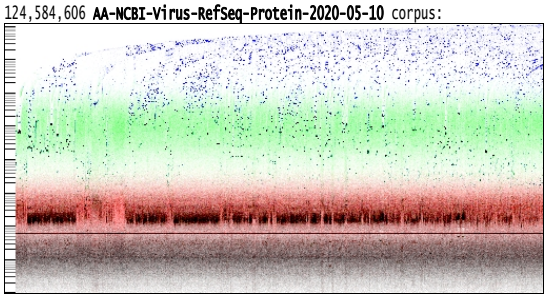
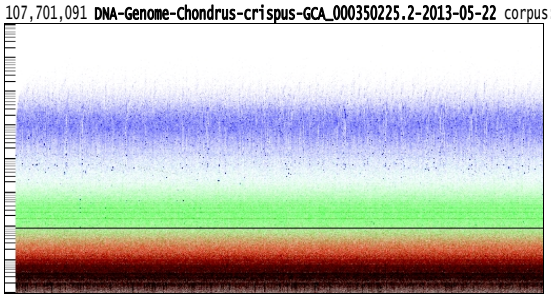
TEXTORAMIC (DNA) Decompression Showdown — LZO vs ZLIB vs ZSTD vs NAF vs pbzip2 vs LIZARD vs Nakamichi 'Ryuugan'

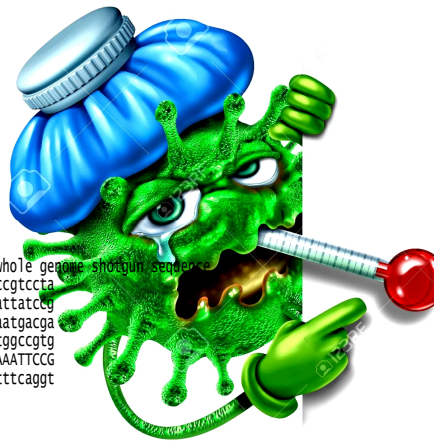
124,584,606 **AA-NCBI-Virus-RefSeq-Protein-2020-05-10** corpus:



```
G:\SCB>type AA-NCBI-Virus-RefSeq-Protein-2020-05-10
>YP_009755832 |ORF1ab polyprotein [Rodent coronavirus]
MPKVNKYGLELHWAPEFPWMFEDTEEKLGDPCSSEVGMVCPTTEQKLGTSEIFSKNHYMA
DCRRLTTRDCAAPLALFSDMVITTGPDDMVSVIEEAFKSQKAVFVVPPRRMSLEECYKVG
CAPRGWSMGMFSRKPECYTGVCGKGHKHVTNELFVIDPEGVCFGAGIFLGWVVTVDVIPK
RALYKLEPWVLYLRKRGYKGAHVASHVDWFLHGSEVYDPYYYGAMDIVQDDPKGKFSSKA
YAMLRGYRGLKPILYVDQYGCDYTGSLAEGLTDYADFSLAEMKELFPVWSQSLDYEVVVA
WHTVRDARFVMKLQTCATIRSIEYVGEPMDDIQGSEILVKEPVHVLSADAIVSKLPKLID
...
```

107,701,091 **DNA-Genome-Chondrus-crispus-GCA_000350225.2-2013-05-22** corpus:





```
G:\SCB>type DNA-Genome-Chondrus-crispus-GCA_000350225.2-2013-05-22
>HG002051.1 Chondrus crispus Stackhouse genomic scaffold, scaffold_490, whole genome shotgun sequence
CGTCCGAACCTCCTATCTCTCGTAGATCGGGCGAGATGTGcacagcttcctgatttaccaacggtccgtcatccgtcccta
ttctttgcttgcgcggcgggttcttctcgttcccctcttgattgaggaactgtagcgcttgccgtagcccgattatccg
cgtgtagtccatgggactagtggcttattcgactctatctcgcgacacgcgctcttcttggtcgccgatggcaatgacga
ccgtattgtcagttgactccttcacttgatacggacccgtagccacttgtgctaacttatgtttcggctcggtggccgtg
ccttgctcttttccgaaggaatacgtatgatcctgaagggatgtcgtACTTCGGCCTGCGAAGTCTTGAATCGAAATTCCG
CTTCTACCGTGCCTTTTCCTTGCGCAGACTCTTATCGGCACTTCTCATcaagctctccaaccacgactgccatttcaggt
...
```

Sequence Compression Benchmark - http://kirr.dyndns.org/sequence-compression-benchmark/

Speed of copy-cat: 122.4/0.034=3600MB/s
Speed of nakamichi: 122.4/0.22=556MB/s

// Sorted by 'D-Time (s)'
D:\Downloads>sort 122.txt /+49

Compressor	Size (B)	C-Time (s)	**D-Time (s)**	DatasetName
copy-cat	122442166	0.084	**0.034**	NCBI Virus RefSeq Protein
lz4-9	78256616	2.944	**0.11**	NCBI Virus RefSeq Protein
lizard-19	78231873	10.75	**0.117**	NCBI Virus RefSeq Protein
lizard-49	52242960	40.06	**0.184**	NCBI Virus RefSeq Protein
zstd-22-4t	32931363	117.2	**0.217**	NCBI Virus RefSeq Protein
nakamichi	51685771	5409	**0.22**	NCBI Virus RefSeq Protein
naf-22	29879592	115.4	**0.257**	NCBI Virus RefSeq Protein
lzop-9	75886220	11.12	**0.388**	NCBI Virus RefSeq Protein
pigz-11-4t	54067580	69.51	**0.424**	NCBI Virus RefSeq Protein
pbzip2-9-4t	53843897	2.652	**0.49**	NCBI Virus RefSeq Protein
brotli-11w30	30840740	306.8	**0.609**	NCBI Virus RefSeq Protein
brieflz-opt	68702413	19.18	**0.663**	NCBI Virus RefSeq Protein
zpaq-2-4t	48805968	33.11	**0.998**	NCBI Virus RefSeq Protein
gzip-9	60305267	4.228	**1.023**	NCBI Virus RefSeq Protein
bsc-b1024cfe2	40904228	26.53	**2.941**	NCBI Virus RefSeq Protein
bzip2-9	53818154	10.16	**4.951**	NCBI Virus RefSeq Protein

// Sorted by 'Size (B)'
D:\Downloads>sort 122.txt /+16

Compressor	**Size (B)**	C-Time (s)	D-Time (s)	DatasetName
naf-22	**29879592**	115.4	0.257	NCBI Virus RefSeq Protein
brotli-11w30	**30840740**	306.8	0.609	NCBI Virus RefSeq Protein
zstd-22-4t	**32931363**	117.2	0.217	NCBI Virus RefSeq Protein
bsc-b1024cfe2	**40904228**	26.53	2.941	NCBI Virus RefSeq Protein
zpaq-2-4t	**48805968**	33.11	0.998	NCBI Virus RefSeq Protein
nakamichi	**51685771**	5409	0.22	NCBI Virus RefSeq Protein
lizard-49	**52242960**	40.06	0.184	NCBI Virus RefSeq Protein
bzip2-9	**53818154**	10.16	4.951	NCBI Virus RefSeq Protein
pbzip2-9-4t	**53843897**	2.652	0.49	NCBI Virus RefSeq Protein
pigz-11-4t	**54067580**	69.51	0.424	NCBI Virus RefSeq Protein
gzip-9	**60305267**	4.228	1.023	NCBI Virus RefSeq Protein
brieflz-opt	**68702413**	19.18	0.663	NCBI Virus RefSeq Protein
lzop-9	**75886220**	11.12	0.388	NCBI Virus RefSeq Protein
lizard-19	**78231873**	10.75	0.117	NCBI Virus RefSeq Protein
lz4-9	**78256616**	2.944	0.11	NCBI Virus RefSeq Protein
copy-cat	**122442166**	0.084	0.034	NCBI Virus RefSeq Protein

Speed of copy-cat: 106.4/0.032=3325MB/s
Speed of nakamichi: 106.4/0.156=682MB/s

// Sorted by 'D-Time (s)'
D:\Downloads>sort 106.txt /+49

Compressor	Size (B)	C-Time (s)	**D-Time (s)**	DatasetName
copy-cat	106387446	0.075	**0.032**	Chondrus crispus GCA_000350225.2
lz4-9	42521180	25.54	**0.086**	Chondrus crispus GCA_000350225.2
lizard-19	40593814	20.66	**0.088**	Chondrus crispus GCA_000350225.2
lizard-49	28706051	47.68	**0.122**	Chondrus crispus GCA_000350225.2
nakamichi	28629121	4588	**0.156**	Chondrus crispus GCA_000350225.2
naf-22	16005607	49.47	**0.172**	Chondrus crispus GCA_000350225.2
zstd-22-4t	18502420	106.4	**0.225**	Chondrus crispus GCA_000350225.2
lzop-9	39874060	82.53	**0.266**	Chondrus crispus GCA_000350225.2
pigz-11-4t	30540010	181.2	**0.306**	Chondrus crispus GCA_000350225.2
pbzip2-9-4t	30075983	2.163	**0.408**	Chondrus crispus GCA_000350225.2
brieflz-opt	33174419	32.5	**0.465**	Chondrus crispus GCA_000350225.2
brotli-11w30	17050457	316.5	**0.566**	Chondrus crispus GCA_000350225.2
gzip-9	32681727	37.56	**0.609**	Chondrus crispus GCA_000350225.2
bsc-b1024cfe2	21945288	18.43	**2.177**	Chondrus crispus GCA_000350225.2
bzip2-9	30062239	8.211	**3.946**	Chondrus crispus GCA_000350225.2
zpaq-2-4t	27929041	18.33	**0.869**	Chondrus crispus GCA_000350225.2

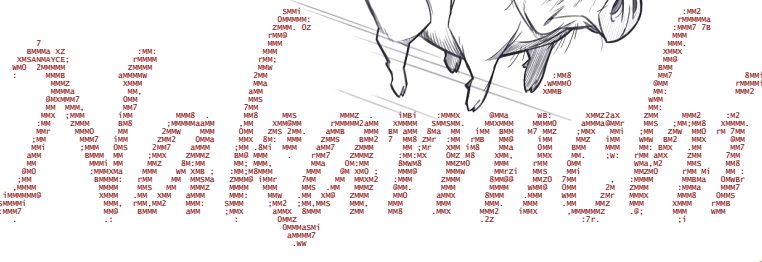
// Sorted by 'Size (B)'
D:\Downloads>sort 106.txt /+17

