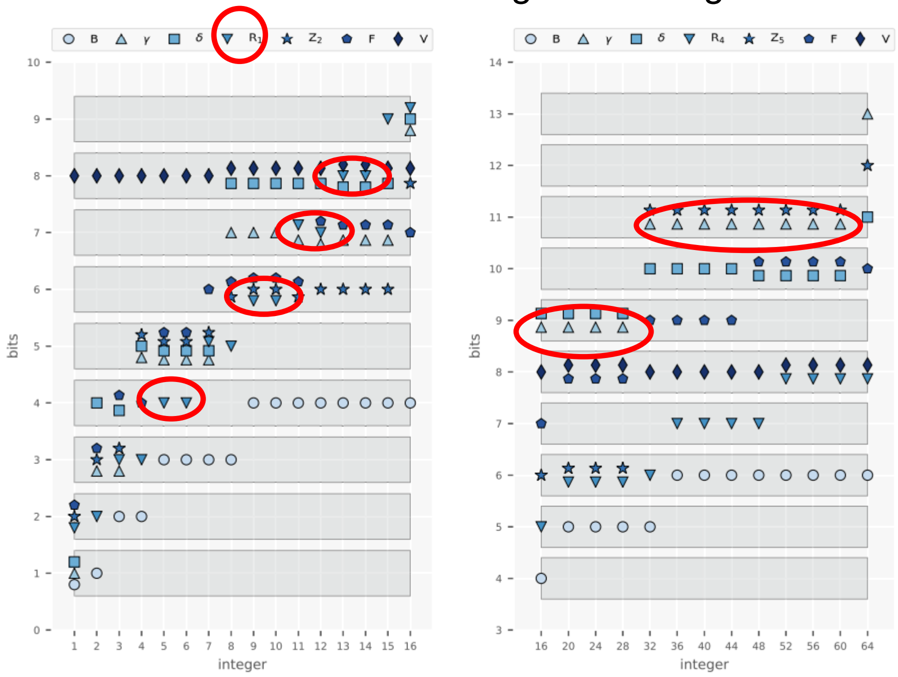
- Gamma encoding of the length of the binary representation followed by $(|bin(x)| - 1)$ bits from $bin(x)$.
- Replace first part in Gamma by $\gamma(|bin(x)|)$.
- $|C(x)| = |\gamma(|bin(x)|)| + |bin(x)| - 1$.
- Optimal when $\mathbb{P}(x) \approx 1/(2x(\log_2 x)^2)$.

x	$\boldsymbol{\delta}(x)$
1	0.
2	100.0
3	100.1
4	101.00
5	101.01
6	101.10
7	101.11
8	11000.000

Integer Encoding

Encoding	Optimal when $\mathbb{P}(x) \approx$
Unary	$1/2^x$
Binary	$1/2^k$
Gamma	$1/(2x^2)$
Delta	$\mathbf{1/(2x(\log_2 x)^2)}$
Golumb	$p(1-p)^{x-1}$
Rice	$\mathbf{p(1-p)^{x-1}}$
Zeta	$1/(\zeta(\alpha)x^\alpha)$
Fibonnaci	$1/(2x^{\frac{1}{log_2\phi}}) \approx 1/(2x^{1.44})$
VByte	$\sqrt[7]{1/x^8}$
SC-Dense	$(s+c)^{-k(x)}$

Codeword Length Per Integer



Golomb Encoding

- Unary encoding of quotient(q) followed by binary codeword for remainder(r) with parameter $b > 1$.
- Optimal when $\mathbb{P}(x) = p(1-p)^{x-1}$ (geometric).

x	$G_2(x)$
1	0.0
2	0.1
3	10.0
4	10.1
5	110.0
6	110.1
7	1110.0
8	1110.1

Rice Encoding

- Special case of Golumb when $b = 2^k$.
- $|Rice_k(x)| = (x-1)/2^k + k + 1.$

x	$G_2(x)$
1	0.0
2	0.1
3	10.0
4	10.1
5	110.0
6	110.1
7	1110.0
8	1110.1

Integer Encoding

Encoding	Optimal when $\mathbb{P}(x) \approx$
Unary	$1/2^x$
Binary	$1/2^k$
Gamma	$1/(2x^2)$
Delta	$\mathbf{1/(2x(\log_2 x)^2)}$
Golumb	$p(1-p)^{x-1}$
Rice	$\mathbf{p(1-p)^{x-1}}$
Zeta	$1/(\zeta(\alpha)x^\alpha)$
Fibonnaci	$1/(2x^{\frac{1}{log_2\phi}}) \approx 1/(2x^{1.44})$
VByte	$\mathbf{\sqrt[7]{1/x^8}}$
SC-Dense	$(s+c)^{-k(x)}$

Codeword Length

Byte-aligned Encoding(VByte)

- Idea: align the bits used in codeword to byte or word lengths for faster reads.
- Most significant bit in each byte is reserved as a continuation bit, others used for data.
- Exploits SIMD instruction parallelisms and other hardware optimizations.
- OPT-Vbyte is a variation where continuation bits are stored separately.
- Optimal when $\mathbb{P}(x) \approx \sqrt[3]{1/x^4}$ or $\mathbb{P}(x) \approx \sqrt[7]{1/x^8}$.

List Compressors

List Compressors

- *Assume that integers are strongly ordered per list.

- **Idea: encode entire list instead of each single integer separately.**
- Theoretical lower bound on needed bits for encoding n integers from U:

$$\left\lceil \log_2 \binom{U}{n} \right\rceil = n\lceil \log_2(eU/n) \rceil - \Theta(n^2/U) - O(logn) \approx n\lceil \log_2(U/n) \rceil + 1.443n$$

- Can be approximated considering that lists feature **cluster of close integers**.
- Given the existence of these clusters can encode relative changes.
- Might help if we reorder docIDs to form larger clusters.

Binary Packing

- Partition sequence into blocks and encode them separately.
- Gaps between the integers can also be used.
- Size of blocks can be fixed but better to be of **variable** size.
- Descriptor is needed for each variable sized block.
- Blocks can further be hardware-aligned (SIMD-BP128).

Simple Encoders

- Idea: partition on fixed-memory units and pack as many integers in them as possible.
- Good compression and high decompression rates.
- Simple16 has 16 possible configurations and uses 32-bit words.
- QMX packs into 128 or 256-bit words and stores the selectors separately.

Table 6. Nine Different Ways of Packing Integers in a 28-Bit Segment as Used by Simple9

4-Bit Selector	Integers	Bits per Integer	Wasted Bits
0000	28	1	0
0001	14	2	0
0010	9	3	1
0011	7	4	0
0100	5	5	3
0101	4	7	0
0110	3	9	1
0111	2	14	0
1000	1	28	0

PForDelta(PFor) Encoders

Problem with Simple: space-inefficient when a block contains just one large value.

- Solution: pick a range $[b, b + 2^k - 1]$ that fits majority of the integers.
- Encode them with k bits.
- Mark other integers as exceptions and encode them separately with a different encoder algorithm.

[3, 4, 7, 21, 9, 12, 5, 17, 6, 2, 34]

b=2;
k=4;

[3, 4, 7, *, 9, 12, 5, *, 6, 2, *] – [21, 17, 34]

Elias-Fano Encoding

- Given n sorted integers from range $[1..U]$ - Universe.
- Split integers into $l = \lceil \log_2(U/n) \rceil$ low bits and $\lceil \log_2 U \rceil - l \approx \lfloor \log_2 n \rfloor$ high bits.
- Encode low bits separately with $n\lceil \log_2(U/n) \rceil$ size bitvector.
- Encode high bits separately with $2n$ bits:
  - Observe that $0 \le h_i \le n$. And that $h_{i-1} \le h_i$.
  - For each element, set $(h_i + i)$th bit to 1.
  - As a result we will get unary encodings of how many integers have h_i equal to particular value.

Theoretical LB:
$n\lceil \log_2(U/n) \rceil + 1.443n$

$$EF(S(n,U)) \le n\lceil \log_2(U/n) \rceil + 2n$$

Elias-Fano Encoding

Table 7. Example of Elias-Fano Encoding Applied to the Sequence $\mathcal{S} = [3, 4, 7, 13, 14, 15, 21, 25, 36, 38, 54, 62]$

$\mathcal{S}$	3	4	7	13	14	15	21	25	36	38		54	62
	0	0	0	0	0	0	0	0	1	1	1	1	1
high	0	0	0	0	0	0	1	1	0	0	0	1	1
	0	0	0	1	1	1	0	1	0	0	1	0	1
	0	1	1	1	1	1	1	0	1	1		1	1
low	1	0	1	0	1	1	0	0	0	1		1	1
	1	0	1	1	0	1	1	1	0	0		0	0
H		1110			1110		10	10	110		0	10	10
L		011.100.111			101.110.111		101	001	100.110			110	110

Elias-Fano Encoding: Random Access

Problem: how to decode a single individual integer?

- Get l_i low bits with direct access.
- Implement data structure to get $Select_b(i) = (ith\ bit\ set\ to$ b in H) in O(1).
- Then $h_i = Select_1(i) - i$.
- Concatenate l_i and h_i to get S_i.
- Runs in O(1).

Elias-Fano Encoding: Successor Queries

Problem: how to get smallest $y \geq x$ for some x?

- Let h_x be the high bits of x.
- Set $i = Select_o(h_x) - h_x + 1$ and j $= Select_o(h_x + 1) - h_x$.
- $[i..j]$ interval is where y must be.
- Do binary search.
- Runs in $O(1 + \log(U/n))$.

Elias-Fano Encoding: Partitioning by Cardinality(PEF)

Observation: in the inverted index integers are clustered.

- Partition into k blocks of variable length
- On the first level encode with EF (1)$\{U_1, .., U_k\}$ upper bounds of the blocks and (2)prefix-summed sequence of sizes of blocks.
- On the second level encode the blocks themselves.
- Suppose a block with size b and universe M:
    1. If $b = M$ – each element appears exactly once nothing to encode on the 2^{nd} level.
    2. If $b > M/4$ – since $EF(b, M) > M$ use characteristic encoding of size M.
    3. If $b \leq M/4$ – use EF on the 2nd level.
- It can be shown that using DP to determine blocks sizes is only $(1 + \epsilon)$ away from the optimal. But gets worse if ϵ is fixed.