Compressor	**Size (B)**	C-Time (s)	D-Time (s)	DatasetName
naf-22	**16005607**	49.47	0.172	Chondrus crispus GCA_000350225.2
brotli-11w30	**17050457**	316.5	0.566	Chondrus crispus GCA_000350225.2
zstd-22-4t	**18502420**	106.4	0.225	Chondrus crispus GCA_000350225.2
bsc-b1024cfe2	**21945288**	18.43	2.177	Chondrus crispus GCA_000350225.2
zpaq-2-4t	**27929041**	18.33	0.869	Chondrus crispus GCA_000350225.2
nakamichi	**28629121**	4588	0.156	Chondrus crispus GCA_000350225.2
lizard-49	**28706051**	47.68	0.122	Chondrus crispus GCA_000350225.2
bzip2-9	**30062239**	8.211	3.946	Chondrus crispus GCA_000350225.2
pbzip2-9-4t	**30075983**	2.163	0.408	Chondrus crispus GCA_000350225.2
pigz-11-4t	**30540010**	181.2	0.306	Chondrus crispus GCA_000350225.2
gzip-9	**32681727**	37.56	0.609	Chondrus crispus GCA_000350225.2
brieflz-opt	**33174419**	32.5	0.465	Chondrus crispus GCA_000350225.2
lzop-9	**39874060**	82.53	0.266	Chondrus crispus GCA_000350225.2
lizard-19	**40593814**	20.66	0.088	Chondrus crispus GCA_000350225.2
lz4-9	**42521180**	25.54	0.086	Chondrus crispus GCA_000350225.2
copy-cat	**106387446**	0.075	0.032	Chondrus crispus GCA_000350225.2

Note1: Joergen Ibsen's BriefLZ 1.3.0 (https://github.com/jibsen/brieflz) is stronger than Nakamichi, inhere it uses its default window/block - 1MB - instead of its powerful 3600MB!
Note2: Since this CPU is from 4th Haswell family, Nakamichi decompresses slower in contrast to 7th generation which (as documented by Agner Fog) deals faster with overlapping writes/stores!
Note3: Usually, (for textual corpora) Lizard 49 is on par with Nakamichi, ratiowise and decompressionwise, as inhere.
Note4: Hmm, PIGZEE quadro-threaded is 2x slower, didn't know that.

Benchmark machine:
- CPU: dual Xeon E5-2643v3 (3.4 GHz, 6 cores), hyperthreading: off, L2 Cache: 256KB, L3 Cache: 20MB
- RAM: 128 GB DDR4-2133 ECC Registered
- Storage: 4 x 2 TB SSD, in RAID 0, XFS filesystem, block size: 4096 bytes (blockdev --getbsz)
- OS: Ubuntu 18.04.1 LTS, kernel: 4.15.0
- GCC: 7.4.0

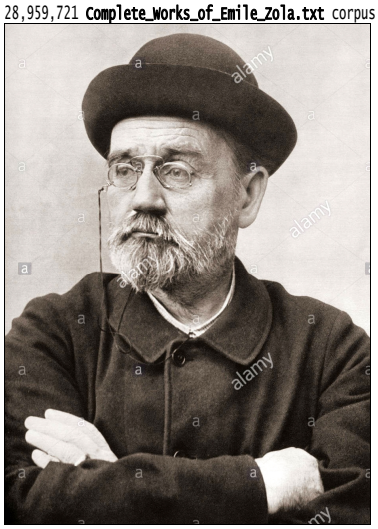
http://kirr.dyndns.org/sequence-compression-benchmark/?page=Commands#nakamichi
nakamichi-compress.pl --out {OUT} --hashsize 28 --treesize 81000 --treetype i



Nakamichi 'Dragoneye' LITE a.k.a. **Satanichi** highlights:
- The Zennish LZSS Microdeduplicator, written in plain C;
- The first Lempel-Ziv-Storer-Szymanski (LZSS) compressor using 1TB sliding window;
- It targets superfast decompression of huge/tera corpora of textual data;
- Compressing at Linear Rate due to Matchfinding based on B-trees only;
- The Internal/External B-trees order 3 (2 keys MAX) are brutally-optimized;
- Ability to deduplicate (as little as) 64 bytes long chunks 1TB backwards;
- Single-threaded Non-SIMD console tool written entirely by yours truly Sanmayce;
- SCALABLE! Gets faster when more Physical or/and External RAM is available;
- Dynamical (as command line parameters) selection of HASH & B-trees memory blocks;
- Compileable under Windows and Linux, most suitable to run on 3TB machines;
- Many thanks go to: Haruhiko Okumura, Nobuo Ito, Hamid Buzidi, Landon Noll.

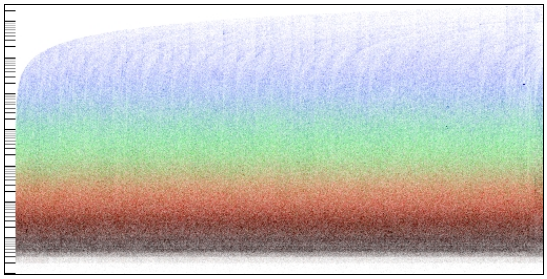
TEXTORAMIC Decompression Showdown — SNAPPY vs LZO vs ZLIB vs ZSTD vs LZTURBO vs OODLE vs LIZARD vs Nakamichi

28,959,721 **Complete_Works_of_Emile_Zola.txt** corpus crunched on i5-7200U 2.5GHz, 8GB DDR4 2133MHz:



中道乙成

```
E:\TEXTUAL_MADNESS_bare-minimum_2020_May_22>TIMER64
Satanichi_GCC730_64bit.exe "Complete_Works_of_Emile_Zola.txt" "Complete_Works_of_Emile_Zola.txt.Nakamichi" 20 6000 i
...
Leprechaun: Total Searches-n-Inserts Per Second: 29,577,033 SNIPS = 18,899,724,520 keys in 639 seconds
Leprechaun: RAM needed to house B-trees (relative to the file being ripped): 31N = 858MB
Leprechaun: RAM needed to build B-trees IN ONE PASS: (Target-Buffer = 59 MB) x 128 passes = 7,552MB
Note: In case of RAM in spades then may use compile option: -DSpeeduptuilding=7552
```



The struture of the corpus is shown in the diagram above. Each pixel represents a match between consecutive occurrences of a string. The color of the pixel represents the length of the match: black for 1 byte, red for 2, green for 4 and blue for 8. The horizontal axis represents the position of the second occurrence of the string. The vertical axis represents the distance back to the match on a logarithmic scale. (The image was generated by the fv program).
http://www.mattmahoney.net/dc/dce.html

```
E:\TEXTUAL_MADNESS_bare-minimum_2020_May_22>lzbench18 -c4 -i1,15 -o3 -esnappy/lizard,49/zlib,9/lzsse2,17/lzo1b
lzbench 1.8 (64-bit Windows)   Assembled by P.Skibinski
```

Compressor name	Compress.	Decompress.	Compr. size	Ratio	Filename
lizard 1.0 -49	1.33 MB/s	**705** MB/s	8579859	29.63	Complete_Works_of_Emile_Zola.txt
Nakamichi 2020-May-09		**894** MB/s	9129360		
libdeflate 1.3 -12	5.35 MB/s	**633** MB/s	10003305	34.54	Complete_Works_of_Emile_Zola.txt
lzsse2 2019-04-18 -17	5.80 MB/s	**3107** MB/s	10429480	36.01	Complete_Works_of_Emile_Zola.txt
zlib 1.2.11 -9	9.42 MB/s	**244** MB/s	10535533	36.38	Complete_Works_of_Emile_Zola.txt
lzo1b 2.10 -999	6.45 MB/s	**434** MB/s	11548104	39.88	Complete_Works_of_Emile_Zola.txt
snappy 2019-09-30	199 MB/s	**726** MB/s	17602552	60.78	Complete_Works_of_Emile_Zola.txt
memcpy	10697 MB/s	**10599** MB/s	28959721	100.00	Complete_Works_of_Emile_Zola.txt

```
E:\TEXTUAL_MADNESS_bare-minimum_2020_May_22>"turbobench_v18.05_-_build_04_May_2018" "Complete_Works_of_Emile_Z
```