Elias-Fano Encoding: Partitioning by Universe

Observation: high and low bit split can be chosen arbitrarily.

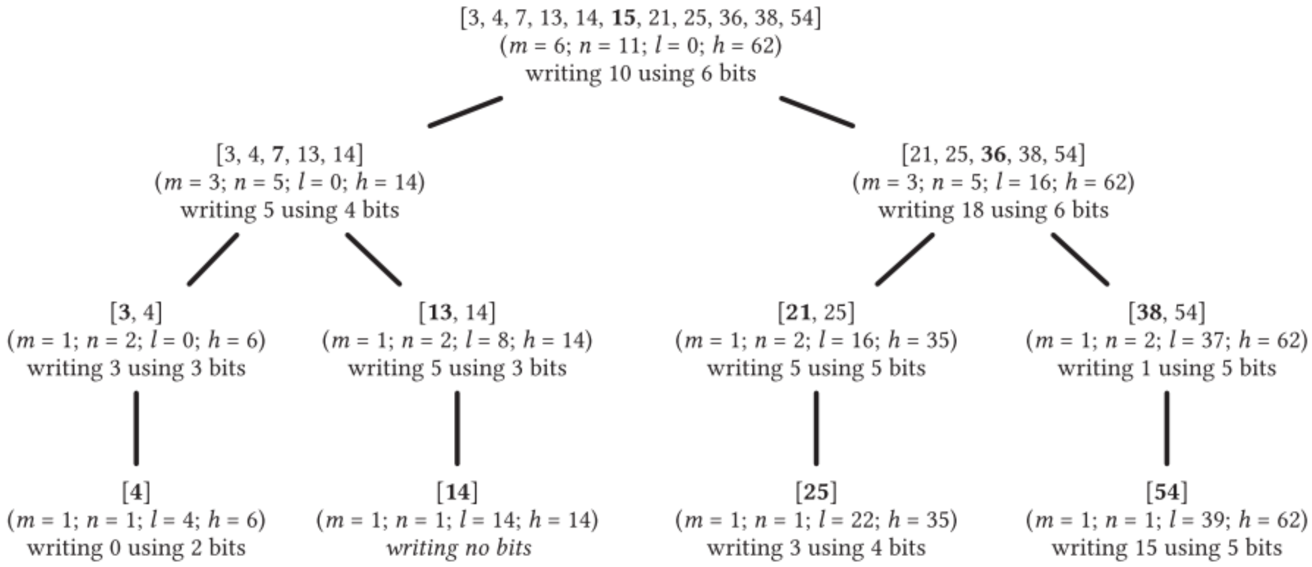
- **Roaring:** partition $U(2^{32})$ into chunk spanning 2^{16} values each:
  1. If a chunk is *sparse* (less than 2^{12} elements), encode as a sorted array of 16-bit integers.
  2. If a chunk is *dense* (more than 2^{12} elements), encode as a bitmap.
  3. If a chunk is *full* (2^{16} elements), encode implicitly.
- **Slicing:** similar to Roaring but continue encoding recursively if the chunk is *sparse.*

Binary Interpolative Code (BIC)

**Remember: strongly sorted sequence of clustered integers.*

- Idea: fully use the clustering prior of the integers in the index, by squashing together any runs of consecutive integers.
- Recursively divide the index and the value range in half while encoding the middle element with as little amount of bits as possible
- In particular in a given interval $S[i..j]$ with $l \leq S[i]$ and $S[j] \leq h$:
  1. Encode $S[(i+j)/2] - l - m + 1$ using $\lceil \log_2(h - l - j + i) \rceil$ bits.
  2. Continue encoding of $S[i..(i+j)/2 - 1]$ and $S[(i+j)/2 + 1..j]$ recursively.
  3. If $l + j - i = h$ holds, stop recursion and encode implicitly.

Binary Interpolative Code (BIC)



Entropy Encodings

Usually Good average codeword length, but can not compete with other methods.

- Huffman: Maintain a candidate set of tree and each step merge trees with lowest weight. Assign codewords based on the symbol's location in the eventual tree. Let L be average Huffman codeword length:
  - L is minimum possible among all the prefix-free encodings.
  - $H_0 \leq L < H_0 + 1$ where H_0 bits is the entropy of the system.
- Arithmetic: partition [0,1) interval to proportional length of system probabilities, pick first interval and recursively partition it. Eventually emit real number x from $[l_n, r_n)$.
  - Requires infinite precision arithmetic but can be approximated.
  - Takes at most $nH_0 + 2$ bits to encode entire sequence. In practice $nH_0 + 2n/100$ bits.
- Asymmetric Numeral Systems(ANS): Generate a frame from the sequence symbols with retaining the same probabilities. To encode start from column 0 and move to the column corresponding to the first symbol in the sequence. Continue the process emitting column number along the way.

Full Index Compressors

Clustered

- Group clusters of the lists sharing many integers.
- All lists in the cluster are then encoded with respect to the reference list.
- Used PEF for such encoding.