C Size	ratio%	C MB/s	D MB/s	Name	File
5823873	20.1	7.85	**39**.09	lzturbo 59	Complete_Works_of_Emile_Zola.txt
7332690	25.3	1.01	**660**.23	lzturbo 39	Complete_Works_of_Emile_Zola.txt
7371336	25.5	0.34	**569**.20	oodle 139	Complete_Works_of_Emile_Zola.txt
8292795	28.6	1.02	**1037**.69	lzturbo 29	Complete_Works_of_Emile_Zola.txt
8644676	29.9	0.96	**1415**.29	oodle 99	Complete_Works_of_Emile_Zola.txt
9129360			**894**	Nakamichi 2020-May-09	
11835677	40.9	1.21	**3129**.77	lzturbo 19	Complete_Works_of_Emile_Zola.txt

```
E:\TEXTUAL_MADNESS_bare-minimum_2020_May_22>zstd-v1.4.5-win64.
-priority=rt "Complete_Works_of_Emile_Zola.txt"
Note : switching to real-time priority t...
 1#of_Emile_Zola.txt :  28959721 ->  11573790 (2.502), 211.1 MB/s , 819.8 MB/s
 2#of_Emile_Zola.txt :  28959721 ->  10607188 (2.730), 149.7 MB/s , 654.3 MB/s
 3#of_Emile_Zola.txt :  28959721 ->  10033930 (2.886), 126.4 MB/s , 637.5 MB/s
 4#of_Emile_Zola.txt :  28959721 ->   9846189 (2.941), 101.9 MB/s , 626.9 MB/s
 5#of_Emile_Zola.txt :  28959721 ->   9963350 (2.907),  59.8 MB/s , 603.0 MB/s
 6#of_Emile_Zola.txt :  28959721 ->   9653335 (3.000),  35.8 MB/s , 628.4 MB/s
 7#of_Emile_Zola.txt :  28959721 ->   9297068 (3.115),  25.6 MB/s , 673.6 MB/s
 8#of_Emile_Zola.txt :  28959721 ->   9168892 (3.158),  19.8 MB/s , 699.3 MB/s
Nakamichi 2020-May-09              9129360                          894 MB/s
 9#of_Emile_Zola.txt :  28959721 ->   9003448 (3.217),  11.9 MB/s , 716.2 MB/s
10#of_Emile_Zola.txt :  28959721 ->   8915100 (3.248),   8.48 MB/s , 689.9 MB/s
11#of_Emile_Zola.txt :  28959721 ->   8862947 (3.268),   6.89 MB/s , 663.6 MB/s
12#of_Emile_Zola.txt :  28959721 ->   8703955 (3.327),   3.58 MB/s , 666.0 MB/s
13#of_Emile_Zola.txt :  28959721 ->   8433917 (3.434),   3.46 MB/s , 704.2 MB/s
14#of_Emile_Zola.txt :  28959721 ->   8273679 (3.500),   2.75 MB/s , 590.7 MB/s
15#of_Emile_Zola.txt :  28959721 ->   8153859 (3.552),   2.15 MB/s , 510.3 MB/s
16#of_Emile_Zola.txt :  28959721 ->   7785662 (3.720),   2.33 MB/s , 666.8 MB/s
17#of_Emile_Zola.txt :  28959721 ->   7650462 (3.785),   1.83 MB/s , 568.4 MB/s
18#of_Emile_Zola.txt :  28959721 ->   7547800 (3.837),   1.55 MB/s , 492.4 MB/s
19#of_Emile_Zola.txt :  28959721 ->   7462798 (3.881),   1.46 MB/s , 454.1 MB/s
20#of_Emile_Zola.txt :  28959721 ->   7348531 (3.941),   1.32 MB/s , 576.5 MB/s
21#of_Emile_Zola.txt :  28959721 ->   7338696 (3.946),   1.30 MB/s , 575.2 MB/s
22#of_Emile_Zola.txt :  28959721 ->   7338651 (3.946),   1.27 MB/s , 575.3 MB/s
```

Notes:
- Latest available '**oo2core_8_win64.dll**' was used;
- **Oodle** Compressors: Kraken = 8 (Default); Leviathan = 13 (Best); Mermaid = 9 (Crazy fast); Selkie = 11 (Faste
- **Oodle** Levels: HyperFast = -4..-1; None = 0; SuperFast = 1; VeryFast = 2; Fast = 3; Normal = 4; Optimal1,2,3,
- The tweak value **spacespeedtradeoff**=[64..1024] "tweak size vs decode time" has not been played with!
- If you wanna see more performers/options, write me to sanmayce@sanmayce.com, will try to add them.

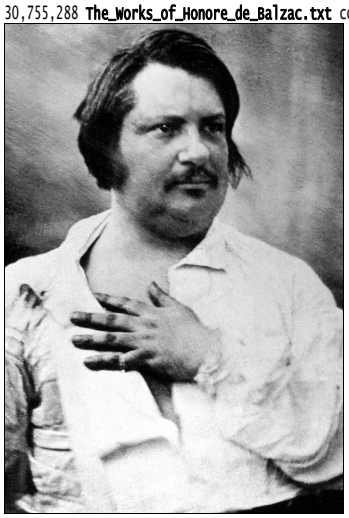
By the way, is someone aware of existing an LZ approach dealing with the stream bi-directionally (not from left-to-right but from left-to-the-right and right-to-the-left until they converge)? Inspired by the kanji 中 denoting center [|] and 乙成, 音成 I look forward to implementing either [<-|->] or [-> | <-] LZ type of parsing while the left and right streams will share matches and will be dealt back-to-back at a loop level - hoping to improve compression (there will be flags for using left-or-right stream and using offset from last offset, plus keeping 1TB in both directions thus having 2TB window) and reduce latency during fetching uncached RAM ...

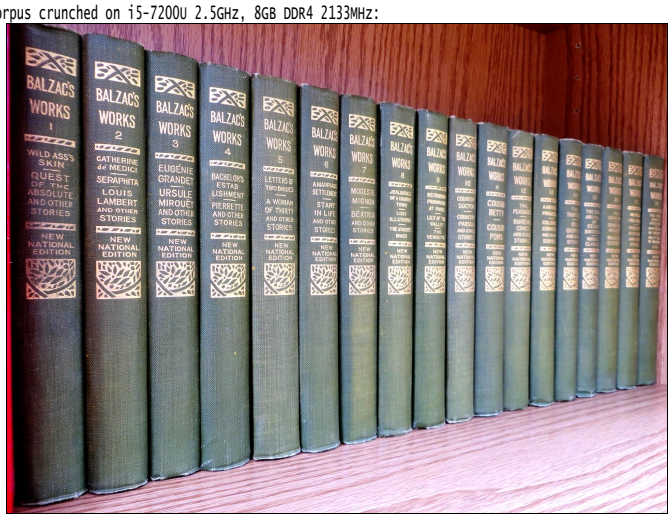
Incoming... Nakamichi 'Otonari', to put it poetically, 'The-Whisper-Of-Sword's-Soul' or '**Sword's Breath**' since soul etymology is breath. "Could there be a hidden wordplay with 乙成, "duplicate, strange, the latter, witty" and "become, elapse, get, grow, reach, turn into"? My goal is to use 'Otonari' as a name of one new compression algorithm based on Lempel-Ziv but cutting the data in half and since DUPLICATE GET/REACH depicts the process I wanna go with 乙成, a pun I want to have, that is." Source: https://www.quora.com/Having-seen-a-Japanese-saber-master-saying-otonari-meaning-the-hissing-sound-coming-from-cutting-the-air-I-ask-how-to-write-it-Why-when-searching-for-otonari-wiktionary-the-definition-is-a-fay-a-fairy

Nakamichi 'Dragoneye' LITE a.k.a. **Satanichi** highlights:
- The Zennish LZSS Microdeduplicator, written in plain C;
- The first Lempel-Ziv-Storer-Szymanski (LZSS) compressor using 1TB sliding window;
- It targets superfast decompression of huge/tera corpora of textual data;
- Compressing at Linear Rate due to Matchfinding based on B-trees only;
- The Internal/External B-trees order 3 (2 keys MAX) are brutally-optimized;
- Ability to deduplicate (as little as) 64 bytes long chunks 1TB backwards;
- Single-threaded Non-SIMD console tool written entirely by yours truly Sanmayce;
- SCALABLE! Gets faster when more Physical or/and External RAM is available;
- Dynamical (as command line parameters) selection of HASH & B-trees memory blocks;
- Compileable under Windows and Linux, most suitable to run on 3TB machines;
- Many thanks go to: Haruhiko Okumura, Nobuo Ito, Hamid Buzidi, Landon Noll.

TEXTORAMIC Decompression Showdown — SNAPPY vs LZO vs ZLIB vs ZSTD vs LZTURBO vs OODLE vs LIZARD vs Nakamichi

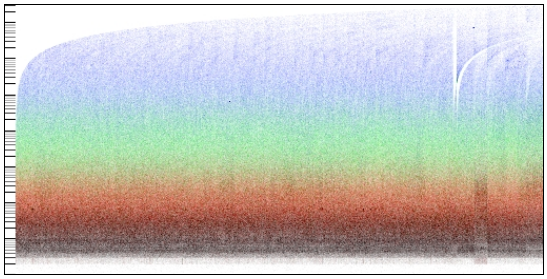
30,755,288 **The_Works_of_Honore_de_Balzac.txt** corpus crunched on i5-7200U 2.5GHz, 8GB DDR4 2133MHz:





中道乙成

```
D:\TEXTUAL_MADNESS_bare-minimum_2020_May_22>TIMER64 Satanichi_GCC730_64bit.exe "The_Works_of_Honore_de_Balzac.txt"
"The_Works_of_Honore_de_Balzac.txt.Nakamichi" 20 6000 1
...
Leprechaun: Total Searches-n-Inserts Per Second: 29,506,331 SNIPS = 20,064,305,655 keys in 680 seconds
Leprechaun: RAM needed to house B-trees (relative to the file being ripped): 31N = 913MB
Leprechaun: RAM needed to build B-trees IN ONE PASS: (Target-Buffer = 61 MB) x 128 passes = 7,808MB
Note: In case of RAM in spades then may use compile option: -DSpeedupBuilding=7808
```



The struture of the corpus is shown in the diagram above. Each pixel represents a match between consecutive occurrences of a string. The color of the pixel represents the length of the match: black for 1 byte, red for 2, green for 4 and blue for 8. The horizontal axis represents the position of the second occurrence of the string. The vertical axis represents the distance back to the match on a logarithmic scale. (The image was generated by the fv program).
http://www.mattmahoney.net/dc/dce.html

```
D:\TEXTUAL_MADNESS_bare-minimum_2020_May_22>lzbench18 -c4 -i1,15 -o3 -esnappy/lizard,49/zlib,9/lzsse2,17/lzo
lzbench 1.8 (64-bit Windows)   Assembled by P.Skibinski
```

Compressor name	Compress.	Decompress.	Compr. size	Ratio	Filename
lizard 1.0 -49	1.33 MB/s	**700** MB/s	9459737	30.76	The_Works_of_Honore_de_Balzac.txt
Nakamichi 2020-May-09		**865** MB/s	9963212		
libdeflate 1.3 -12	5.26 MB/s	**604** MB/s	10816799	35.17	The_Works_of_Honore_de_Balzac.txt
lzsse2 2019-04-18 -17	5.89 MB/s	**3024** MB/s	11329496	36.84	The_Works_of_Honore_de_Balzac.txt
zlib 1.2.11 -9	9.74 MB/s	**239** MB/s	11363596	36.95	The_Works_of_Honore_de_Balzac.txt
lzo1b 2.10 -999	6.95 MB/s	**423** MB/s	12513296	40.69	The_Works_of_Honore_de_Balzac.txt
snappy 2019-09-30	199 MB/s	**721** MB/s	18829221	61.22	The_Works_of_Honore_de_Balzac.txt
memcpy	10755 MB/s	**10382** MB/s	30755288	100.00	The_Works_of_Honore_de_Balzac.txt

```
D:\TEXTUAL_MADNESS_bare-minimum_2020_May_22>"turbobench_v18.05_-_build_04_May_2018" "The_Works_of_Honore_de_
```