ANS based

- Universe can be very large even if only gaps are taken into account.
- Pre-process input list to a sequence of bytes.
- Then apply a combination of VByte and ANS.

Dictionary based(DINT)

- Store most frequent 2^b patterns in dictionary for some b.
- Use this dictionary to encode subsequences of gaps.
- Can be further optimized if we take advantage of the presence of runs of 1s in codeword modelling.

Dictionary-based Coding

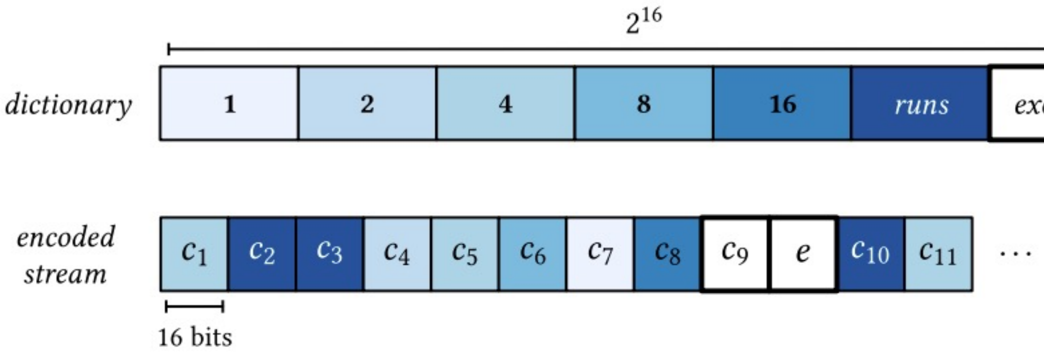


Fig. 6. A dictionary-based encoded stream example, where dictionary entries corresponding to $\{1, 2, 4, 8, 16\}$-long integer patterns, runs, and exceptions are labeled with different shades. Once provision has been made for such a dictionary structure, a sequence of gaps can be modeled as a sequence of codewords $\{c_k\}$, each being a reference to a dictionary entry, as represented with the *encoded stream* in the picture. Note that, for example, codeword c_9 signals an exception, and therefore the next symbol e is decoded using an escape mechanism.

Experimentations

Experimental Setting

- Machine: $Intel\, i9 - 9900K\,(@3.6Ghz), 64GB$ *DDR3 RAM, Running Linux 5 (64bit)*
- Code written in C++ with the highest optimization enabled:
  - Flags *-03* and *–march=native*
- Datasets:

(a) Basic Statistics

	Gov2	ClueWeb09	CCNews
Lists	39,177	96,722	76,474
Universe	24,622,347	50,131,015	43,530,315
Integers	5,322,883,266	14,858,833,259	19,691,599,096
Entropy of the gaps	3.02	4.46	5.44
$\lceil \log_2 \rceil$ of the gaps	1.35	2.28	2.99

(b) TREC 2005/06 Queries

	Gov2	ClueWeb09	CCNews
Queries	34,327	42,613	22,769
2 terms	32.2%	33.6%	37.5%
3 terms	26.8%	26.5%	27.3%
4 terms	18.2%	17.7%	16.8%
5+ terms	22.8%	22.2%	18.4%

Experimental Methodology

- Data structure is a memory mapped from the file.
- Warm-up run is executed before the experiments are run.
- Testing on sequential reads.
- Queries consist of randomly chosen 1000 samples of intersection(AND) and union(OR) queries consisting of terms from 2 to 5+.
- Average run time reported among 3 runs of the same experiment.
- What to watch out for:
  - **Space Usage:** measured in number of bits per integer *bits/int*.
  - **Access Time:** sequential or random. Measured in *ns/int*.

Tested Algorithms

Table 9. Different Tested Index Representations

	Method	Partitioned by	SIMD	Alignment	Description
Variable Byte	VByte	Cardinality	Yes	Byte	Fixed-size partitions of 128
Optimized VByte	Opt-VByte	Cardinality	Yes	Bit	Variable-size partitions
Interpolative	BIC	Cardinality	No	Bit	Fixed-size partitions of 128
Delta	δ	Cardinality	No	Bit	Fixed-size partitions of 128
Rice	Rice	Cardinality	No	Bit	Fixed-size partitions of 128
Elias-Fano	PEF	Cardinality	No	Bit	Variable-size partitions
Dictionary based	DINT	Cardinality	No	16-bit word	Fixed-size partitions of 128
PForDelta	Opt-PFor	Cardinality	No	32-bit word	Fixed-size partitions of 128
Simple	Simple16	Cardinality	No	64-bit word	Ffixed-size partitions of 128
Simple	QMX	Cardinality	Yes	128-bit word	Fixed-size partitions of 128
Elias-Fano	Roaring	Universe	Yes	byte	Single span
Elias-Fano	Slicing	Universe	Yes	byte	Multi-span

Space Usage and Sequential Decoding Speed

Table 11. Space Effectiveness in Total GiB and Bits per Integer, and Nanoseconds per Decoded Integer

	Method	Gov2			ClueWeb09			CCNews		
		GiB	Bits/int	ns/int	GiB	Bits/int	ns/int	GiB	bits/int	ns/int
Variable Byte	VByte	5.46	8.81	0.96	15.92	9.20	1.09	21.29	9.29	1.03
Optimized VByte	Opt-VByte	2.41	3.89	0.73	9.89	5.72	0.92	14.73	6.42	0.72
Interpolative	BIC	1.82	2.94	5.06	7.66	4.43	6.31	12.02	5.24	6.97
Delta	δ	2.32	3.74	3.56	8.95	5.17	3.72	14.58	6.36	3.85
Rice	Rice	2.53	4.08	2.92	9.18	5.31	3.25	13.34	5.82	3.32
Elias-Fano	PEF	1.93	3.12	0.76	8.63	4.99	1.10	12.50	5.45	1.31
Dictionary based	DINT	2.19	3.53	1.13	9.26	5.35	1.56	14.76	6.44	1.65
PForDelta	Opt-PFor	2.25	3.63	1.38	9.45	5.46	1.79	13.92	6.07	1.53
Simple	Simple16	2.59	4.19	1.53	10.13	5.85	1.87	14.68	6.41	1.89
Simple	QMX	3.17	5.12	0.80	12.60	7.29	0.87	16.96	7.40	0.84
Elias-Fano	Roaring	4.11	6.63	0.50	16.92	9.78	0.71	21.75	9.49	0.61
Elias-Fano	Slicing	2.67	4.31	0.53	12.21	7.06	0.68	17.83	7.78	0.69

Space Usage

BIC for the Win!
PEF Close 2nd

VBYTE and ROARING have struggled.

Table 11. Space Effectiveness in Total GiB and Bits per Integer, and Nanoseconds per Decoded Integer

	Method	Gov2 3.02			ClueWeb09 4.46			CCNews 5.44		
		GiB	Bits/int	ns/int	GiB	Bits/int	ns/int	GiB	bits/int	ns/int
Variable Byte	VByte	5.46	8.81	0.96	15.92	9.20	1.09	21.29	9.29	1.03
Optimized VByte	Opt-VByte	2.41	3.89	0.73	9.89	5.72	0.92	14.73	6.42	0.72
Interpolative	BIC	1.82	2.94	5.06	7.66	4.43	6.31	12.02	5.24	6.97
Delta	δ	2.32	3.74	3.56	8.95	5.17	3.72	14.58	6.36	3.85
Rice	Rice	2.53	4.08	2.92	9.18	5.31	3.25	13.34	5.82	3.32
Elias-Fano	PEF	1.93	3.12	0.76	8.63	4.99	1.10	12.50	5.45	1.31
Dictionary based	DINT	2.19	3.53	1.13	9.26	5.35	1.56	14.76	6.44	1.65
PForDelta	Opt-PFor	2.25	3.63	1.38	9.45	5.46	1.79	13.92	6.07	1.53
Simple	Simple16	2.59	4.19	1.53	10.13	5.85	1.87	14.68	6.41	1.89
Simple	QMX	3.17	5.12	0.80	12.60	7.29	0.87	16.96	7.40	0.84
Elias-Fano	Roaring	4.11	6.63	0.50	16.92	9.78	0.71	21.75	9.49	0.61
Elias-Fano	Slicing	2.67	4.31	0.53	12.21	7.06	0.68	17.83	7.78	0.69

Decoding Speed

ROARING and SLICING are crushing it!!

BIC, DELTA and RICE are all struggling

Table 11. Space Effectiveness in Total GiB and Bits per Integer, and Nanoseconds per Decoded Integer

	Method	Gov2			ClueWeb09			CCNews		
		GiB	Bits/int	ns/int	GiB	Bits/int	ns/int	GiB	bits/int	ns/int
Variable Byte	VByte	5.46	8.81	0.96	15.92	9.20	1.09	21.29	9.29	1.03
Optimized VByte	Opt-VByte	2.41	3.89	0.73	9.89	5.72	0.92	14.73	6.42	0.72
Interpolative	BIC	1.82	2.94	5.06	7.66	4.43	6.31	12.02	5.24	6.97
Delta	δ	2.32	3.74	3.56	8.95	5.17	3.72	14.58	6.36	3.85
Rice	Rice	2.53	4.08	2.92	9.18	5.31	3.25	13.34	5.82	3.32
Elias-Fano	PEF	1.93	3.12	0.76	8.63	4.99	1.10	12.50	5.45	1.31
Dictionary based	DINT	2.19	3.53	1.13	9.26	5.35	1.56	14.76	6.44	1.65
PForDelta	Opt-PFor	2.25	3.63	1.38	9.45	5.46	1.79	13.92	6.07	1.53
Simple	Simple16	2.59	4.19	1.53	10.13	5.85	1.87	14.68	6.41	1.89
Simple	QMX	3.17	5.12	0.80	12.60	7.29	0.87	16.96	7.40	0.84
Elias-Fano	Roaring	4.11	6.63	0.50	16.92	9.78	0.71	21.75	9.49	0.61
Elias-Fano	Slicing	2.67	4.31	0.53	12.21	7.06	0.68	17.83	7.78	0.69

Best Of Both Worlds

PEF and DINT have the best balance.

BIC and ROARING are the extremes.