C Size	ratio%	C MB/s	D MB/s	Name	File
6388621	20.8	7.75	**38**.10	lzturbo 59	The_Works_of_Honore_de_Balzac.txt
7998930	26.0	1.02	**644**.29	lzturbo 39	The_Works_of_Honore_de_Balzac.txt
8036825	26.1	0.34	**544**.35	oodle 139	The_Works_of_Honore_de_Balzac.txt
9050579	29.4	1.03	**1005**.30	lzturbo 29	The_Works_of_Honore_de_Balzac.txt
9489367	30.9	0.97	**1442**.62	oodle 99	The_Works_of_Honore_de_Balzac.txt
9963212			**865**	Nakamichi 2020-May-09	
12819693	41.7	1.22	**3075**.53	lzturbo 19	The_Works_of_Honore_de_Balzac.txt

```
D:\TEXTUAL_MADNESS_bare-minimum_2020_May_22>zstd-v1.4.5-win64.exe
-priority=rt "The_Works_of_Honore_de_Balzac.txt"
Note : switching to real-time priority xt...
 1#ore_de_Balzac.txt :  30755288 ->  12526422 (2.455), 213.2 MB/s , 819.0 MB/s
 2#ore_de_Balzac.txt :  30755288 ->  11472367 (2.681), 148.0 MB/s , 656.1 MB/s
 3#ore_de_Balzac.txt :  30755288 ->  10902090 (2.821), 123.9 MB/s , 630.1 MB/s
 4#ore_de_Balzac.txt :  30755288 ->  10687788 (2.878),  96.9 MB/s , 617.3 MB/s
 5#ore_de_Balzac.txt :  30755288 ->  10757994 (2.859),  59.2 MB/s , 592.1 MB/s
 6#ore_de_Balzac.txt :  30755288 ->  10449914 (2.943),  35.2 MB/s , 616.7 MB/s
 7#ore_de_Balzac.txt :  30755288 ->  10083431 (3.050),  25.1 MB/s , 660.4 MB/s
Nakamichi 2020-May-09                9963212                                  865 MB/s
 8#ore_de_Balzac.txt :  30755288 ->   9953121 (3.090),  19.3 MB/s , 684.1 MB/s
 9#ore_de_Balzac.txt :  30755288 ->   9792432 (3.141),  12.0 MB/s , 699.8 MB/s
10#ore_de_Balzac.txt :  30755288 ->   9687804 (3.175),  8.32 MB/s , 672.3 MB/s
11#ore_de_Balzac.txt :  30755288 ->   9624147 (3.196),  6.70 MB/s , 644.8 MB/s
12#ore_de_Balzac.txt :  30755288 ->   9466320 (3.249),  3.55 MB/s , 627.2 MB/s
13#ore_de_Balzac.txt :  30755288 ->   9207745 (3.340),  2.88 MB/s , 515.3 MB/s
14#ore_de_Balzac.txt :  30755288 ->   9028102 (3.407),  2.40 MB/s , 574.0 MB/s
15#ore_de_Balzac.txt :  30755288 ->   8899769 (3.456),  2.21 MB/s , 499.3 MB/s
16#ore_de_Balzac.txt :  30755288 ->   8518793 (3.610),  2.31 MB/s , 647.0 MB/s
17#ore_de_Balzac.txt :  30755288 ->   8365547 (3.676),  1.84 MB/s , 556.0 MB/s
18#ore_de_Balzac.txt :  30755288 ->   8239688 (3.733),  1.56 MB/s , 473.0 MB/s
19#ore_de_Balzac.txt :  30755288 ->   8146378 (3.775),  1.48 MB/s , 434.7 MB/s
20#ore_de_Balzac.txt :  30755288 ->   8014653 (3.837),  1.31 MB/s , 553.5 MB/s
21#ore_de_Balzac.txt :  30755288 ->   8001472 (3.844),  1.29 MB/s , 554.1 MB/s
22#ore_de_Balzac.txt :  30755288 ->   8001421 (3.844),  1.27 MB/s , 554.5 MB/s
```

Notes:
- Latest available '**oo2core_8_win64.dll**' was used;
- **Oodle** Compressors: Kraken = 8 (Default); Leviathan = 13 (Best); Mermaid = 9 (Crazy fast); Selkie = 11 (Fas
- **Oodle** Levels: HyperFast = -4..-1; None = 0; SuperFast = 1; VeryFast = 2; Fast = 3; Normal = 4; Optimal1,2,
- The tweak value **spacespeedtradeoff**=[64..1024] "tweak size vs decode time" has not been played with!
- If you wanna see more performers/options, write me to sanmayce@sanmayce.com, will try to add them.

By the way, is someone aware of existing an LZ approach dealing with the stream bi-directionally (not from left-to-right but from left-to-the-right and right-to-the-left until they converge)? Inspired by the kanji 中 denoting center [|] and 乙成, 音成 I look forward to implementing either [<-|->] or [-> | <-] LZ type of parsing while the left and right streams will share matches and will be dealt back-to-back at a loop level - hoping to improve compression (there will be flags for using left-or-right stream and using offset from last offset, plus seeing 1TB in both directions thus having 2TB window) and reduce latency during fetching uncached RAM ...

Incoming... Nakamichi 'Otonari', to put it poetically, 'The-Whisper-Of-Sword's-Soul' or '**Sword's Breath**' since soul etymology is breath. "Could there be a hidden wordplay with 乙成, "duplicate, strange, the latter, witty" and "become, elapse, get, grow, reach, turn into"? My goal is to use 'Otonari' as a name of one new compression algorithm based on Lempel-Ziv but cutting the data in half and since DUPLICATE GET/REACH depicts the process I wanna go with 乙成, a pun I want to have, that is." Source: https://www.quora.com/Having-seen-a-Japanese-sword-master-saying-otonari-meaning-the-hissing-sound-coming-from-cutting-the-air-I-ask-how-to-write-it-Why-when-searching-for-otonari-wiktionary-the-definition-is-a-fay-a-fairy

Nakamichi 'Dragoneye' LITE a.k.a. **Satanichi** highlights:
- The Zennish LZSS Microdeduplicator, written in plain C;
- The first Lempel-Ziv-Storer-Szymanski (LZSS) compressor using 1TB sliding window;
- It targets superfast decompression of huge/tera corpora of textual data;
- Compressing at Linear Rate due to Matchfinding based on B-trees only;
- The Internal/External B-trees order 3 (2 keys MAX) are brutally-optimized;
- Ability to deduplicate (as little as) 64 bytes long chunks 1TB backwards;
- Single-threaded Non-SIMD console tool written entirely by yours truly Sanmayce;
- SCALABLE! Gets faster when more Physical or/and External RAM is available;
- Dynamical (as command line parameters) selection of HASH & B-trees memory blocks;
- Compileable under Windows and Linux, most suitable to run on 3TB machines;
- Many thanks go to: Haruhiko Okumura, Nobuo Ito, Hamid Buzidi, Landon Noll.

TEXTORAMIC Decompression Showdown — SNAPPY vs LZO vs ZLIB vs ZSTD vs LZTURBO vs OODLE vs LIZARD vs Nakamichi

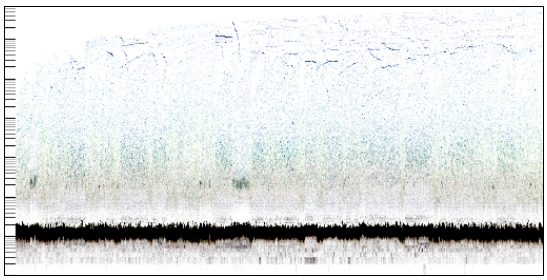
7,000,453 **Word-list_584879_words_Russian_Spell-Check.txt** corpus crunched on i5-7200U 2.5GHz, 8GB DDR4 2133MHz:

АБАЖУР
АБАЖУРА
АБАЖУРАМ
АБАЖУРАМИ
АБАЖУРАХ
АБАЖУРЕ
АБАЖУРОВ
АБАЖУРОМ
АБАЖУРУ
АБАЖУРЫ
АБАК
АБАКА
АБАКАМ
АБАКАМИ
АБАКАХ
АБАКЕ
АБАКИ
АБАКОВ
АБАКОМ
АБАКУ
АББАТ
...