Table 11. Space Effectiveness in Total GiB and Bits per Integer, and Nanoseconds per Decoded Integer

	Method	Gov2			ClueWeb09			CCNews		
		GiB	Bits/int	ns/int	GiB	Bits/int	ns/int	GiB	bits/int	ns/int
Variable Byte	VByte	5.46	8.81	0.96	15.92	9.20	1.09	21.29	9.29	1.03
Optimized VByte	Opt-VByte	2.41	3.89	0.73	9.89	5.72	0.92	14.73	6.42	0.72
Interpolative	BIC	1.82	2.94	5.06	7.66	4.43	6.31	12.02	5.24	6.97
Delta	δ	2.32	3.74	3.56	8.95	5.17	3.72	14.58	6.36	3.85
Rice	Rice	2.53	4.08	2.92	9.18	5.31	3.25	13.34	5.82	3.32
Elias-Fano	PEF	1.93	3.12	0.76	8.63	4.99	1.10	12.50	5.45	1.31
Dictionary based	DINT	2.19	3.53	1.13	9.26	5.35	1.56	14.76	6.44	1.65
PForDelta	Opt-PFor	2.25	3.63	1.38	9.45	5.46	1.79	13.92	6.07	1.53
Simple	Simple16	2.59	4.19	1.53	10.13	5.85	1.87	14.68	6.41	1.89
Simple	QMX	3.17	5.12	0.80	12.60	7.29	0.87	16.96	7.40	0.84
Elias-Fano	Roaring	4.11	6.63	0.50	16.92	9.78	0.71	21.75	9.49	0.61
Elias-Fano	Slicing	2.67	4.31	0.53	12.21	7.06	0.68	17.83	7.78	0.69

AND Queries

Table 12. Milliseconds Spent per AND Query by Varying the Number of Query Terms

	Method	Gov2					ClueWeb09					CCNews				
		2	3	4	5+	avg.	2	3	4	5+	avg.	2	3	4	5+	avg.
Variable Byte	VByte	2.2	2.8	2.7	3.3	2.8	10.2	12.1	13.7	13.9	12.5	14.0	22.4	19.7	21.9	19.5
Optimized VByte	Opt-VByte	2.8	3.1	2.8	3.2	3.0	12.2	13.3	14.0	13.6	13.3	16.0	23.2	19.6	20.3	19.8
Interpolative	BIC	6.8	9.7	10.4	13.2	10.0	31.7	44.2	51.5	53.8	45.3	45.6	79.7	76.9	88.8	72.8
Delta	δ	4.6	6.3	6.5	8.2	6.4	20.9	28.3	33.5	34.5	29.3	28.6	50.9	48.0	55.6	45.8
Rice	Rice	4.1	5.6	5.8	7.3	5.7	19.2	25.7	30.2	31.1	26.6	26.5	46.5	43.5	50.1	41.6
Elias-Fano	PEF	2.5	3.1	2.8	3.2	2.9	12.3	13.5	14.4	13.8	13.5	17.2	24.6	21.0	21.9	21.2
Dictionary based	DINT	2.5	3.3	3.3	4.1	3.3	11.9	14.6	16.5	17.1	15.0	16.9	27.3	24.6	28.1	24.2
PForDelta	Opt-PFor	2.6	3.5	3.5	4.3	3.5	12.8	15.9	18.0	18.3	16.3	16.6	27.2	24.3	27.1	23.8
Simple	Simple16	2.8	3.7	3.7	4.6	3.7	12.8	16.3	18.4	18.9	16.6	17.6	28.8	26.3	29.5	25.5
Simple	QMX	2.0	2.6	2.5	3.0	2.5	9.6	11.5	13.0	13.1	11.8	13.3	21.5	18.8	20.8	18.6
Elias-Fano	Roaring	0.3	0.5	0.7	0.8	0.6	1.5	2.5	3.1	4.3	2.9	1.1	2.0	2.6	4.1	2.5
Elias-Fano	Slicing	0.3	1.0	1.2	1.6	1.0	1.5	4.5	5.4	6.7	4.5	1.8	4.3	5.1	6.0	4.3