...
ЯЩИЧКОВ
ЯЩИЧКОМ
ЯЩИЧКУ
ЯЩИЧНАЯ
ЯЩИЧНОГО
ЯЩИЧНОЕ
ЯЩИЧНОЙ
ЯЩИЧНОМ
ЯЩИЧНОМУ
ЯЩИЧНОЮ
ЯЩИЧНУЮ
ЯЩИЧНЫЕ
ЯЩИЧНЫЙ
ЯЩИЧНЫМ
ЯЩИЧНЫМИ
ЯЩИЧНЫХ
ЯЩУР
ЯЩУРА
ЯЩУРЕ
ЯЩУРОМ
ЯЩУРУ

中道乙成

```
D:\TEXTUAL_MADNESS_bare-minimum_2020_May_22>TIMER64 Satanichi_gcc730_64bit.exe "word-list_584879_words_Russian_Spell-Check.txt" "word-list_584879_words_Russian_Spell-Check.txt.Nakamichi" 20 6000 i
...
Leprechaun: Total Searches-n-Inserts Per Second: 26,254,243 SNIPS = 2,284,119,195 keys in 87 seconds
Leprechaun: RAM needed to house B-trees (relative to the file being ripped): 18N = 123MB
Leprechaun: RAM needed to build B-trees IN ONE PASS: (Target-Buffer = 38 MB) x 64 passes = 2,432MB
Note: In case of RAM in spades then may use compile option: -DSpeedupbuilding=2432
```



The struture of the corpus is shown in the diagram above. Each pixel represents a match between consecutive occurrences of a string. The color of the pixel represents the length of the match: black for 1 byte, red for 2, green for 4 and blue for 8. The horizontal axis represents the position of the second occurrence of the string. The vertical axis represents the distance back to the match on a logarithmic scale. (The image was generated by the fv program).
http://www.mattmahoney.net/dc/dce.html

```
D:\TEXTUAL_MADNESS_bare-minimum_2020_May_22>lzbench18 -c4 -i1,15 -o3 -esnappy/lizard,49/zlib,9/lzsse2,17/lzo1b,999/
lzbench 1.8 (64-bit Windows)   Assembled by P.Skibinski
```

Compressor name	Compress.	Decompress.	Compr. size	Ratio	Filename
libdeflate 1.3 -12	9.02 MB/s	**717** MB/s	1127846	16.11	Word-list_584879_words_Russian_Spell-Check.txt
zlib 1.2.11 -9	1.66 MB/s	**311** MB/s	1410041	20.14	Word-list_584879_words_Russian_Spell-Check.txt
lizard 1.0 -49	0.92 MB/s	**1027** MB/s	1683405	24.05	Word-list_584879_words_Russian_Spell-Check.txt
Nakamichi 2020-May-09		**1060** MB/s	1999186		
lzsse2 2019-04-18 -17	10 MB/s	**2114** MB/s	2152105	30.74	Word-list_584879_words_Russian_Spell-Check.txt
lzo1b 2.10 -999	6.51 MB/s	**640** MB/s	2221711	31.74	Word-list_584879_words_Russian_Spell-Check.txt
snappy 2019-09-30	303 MB/s	**1113** MB/s	2449583	34.99	Word-list_584879_words_Russian_Spell-Check.txt
memcpy	11854 MB/s	**10906** MB/s	7000453	100.00	Word-list_584879_words_Russian_Spell-Check.txt

```
D:\TEXTUAL_MADNESS_bare-minimum_2020_May_22>"turbobench_v18.05_-_build_04_May_2018" "Word-list_584879_words_Russian
```

C Size	ratio%	C MB/s	D MB/s	Name	File
989344	14.1	0.35	**838**.58	oodle 139	Word-list_584879_words_Russian_Spell-Check.txt
1037312	14.8	2.67	**803**.91	lzturbo 39	Word-list_584879_words_Russian_Spell-Check.txt
1114009	15.9	2.49	**1383**.76	oodle 99	Word-list_584879_words_Russian_Spell-Check.txt
1638495	23.4	2.86	**1001**.07	lzturbo 29	Word-list_584879_words_Russian_Spell-Check.txt
1650129	23.6	14.38	**48**.87	lzturbo 59	Word-list_584879_words_Russian_Spell-Check.txt
1999186			**1060**	Nakamichi 2020-May-09	
2339944	33.4	3.62	**1483**.78	lzturbo 19	Word-list_584879_words_Russian_Spell-Check.txt

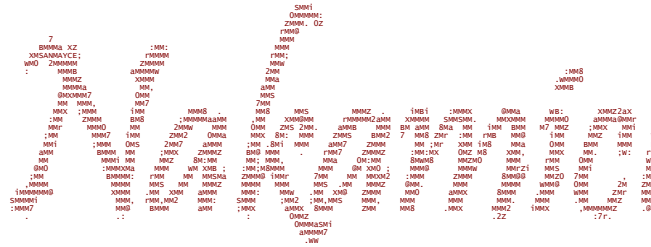
```
D:\TEXTUAL_MADNESS_bare-minimum_2020_May_22>zstd-v1.4.5-wi
-priority=rt "Word-list_584879_words_Russian_Spell-Check.t
Note : switching to real-time priority Spell-Check.txt...
```

Nakamichi 2020-May-09		1999186		**1060** MB/s
1#n_Spell-Check.txt :	7000453 ->	1778583 (3.936),	246.6 MB/s ,	**641**.5 MB/s
2#n_Spell-Check.txt :	7000453 ->	1723243 (4.062),	224.6 MB/s ,	**585**.5 MB/s
3#n_Spell-Check.txt :	7000453 ->	1671341 (4.189),	172.7 MB/s ,	**572**.1 MB/s
4#n_Spell-Check.txt :	7000453 ->	1678305 (4.171),	141.6 MB/s ,	**566**.3 MB/s
5#n_Spell-Check.txt :	7000453 ->	1548285 (4.521),	111.3 MB/s ,	**551**.3 MB/s
6#n_Spell-Check.txt :	7000453 ->	1555758 (4.500),	85.9 MB/s ,	**555**.5 MB/s
7#n_Spell-Check.txt :	7000453 ->	1374383 (5.094),	60.8 MB/s ,	**619**.4 MB/s
8#n_Spell-Check.txt :	7000453 ->	1314094 (5.327),	50.8 MB/s ,	**656**.3 MB/s
9#n_Spell-Check.txt :	7000453 ->	1321925 (5.296),	33.5 MB/s ,	**659**.8 MB/s
10#n_Spell-Check.txt :	7000453 ->	1325193 (5.283),	30.2 MB/s ,	**658**.5 MB/s
11#n_Spell-Check.txt :	7000453 ->	1327768 (5.272),	28.0 MB/s ,	**657**.7 MB/s
12#n_Spell-Check.txt :	7000453 ->	1337155 (5.235),	16.5 MB/s ,	**661**.6 MB/s
13#n_Spell-Check.txt :	7000453 ->	1320209 (5.303),	20.0 MB/s ,	**659**.4 MB/s
14#n_Spell-Check.txt :	7000453 ->	1319328 (5.306),	17.6 MB/s ,	**659**.2 MB/s
15#n_Spell-Check.txt :	7000453 ->	1319927 (5.304),	11.9 MB/s ,	**659**.0 MB/s
16#n_Spell-Check.txt :	7000453 ->	1138141 (6.151),	8.01 MB/s ,	**580**.7 MB/s
17#n_Spell-Check.txt :	7000453 ->	1139508 (6.143),	5.42 MB/s ,	**580**.5 MB/s
18#n_Spell-Check.txt :	7000453 ->	1113950 (6.284),	3.08 MB/s ,	**582**.6 MB/s
19#n_Spell-Check.txt :	7000453 ->	1112720 (6.291),	2.18 MB/s ,	**583**.3 MB/s
20#n_Spell-Check.txt :	7000453 ->	1112720 (6.291),	2.17 MB/s ,	**583**.1 MB/s
21#n_Spell-Check.txt :	7000453 ->	1112720 (6.291),	2.17 MB/s ,	**583**.3 MB/s
22#n_Spell-Check.txt :	7000453 ->	1112636 (6.292),	1.14 MB/s ,	**582**.8 MB/s

Notes:
- Latest available '**oo2core_8_win64.dll**' was used;
- **Oodle** Compressors: Kraken = 8 (Default); Leviathan = 13 (Best); Mermaid = 9 (Crazy fast); Selkie = 11 (Fastest);
- **Oodle** Levels: HyperFast = -4..-1; None = 0; SuperFast = 1; VeryFast = 2; Fast = 3; Normal = 4; Optimal1,2,3,4 = 5
- The tweak value **spacespeedtradeoff**=[64..1024] "tweak size vs decode time" has not been played with!
- If you wanna see more performers/options, write me to sanmayce@sanmayce.com, will try to add them.

By the way, is someone aware of existing an LZ approach dealing with the stream bi-directionally (not from left-to-right but from left-to-the-right and right-to-the-left until they converge)? Inspired by the kanji 中 denoting center [|] and 乙成, 音成 I look forward to implementing either [<-|->] or [-> | <-] LZ type of parsing while the left and right streams will share matches and will be dealt back-to-back at a loop level - hoping to improve compression (there will be flags for using left-or-right stream and using offset from last offset, plus seeing 1TB in both directions thus having 2TB window) and reduce latency during fetching uncached RAM ...

Incoming... Nakamichi 'Otonari', to put it poetically, 'The-Whisper-Of-Sword's-Soul' or '**Sword's Breath**' since soul etymology is breath. "Could there be a hidden wordplay with 乙成, "duplicate, strange, the latter, witty" and "become, elapse, get, grow, reach, turn into"? My goal is to use 'Otonari' as a name of one new compression algorithm based on Lempel-Ziv but cutting the data in half and since DUPLICATE GET/REACH depicts the process I wanna go with 乙成, a pun I want to have, that is." Source: https://www.quora.com/Having-seen-a-Japanese-sabre-master-saying-otonari-meaning-the-hissing-sound-coming-from-cutting-the-air-I-ask-how-to-write-it-Why-when-searching-for-otonari-wiktionary-the-definition-is-a-fay-a-fairy



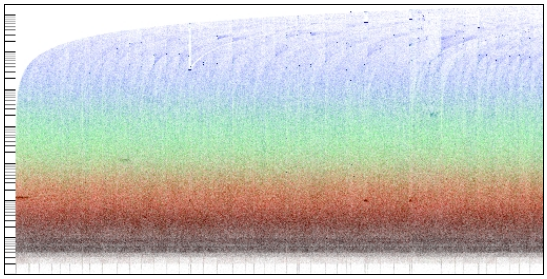
Nakamichi 'Dragoneye' LITE a.k.a. **Satanichi** highlights:
- The Zennish LZSS Microdeduplicator, written in plain C;
- The first Lempel-Ziv-Storer-Szymanski (LZSS) compressor using 1TB sliding window;
- It targets superfast decompression of huge/tera corpora of textual data;
- Compressing at Linear Rate due to Matchfinding based on B-trees only;
- The Internal/External B-trees order 3 (2 keys MAX) are brutally-optimized;
- Ability to deduplicate (as little as) 64 bytes long chunks 1TB backwards;
- Single-threaded Non-SIMD console tool written entirely by yours truly Sanmayce;
- SCALABLE! Gets faster when more Physical or/and External RAM is available;
- Dynamical (as command line parameters) selection of HASH & B-trees memory blocks;
- Compileable under Windows and Linux, most suitable to run on 3TB machines;
- Many thanks go to: Haruhiko Okumura, Nobuo Ito, Hamid Buzidi, Landon Noll.

TEXTORAMIC Decompression Showdown — SNAPPY vs LZO vs ZLIB vs ZSTD vs LZTURBO vs OODLE vs LIZARD vs Nakamichi

18,933,760 **James_Bond_Collection_-_52_Ebooks.tar** corpus crunched on i5-7200U 2.5GHz, 8GB DDR4 2133MHz:

中道乙成

```
D:\TEXTUAL_MADNESS_bare-minimum_2020_May_22>TIMER64 Satanichi_GCC730_64bit.exe "James_Bond_Collection_-
_52_Ebooks.tar" "James_Bond_Collection_-_52_Ebooks.tar.Nakamichi" 20 6000 1
...
Leprechaun: Total Searches-n-Inserts Per Second: 29,827,611 SNIPS = 11,453,802,735 keys in 384 seconds
Leprechaun: RAM needed to house B-trees (relative to the file being ripped): 40N = 726MB
Leprechaun: RAM needed to build B-trees IN ONE PASS: (Target-Buffer = 50 MB) x 128 passes = 6,400MB
Note: In case of RAM in spades then may use compile option: -DSpeeduponBuilding=6400
```



The struture of the corpus is shown in the diagram above. Each pixel represents a match between consecutive occurrences of a string. The color of the pixel represents the length of the match: black for 1 byte, red for 2, green for 4 and blue for 8. The horizontal axis represents the position of the second occurrence of the string. The vertical axis represents the distance back to the match on a logarithmic scale. (The image was generated by the fv program).
http://www.mattmahoney.net/dc/dce.html

```
D:\TEXTUAL_MADNESS_bare-minimum_2020_May_22>lzbench18 -c4 -i1,15 -o3 -esnappy/lizard,49/zlib,9/lzsse2,17/lzo1b
lzbench 1.8 (64-bit Windows)   Assembled by P.Skibinski
```

Compressor name	Compress.	Decompress.	Compr. size	Ratio	Filename
lizard 1.0 -49	1.39 MB/s	**741** MB/s	5859826	30.95	James_Bond_Collection_-_52_Ebooks.tar
Nakamichi 2020-May-09		**885** MB/s	6270629		
libdeflate 1.3 -12	5.43 MB/s	**605** MB/s	6727797	35.53	James_Bond_Collection_-_52_Ebooks.tar
zlib 1.2.11 -9	9.70 MB/s	**236** MB/s	7060922	37.29	James_Bond_Collection_-_52_Ebooks.tar
lzsse2 2019-04-18 -17	5.88 MB/s	**2984** MB/s	7077598	37.38	James_Bond_Collection_-_52_Ebooks.tar
lzo1b 2.10 -999	7.12 MB/s	**418** MB/s	7787857	41.13	James_Bond_Collection_-_52_Ebooks.tar
snappy 2019-09-30	201 MB/s	**726** MB/s	11546414	60.98	James_Bond_Collection_-_52_Ebooks.tar
memcpy	11664 MB/s	**11195** MB/s	18933760	100.00	James_Bond_Collection_-_52_Ebooks.tar

```
D:\TEXTUAL_MADNESS_bare-minimum_2020_May_22>"turbobench_v18.05_-_build_04_May_2018" "James_Bond_Collection_-_5
```

C Size	ratio%	C MB/s	D MB/s	Name	File
4080341	21.6	8.15	**38**.08	lzturbo 59	James_Bond_Collection_-_52_Ebooks.tar
5006363	26.4	1.11	**683**.38	lzturbo 39	James_Bond_Collection_-_52_Ebooks.tar
5011440	26.5	0.36	**568**.39	oodle 139	James_Bond_Collection_-_52_Ebooks.tar
5674570	30.0	1.12	**1060**.77	lzturbo 29	James_Bond_Collection_-_52_Ebooks.tar
5852417	30.9	1.17	**1476**.20	oodle 99	James_Bond_Collection_-_52_Ebooks.tar
6270629			**885**	Nakamichi 2020-May-09	
7980947	42.2	1.33	**3101**.86	lzturbo 19	James_Bond_Collection_-_52_Ebooks.tar

```
D:\TEXTUAL_MADNESS_bare-minimum_2020_May_22>zstd-v1.4.5-win64.
-priority=rt "James_Bond_Collection_-_52_Ebooks.tar"
Note : switching to real-time priority ks.tar...
 1#n_-_52_Ebooks.tar :  18933760 ->   7813100 (2.423), 213.8 MB/s , 812.4 MB/s
 2#n_-_52_Ebooks.tar :  18933760 ->   7142934 (2.651), 148.2 MB/s , 659.2 MB/s
 3#n_-_52_Ebooks.tar :  18933760 ->   6788230 (2.789), 123.1 MB/s , 629.8 MB/s
 4#n_-_52_Ebooks.tar :  18933760 ->   6664034 (2.841),  97.1 MB/s , 614.6 MB/s
 5#n_-_52_Ebooks.tar :  18933760 ->   6681667 (2.834),  57.8 MB/s , 590.1 MB/s
 6#n_-_52_Ebooks.tar :  18933760 ->   6501400 (2.912),  35.4 MB/s , 612.5 MB/s
 7#n_-_52_Ebooks.tar :  18933760 ->   6277030 (3.016),  25.1 MB/s , 657.1 MB/s
Nakamichi 2020-May-09              6270629                                885 MB/s
 8#n_-_52_Ebooks.tar :  18933760 ->   6202956 (3.052),  19.4 MB/s , 680.2 MB/s
 9#n_-_52_Ebooks.tar :  18933760 ->   6112504 (3.098),  12.1 MB/s , 694.1 MB/s
10#n_-_52_Ebooks.tar :  18933760 ->   6037029 (3.136),   8.55 MB/s , 661.9 MB/s
11#n_-_52_Ebooks.tar :  18933760 ->   5995828 (3.158),   6.90 MB/s , 634.9 MB/s
12#n_-_52_Ebooks.tar :  18933760 ->   5908464 (3.205),   3.70 MB/s , 636.0 MB/s
13#n_-_52_Ebooks.tar :  18933760 ->   5754064 (3.291),   3.65 MB/s , 677.0 MB/s
14#n_-_52_Ebooks.tar :  18933760 ->   5642083 (3.356),   2.91 MB/s , 574.2 MB/s
15#n_-_52_Ebooks.tar :  18933760 ->   5561937 (3.404),   2.30 MB/s , 505.1 MB/s
16#n_-_52_Ebooks.tar :  18933760 ->   5326860 (3.554),   2.45 MB/s , 646.8 MB/s
17#n_-_52_Ebooks.tar :  18933760 ->   5183171 (3.653),   1.89 MB/s , 558.9 MB/s
18#n_-_52_Ebooks.tar :  18933760 ->   5114110 (3.702),   1.59 MB/s , 483.6 MB/s
19#n_-_52_Ebooks.tar :  18933760 ->   5054938 (3.746),   1.53 MB/s , 453.9 MB/s
20#n_-_52_Ebooks.tar :  18933760 ->   5004628 (3.783),   1.44 MB/s , 536.1 MB/s
21#n_-_52_Ebooks.tar :  18933760 ->   5003865 (3.784),   1.38 MB/s , 533.5 MB/s
22#n_-_52_Ebooks.tar :  18933760 ->   5003750 (3.784),   1.39 MB/s , 531.3 MB/s
```

Notes:
- Latest available '**oo2core_8_win64.dll**' was used;
- **Oodle** Compressors: Kraken = 8 (Default); Leviathan = 13 (Best); Mermaid = 9 (Crazy fast); Selkie = 11 (Faste
- **Oodle** Levels: HyperFast = -4..-1; None = 0; SuperFast = 1; VeryFast = 2; Fast = 3; Normal = 4; Optimal1,2,3,
- The tweak value **spacespeedtradeoff**=[64..1024] "tweak size vs decode time" has not been played with!
- If you wanna see more performers/options, write me to sanmayce@sanmayce.com, will try to add them.

By the way, is someone aware of existing an LZ approach dealing with the stream bi-directionally (not from left-to-right but from left-to-the-right and right-to-the-left until they converge)? Inspired by the kanji 中 denoting center [|] and 乙成, 音成 I look forward to implementing either [<-|->] or [-> | <-] LZ type of parsing while the left and right streams will share matches and will be dealt back-to-back at a loop level - hoping to improve compression (there will be flags for using left-or-right stream and using offset from last offset, processing 1TB in both directions thus having 2TB window) and reduce latency during fetching uncached RAM ...

Incoming... Nakamichi 'Otonari', to put it poetically, 'The-Whisper-Of-Sword's-Soul' or '**Sword's Breath**' since soul etymology is breath. "Could there be a hidden wordplay with 乙成, "duplicate, strange, the latter, witty" and "become, elapse, get, grow, reach, turn into"? My goal is to use 'Otonari' as a name of one new compression algorithm based on Lempel-Ziv but cutting the data in half and since DUPLICATE GET/REACH depicts the process I wanna go with 乙成, a pun I want to have, that is." Source: https://www.quora.com/Having-seen-a-Japanese-Saber-master-saying-otonari-meaning-the-hissing-sound-coming-from-cutting-the-air-I-ask-how-to-write-it-Why-when-searching-for-otonari-wiktionary-the-definition-is-a-fay-a-fairy

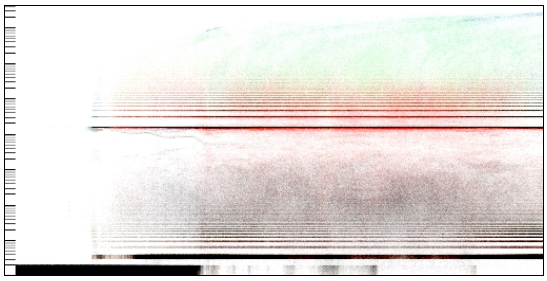
Nakamichi 'Dragoneye' LITE a.k.a. **Satanichi** highlights:
- The Zennish LZSS Microdeduplicator, written in plain C;
- The first Lempel-Ziv-Storer-Szymanski (LZSS) compressor using 1TB sliding window;
- It targets superfast decompression of huge/tera corpora of textual data;
- Compressing at Linear Rate due to Matchfinding based on B-trees only;
- The Internal/External B-trees order 3 (2 keys MAX) are brutally-optimized;
- Ability to deduplicate (as little as) 64 bytes long chunks 1TB backwards;
- Single-threaded Non-SIMD console tool written entirely by yours truly Sanmayce;
- SCALABLE! Gets faster when more Physical or/and External RAM is available;
- Dynamical (as command line parameters) selection of HASH & B-trees memory blocks;
- Compileable under Windows and Linux, most suitable to run on 3TB machines;
- Many thanks go to: Haruhiko Okumura, Nobuo Ito, Hamid Buzidi, Landon Noll.