OR Queries

Table 13. Milliseconds Spent per OR Query by Varying the Number of Query Terms

	Method	Gov2					ClueWeb09					CCNews				
		2	3	4	5+	avg.	2	3	4	5+	avg.	2	3	4	5+	avg.
Variable Byte	VByte	6.8	24.4	54.7	131.7	54.4	20.1	71.3	156.0	379.5	156.7	24.4	94.5	178.8	391.4	172.3
Optimized VByte	Opt-VByte	11.0	35.7	77.4	176.0	75.0	31.3	101.4	213.4	500.1	211.6	36.4	128.0	232.0	510.4	226.7
Interpolative	BIC	16.7	50.3	105.0	238.8	102.7	49.9	145.3	290.4	668.2	288.4	64.4	193.8	332.6	692.5	320.8
Delta	δ	12.6	40.8	87.9	202.5	85.9	34.9	112.9	236.7	557.7	235.6	42.2	144.9	263.8	571.3	255.5
Rice	Rice	13.4	43.1	93.3	211.3	90.3	36.8	118.2	248.5	576.6	245.0	43.6	149.3	270.5	585.6	262.2
Elias-Fano	PEF	10.2	33.0	71.7	164.2	69.8	31.1	99.7	208.5	492.3	207.9	37.6	127.5	232.6	507.1	226.2
Dictionary based	DINT	8.5	28.5	63.7	147.6	62.1	24.9	84.1	178.8	424.3	178.0	30.6	109.2	200.4	432.7	193.2
PForDelta	Opt-PFor	8.9	31.1	69.4	161.4	67.7	27.0	90.8	194.0	453.5	191.3	31.3	113.2	209.0	447.2	200.2
Simple	Simple16	7.8	26.2	58.3	138.2	57.6	23.7	78.0	165.5	394.7	165.5	28.7	101.5	185.3	397.8	178.4
Simple	QMX	6.6	23.8	53.4	128.1	53.0	19.7	70.0	153.2	377.9	155.2	24.0	92.6	175.2	382.4	168.6
Elias-Fano	Roaring	1.2	2.8	4.3	6.4	3.7	4.7	9.0	12.0	15.7	10.3	3.8	7.6	10.5	15.1	9.2
Elias-Fano	Slicing	1.3	4.0	6.3	9.2	5.2	5.0	12.8	18.1	25.3	15.3	5.8	12.9	17.3	23.0	14.8

Space/Time Trade-Offs

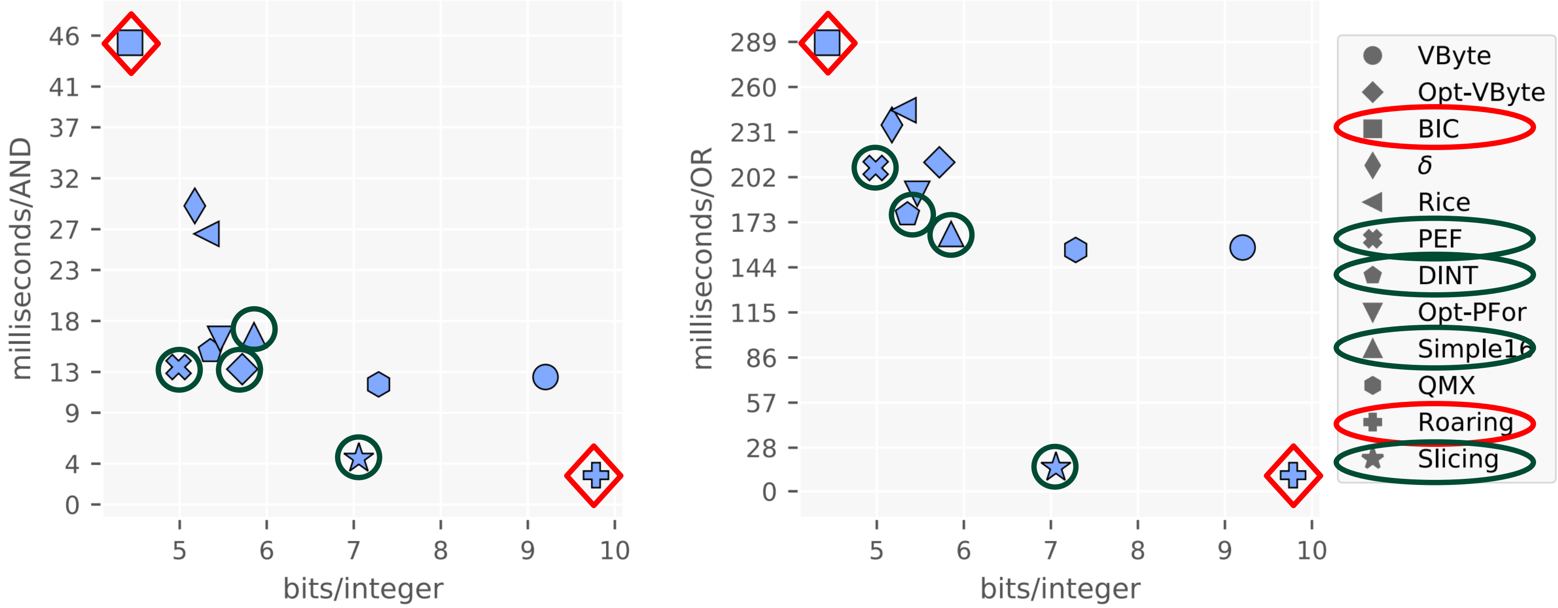


Fig. 7. Space/time trade-off curves for the ClueWeb09 dataset.

Final Thoughts

- If you want:
  - Speed: Roaring.
  - Compression effectiveness: BIC.
  - Best of both Worlds: PEF, DINT or Slicing.

- Try to utilize SIMD and aligning if possible to get better performance!

- How Zeta or Fibonacci would perform on Inverted Index?

Acknowledgments

- Giulio Ermano Pibiri and Rossano Venturini.
- Professors Charles and Julian.
- All the authors of various algorithms described above.