TEXTORAMIC Decompression Showdown — SNAPPY vs LZO vs ZLIB vs ZSTD vs LZTURBO vs OODLE vs LIZARD vs Nakamichi

44,236,856 **detective-pikachu_24bit_5120x2880.bmp** corpus crunched on i5-7200U 2.5GHz, 8GB DDR4 2133MHz:

中道乙成

```
D:\TEXTUAL_MADNESS_bare-minimum_2020_May_22>TIMER64 Satanichi_GCC730_64bit.exe "detective-
pikachu_24bit_5120x2880.bmp" "detective-pikachu_24bit_5120x2880.bmp.Nakamichi" 20 6000 1
...
Leprechaun: Total Searches-n-Inserts Per Second: 31,883,226 SNIPS = 32,144,339,936 keys in 1,008 seconds
Leprechaun: RAM needed to house B-trees (relative to the file being ripped): 18N = 798MB
Leprechaun: RAM needed to build B-trees IN ONE PASS: (Target-Buffer = 74 MB) x 128 passes = 9,472MB
Note: In case of RAM in spades then may use compile option: -DSpeeduptuilding=9472
```



The struture of the corpus is shown in the diagram above. Each pixel represents a match between consecutive occurrences of a string. The color of the pixel represents the length of the match: black for 1 byte, red for 2, green for 4 and blue for 8. The horizontal axis represents the position of the second occurrence of the string. The vertical axis represents the distance back to the match on a logarithmic scale. (The image was generated by the fv program).
http://www.mattmahoney.net/dc/dce.html

```
D:\TEXTUAL_MADNESS_bare-minimum_2020_May_22>lzbench18 -c4 -i1,15 -o3 -esnappy/lizard,49/zlib,9/lzsse2,17/lzo1b
lzbench 1.8 (64-bit Windows)   Assembled by P.Skibinski
Compressor name         Compress. Decompress.  Compr. size  Ratio Filename
libdeflate 1.3 -12         12 MB/s    384 MB/s    23983918  54.22 detective-pikachu_24bit_5120x2880.bmp
zlib 1.2.11 -9             36 MB/s    239 MB/s    24945729  56.39 detective-pikachu_24bit_5120x2880.bmp
lizard 1.0 -49           3.74 MB/s   1640 MB/s    25956634  58.68 detective-pikachu_24bit_5120x2880.bmp
lzo1b 2.10 -999            27 MB/s    737 MB/s    26466669  59.83 detective-pikachu_24bit_5120x2880.bmp
Nakamichi 2020-May-09                  965 MB/s    27488569
lzsse2 2019-04-18 -17    0.07 MB/s   1417 MB/s    29055616  65.68 detective-pikachu_24bit_5120x2880.bmp
snappy 2019-09-30        2287 MB/s   3847 MB/s    31427615  71.04 detective-pikachu_24bit_5120x2880.bmp
memcpy                  10147 MB/s  10127 MB/s    44236856 100.00 detective-pikachu_24bit_5120x2880.bmp
```

```
D:\TEXTUAL_MADNESS_bare-minimum_2020_May_22>"turbobench_v18.05_-_build_04_May_2018" "detective-pikachu_24bit_5
  C Size  ratio%  C MB/s  D MB/s  Name        File
  17309911   39.1    0.44     653.22  oodle 139   detective-pikachu_24bit_5120x2880.bmp
  18711865   42.3   11.27      24.60  lzturbo 59  detective-pikachu_24bit_5120x2880.bmp
  20200979   45.7    2.58    1413.14  oodle 99    detective-pikachu_24bit_5120x2880.bmp
  20768348   46.9    2.34     597.81  lzturbo 39  detective-pikachu_24bit_5120x2880.bmp
  24335793   55.0    2.54    1457.80  lzturbo 29  detective-pikachu_24bit_5120x2880.bmp
  26343258   59.6    3.17    3405.72  lzturbo 19  detective-pikachu_24bit_5120x2880.bmp
  27488569                     965    Nakamichi 2020-May-09
```

```
D:\TEXTUAL_MADNESS_bare-minimum_2020_May_22>zstd-v1.4.5-win64.
-priority=rt "detective-pikachu_24bit_5120x2880.bmp"
Note : switching to real-time priority 80.bmp...
 1#bit_5120x2880.bmp :  44236856 ->  27558105 (1.605), 413.1 MB/s ,1066.2 MB/s
Nakamichi 2020-May-09            27488569                           965 MB/s
 2#bit_5120x2880.bmp :  44236856 ->  26330066 (1.680), 266.3 MB/s ,1123.0 MB/s
 3#bit_5120x2880.bmp :  44236856 ->  25848972 (1.711), 152.9 MB/s ,1209.7 MB/s
 4#bit_5120x2880.bmp :  44236856 ->  25269983 (1.751),  83.0 MB/s ,1246.6 MB/s
 5#bit_5120x2880.bmp :  44236856 ->  24525580 (1.804),  31.5 MB/s ,1175.0 MB/s
 6#bit_5120x2880.bmp :  44236856 ->  24355090 (1.816),  22.6 MB/s ,1184.0 MB/s
 7#bit_5120x2880.bmp :  44236856 ->  24312789 (1.819),  21.0 MB/s ,1221.9 MB/s
 8#bit_5120x2880.bmp :  44236856 ->  24297171 (1.821),  19.4 MB/s ,1241.9 MB/s
 9#bit_5120x2880.bmp :  44236856 ->  24160302 (1.831),  19.5 MB/s ,1223.8 MB/s
10#bit_5120x2880.bmp :  44236856 ->  23874323 (1.853),  16.2 MB/s ,1068.5 MB/s
11#bit_5120x2880.bmp :  44236856 ->  23776183 (1.861),  16.1 MB/s ,1020.9 MB/s
12#bit_5120x2880.bmp :  44236856 ->  23757944 (1.862),  15.2 MB/s ,1025.9 MB/s
13#bit_5120x2880.bmp :  44236856 ->  23770029 (1.861),  11.4 MB/s ,1041.5 MB/s
14#bit_5120x2880.bmp :  44236856 ->  23710181 (1.866),  11.2 MB/s ,1020.2 MB/s
15#bit_5120x2880.bmp :  44236856 ->  23678520 (1.868),  9.96 MB/s ,1008.2 MB/s
16#bit_5120x2880.bmp :  44236856 ->  23391791 (1.891),  9.17 MB/s , 992.5 MB/s
17#bit_5120x2880.bmp :  44236856 ->  22759665 (1.944),  6.70 MB/s , 660.0 MB/s
18#bit_5120x2880.bmp :  44236856 ->  21938376 (2.016),  5.44 MB/s , 468.0 MB/s
19#bit_5120x2880.bmp :  44236856 ->  21883677 (2.021),  4.63 MB/s , 454.6 MB/s
20#bit_5120x2880.bmp :  44236856 ->  21779487 (2.031),  4.57 MB/s , 489.6 MB/s
21#bit_5120x2880.bmp :  44236856 ->  21766212 (2.032),  4.61 MB/s , 488.9 MB/s
22#bit_5120x2880.bmp :  44236856 ->  21765682 (2.032),  4.39 MB/s , 489.2 MB/s
```

Notes:
- Latest available '**oo2core_8_win64.dll**' was used;
- **Oodle** Compressors: Kraken = 8 (Default); Leviathan = 13 (Best); Mermaid = 9 (Crazy fast); Selkie = 11 (Faste
- **Oodle** Levels: HyperFast = -4..-1; None = 0; SuperFast = 1; VeryFast = 2; Fast = 3; Normal = 4; Optimal1,2,3,
- The tweak value **spacespeedtradeoff**=[64..1024] "tweak size vs decode time" has not been played with!
- If you wanna see more performers/options, write me to sanmayce@sanmayce.com, will try to add them.

By the way, is someone aware of existing an LZ approach dealing with the stream bi-directionally (not from left-to-right but from left-to-the-right and right-to-the-left until they converge)? Inspired by the kanji 中 denoting center [|] and 乙成, 音成 I look forward to implementing either [<-|->] or [-> | <-] LZ type of parsing while the left and right streams will share matches and will be dealt back-to-back at a loop level - hoping to improve compression (there will be flags for using left-or-right stream and using offset from last offset, plus seeing 1TB in both directions thus having 2TB window) and reduce latency during fetching uncached RAM ...