Thank you

Appendix

Exponential Golomb Encoding

- Define $B = [0, 2^k, \sum_{i=0}^{1} 2^{k+i}, \sum_{i=0}^{2} 2^{k+i}, \dots]$.
- Unary encoding of bucket identifier followed by binary encoding of bucket specific offset.
- $|C(x)| = 2h + 1$ where $B[h] < x \leq B[h+1]$.

x	$ExpG_2(x)$
1	0.00
2	0.01
3	0.10
4	0.11
5	10.000
6	10.001
7	10.010
8	10.011

Zeta Encoding

- Exponential Golumb with buckets: $[0, 2^k - 1, 2^{2k} - 1, 2^{3k} - 1 \ldots]$.
- Unary encoding of bucket identifier followed by a minimal binary codeword for bucket specific offset.
- Z_1 coincides with $ExpG_0$ and Gamma.
- Optimal when $\mathbb{P}(x) = 1/(\zeta(\alpha)x^\alpha)$ distributed according to a power law and $\zeta()$ is Riemann zeta function.

x	$Z_2(x)$
1	0.0
2	0.10
3	0.11
4	10.000
5	10.001
6	10.010
7	10.011
8	10.1000

Fibonacci Encoding

- Encode x as binary of which Fibonacci numbers are used in unique some representation
- Generate Lexicographic Codewords of same lengths
- Optimal when $\mathbb{P}(x) \approx 1/(2x^{\frac{1}{log_2 \phi}}) \approx 1/(2x^{1.44})$

Table 4. Integers 1..8 as Represented with Fibonacci-Based Codes

(a) "Original" Codewords

x	F(x)					
1	1	**1**				
2	0	1	**1**			
3	0	0	1	**1**		
4	1	0	1	**1**		
5	0	0	0	1	**1**	
6	1	0	0	1	**1**	
7	0	1	0	1	**1**	
8	0	0	0	0	1	**1**
F_i	1	2	3	5	8	13

(b) Lexicographic Codewords

x	F(x)					
1	0	0				
2	0	1	0			
3	0	1	1	0		
4	0	1	1	1		
5	1	0	0	0	0	
6	1	0	0	0	1	
7	1	0	0	1	0	
8	1	0	0	1	1	0

SC-Dense Encoding

- Have c continuers and s stoppers, where $c + s = 2^8$
- Can be better adapt for the distribution of the words
- $|C(x)| = k(x)\lceil log_2(s + c)\rceil$ where $k(x)$ is number of words needed
- Optimal when $\mathbb{P}(x) \approx (s + c)^{-k(x)}$

x	SC(4, 4, x)	SC(5, 3, x)
1	000	000
2	001	001
3	010	010
4	011	011
5	100.000	100
6	100.001	101.000
7	100.010	101.001
8	100.011	101.010
9	101.000	101.011
10	101.001	101.100

x	SC(4, 4, x)	SC(5, 3, x)
11	101.010	110.000
12	101.011	110.001
13	110.000	110.010
14	110.001	110.011
15	110.010	110.100
16	110.011	111.000
17	111.000	111.001
18	111.001	111.010
19	111.010	111.011
20	111.011	111.100

Huffman Coding

symbols	weights	lengths	codewords
2	8	2	00
5	7	2	01
6	2	3	100
7	2	3	101
1	2	4	1100
3	2	4	1101
4	1	4	1110
8	1	4	1111

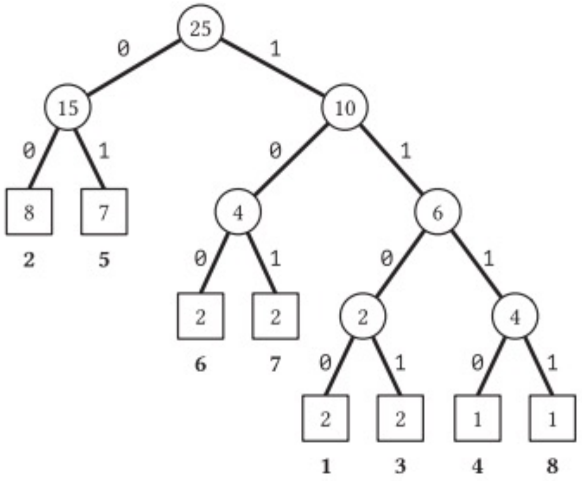


Fig. 5. An example of Huffman coding applied to a sequence of size 25 with symbols 1..8 and associated weights [2, 8, 2, 1, 7, 2, 2, 1].

Arithmetic Numeral Systems(ANS)

- Generate a frame from the sequence symbols with retaining the same probabilities
- To encode start from column 0 and move to the column corresponding to the first symbol in the sequence. Continue the process emitting column number along the way.

(a)

Σ	ℙ	Codes									
a	1/2	1	2	3	7	8	9	13	14	15	19
b	1/3	4	5	10	11	16	17	22	23	28	29
c	1/6	6	12	18	24	30	36	42	48	54	60
		0	1	2	3	4	5	6	7	8	9

(b)

Σ	ℙ	Codes									
a	1/2	2	4	6	8	10	12	14	16	18	20
b	1/4	3	7	11	15	19	23	27	31	35	39
c	1/4	1	5	9	13	17	21	25	29	33	37
		0	1	2	3	4	5	6	7	8	9