Incoming... Nakamichi 'Otonari', to put it poetically, 'The-Whisper-Of-Sword's-Soul' or '**Sword's Breath**' since soul etymology is breath. "Could there be a hidden wordplay with 乙成, "duplicate, strange, the latter, witty" and "become, elapse, get, grow, reach, turn into"? My goal is to use 'Otonari' as a name of one new compression algorithm based on Lempel-Ziv but cutting the data in half and since DUPLICATE GET/REACH depicts the process I wanna go with 乙成, a pun I want to have, that is." Source: https://www.quora.com/Having-seen-a-Japanese-sailor-master-saying-otonari-meaning-the-hissing-sound-coming-from-cutting-the-air-I-ask-how-to-write-it-Why-when-searching-for-otonari-wiktionary-the-definition-is-a-fay-a-fairy

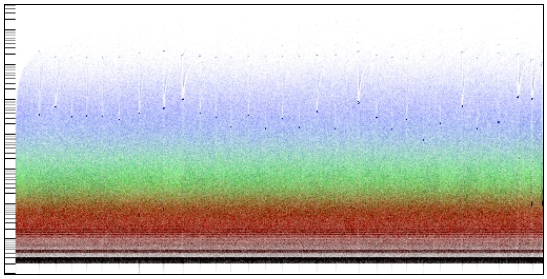
Nakamichi 'Dragoneye' LITE a.k.a. **Satanichi** highlights:
- The Zennish LZSS Microdeduplicator, written in plain C;
- The first Lempel-Ziv-Storer-Szymanski (LZSS) compressor using 1TB sliding window;
- It targets superfast decompression of huge/tera corpora of textual data;
- Compressing at Linear Rate due to Matchfinding based on B-trees only;
- The Internal/External B-trees order 3 (2 keys MAX) are brutally-optimized;
- Ability to deduplicate (as little as) 64 bytes long chunks 1TB backwards;
- Single-threaded Non-SIMD console tool written entirely by yours truly Sanmayce;
- SCALABLE! Gets faster when more Physical or/and External RAM is available;
- Dynamical (as command line parameters) selection of HASH & B-trees memory blocks;
- Compileable under Windows and Linux, most suitable to run on 3TB machines;
- Many thanks go to: Haruhiko Okumura, Nobuo Ito, Hamid Buzidi, Landon Noll.

TEXTORAMIC Decompression Showdown — SNAPPY vs LZO vs ZLIB vs ZSTD vs LZTURBO vs OODLE vs LIZARD vs Nakamichi

55,886,848 **Charles_Dickens_in_30_volumes_(Russian_UTF-8).tar** corpus crunched on i5-7200U 2.5GHz, 8GB DDR4 2133MHz:

中道乙成

```
D:\TEXTUAL_MADNESS_bare-minimum_2020_May_22cc>TIMER64 Satanichi_GCC730_64bit.exe
""Charles_Dickens_in_30_volumes_(Russian_UTF-8).tar"" ""Charles_Dickens_in_30_volumes_(Russian_UTF-
8).tar".Nakamichi" 20 6000 1
...
Leprechaun: Total Searches-n-Inserts Per Second: 32,382,527 SNIPS = 58,709,523,020 keys in 1,813 seconds
Leprechaun: RAM needed to house B-trees (relative to the file being ripped): 50N = 2,678MB
Leprechaun: RAM needed to build B-trees IN ONE PASS: (Target-Buffer = 85 MB) x 256 passes = 21,760MB
Note: In case of RAM in spades then may use compile option: -DSpeedupBuilding=21760
```



The struture of the corpus is shown in the diagram above. Each pixel represents a match between consecutive occurrences of a string. The color of the pixel represents the length of the match: black for 1 byte, red for 2, green for 4 and blue for 8. The horizontal axis represents the position of the second occurrence of the string. The vertical axis represents the distance back to the match on a logarithmic scale. (The image was generated by the fv program).
http://www.mattmahoney.net/dc/dce.html

```
D:\TEXTUAL_MADNESS_bare-minimum_2020_May_22>lzbench18 -c4 -i1,15 -o3 -esnappy/lizard,49/zlib,9/lzsse2,17/l
lzbench 1.8 (64-bit Windows)   Assembled by P.Skibinski
Compressor name         Compress. Decompress.  Compr. size  Ratio Filename
lizard 1.0 -49           1.28 MB/s   921 MB/s    10601754  18.97 Charles_Dickens_in_30_volumes_(Russian_U
Nakamichi 2020-May-09                1097 MB/s   12138431
libdeflate 1.3 -12       3.60 MB/s  1047 MB/s    14065912  25.17 Charles_Dickens_in_30_volumes_(Russian_U
lzsse2 2019-04-18 -17    4.52 MB/s  4066 MB/s    14318814  25.62 Charles_Dickens_in_30_volumes_(Russian_U
zlib 1.2.11 -9           4.15 MB/s   302 MB/s    15028440  26.89 Charles_Dickens_in_30_volumes_(Russian_U
lzo1b 2.10 -999          4.12 MB/s   590 MB/s    16270085  29.11 Charles_Dickens_in_30_volumes_(Russian_U
snappy 2019-09-30         242 MB/s   842 MB/s    27901961  49.93 Charles_Dickens_in_30_volumes_(Russian_U
memcpy                  11252 MB/s 10301 MB/s    55886848 100.00 Charles_Dickens_in_30_volumes_(Russian_U
```

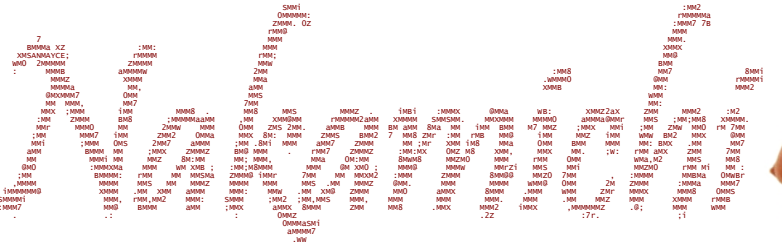
```
D:\TEXTUAL_MADNESS_bare-minimum_2020_May_22>"turbobench_v18.05_-_build_04_May_2018" ""Charles_Dickens_in_3
    C Size   ratio%   C MB/s   D MB/s   Name          File
   7091889    12.7     6.87      45.43   lzturbo 59    Charles_Dickens_in_30_volumes_(Russian_UTF-8)
   9222811    16.5     0.84     839.63   lzturbo 39    Charles_Dickens_in_30_volumes_(Russian_UTF-8)
   9415804    16.8     0.25     658.34   oodle 139     Charles_Dickens_in_30_volumes_(Russian_UTF-8)
  10471915    18.7     0.85    1333.05   lzturbo 29    Charles_Dickens_in_30_volumes_(Russian_UTF-8)
  11213646    20.1     0.67    1456.11   oodle 99      Charles_Dickens_in_30_volumes_(Russian_UTF-8)
  12138431              1097             Nakamichi 2020-May-09
  16359069    29.3     1.01    3236.82   lzturbo 19    Charles_Dickens_in_30_volumes_(Russian_UTF-8)
```

```
D:\TEXTUAL_MADNESS_bare-minimum_2020_May_22cc>zstd-v1.4.5-win64
--priority=rt ""Charles_Dickens_in_30_volumes_(Russian_UTF-8).t
Note : switching to real-time priority Russian_UTF-8).tar...
 1#ussian_UTF-8).tar :  55886848 ->  16865071 (3.314), 212.1 MB/s , 801.4 MB/s
 2#ussian_UTF-8).tar :  55886848 ->  16927486 (3.302), 202.8 MB/s , 684.4 MB/s
 3#ussian_UTF-8).tar :  55886848 ->  15362948 (3.638), 168.4 MB/s , 746.3 MB/s
 4#ussian_UTF-8).tar :  55886848 ->  15232789 (3.669), 150.9 MB/s , 741.1 MB/s
 5#ussian_UTF-8).tar :  55886848 ->  15684281 (3.563),  85.8 MB/s , 698.7 MB/s
 6#ussian_UTF-8).tar :  55886848 ->  14941008 (3.741),  56.6 MB/s , 747.2 MB/s
 7#ussian_UTF-8).tar :  55886848 ->  14277491 (3.914),  39.1 MB/s , 816.4 MB/s
 8#ussian_UTF-8).tar :  55886848 ->  13984811 (3.996),  29.6 MB/s , 847.8 MB/s
 9#ussian_UTF-8).tar :  55886848 ->  13386522 (4.175),  17.0 MB/s , 883.5 MB/s
10#ussian_UTF-8).tar :  55886848 ->  13302919 (4.201),  14.6 MB/s , 871.7 MB/s
11#ussian_UTF-8).tar :  55886848 ->  13258706 (4.215),  13.2 MB/s , 857.2 MB/s
12#ussian_UTF-8).tar :  55886848 ->  12703153 (4.399),  6.04 MB/s , 875.1 MB/s
Nakamichi 2020-May-09                12138431                                1097 MB/s
13#ussian_UTF-8).tar :  55886848 ->  11503797 (4.858),  3.31 MB/s , 914.6 MB/s
14#ussian_UTF-8).tar :  55886848 ->  11203711 (4.988),  2.82 MB/s , 771.9 MB/s
15#ussian_UTF-8).tar :  55886848 ->  10860995 (5.146),  1.91 MB/s , 644.1 MB/s
16#ussian_UTF-8).tar :  55886848 ->  10260828 (5.447),  2.37 MB/s , 831.6 MB/s
17#ussian_UTF-8).tar :  55886848 ->   9994013 (5.592),  2.02 MB/s , 710.6 MB/s
18#ussian_UTF-8).tar :  55886848 ->   9834203 (5.683),  1.63 MB/s , 625.4 MB/s
19#ussian_UTF-8).tar :  55886848 ->   9682116 (5.772),  1.54 MB/s , 579.4 MB/s
20#ussian_UTF-8).tar :  55886848 ->   9410298 (5.939),  1.38 MB/s , 774.6 MB/s
21#ussian_UTF-8).tar :  55886848 ->   9329225 (5.991),  1.29 MB/s , 774.4 MB/s
22#ussian_UTF-8).tar :  55886848 ->   9323673 (5.994),  1.26 MB/s , 778.0 MB/s
```

Notes:
- Latest available '**oo2core_8_win64.dll**' was used;
- **Oodle** Compressors: Kraken = 8 (Default); Leviathan = 13 (Best); Mermaid = 9 (Crazy fast); Selkie = 11 (F
- **Oodle** Levels: HyperFast = -4..-1; None = 0; SuperFast = 1; VeryFast = 2; Fast = 3; Normal = 4; Optimal1,
- The tweak value **spacespeedtradeoff**=[64..1024] "tweak size vs decode time" has not been played with!
- If you wanna see more performers/options, write me to sanmayce@sanmayce.com, will try to add them.

By the way, is someone aware of existing an LZ approach dealing with the stream bi-directionally (not from left-to-right but from left-to-the-right and right-to-the-left until they converge)? Inspired by the kanji 中 denoting center [|] and 乙成, 音成 I look forward to implementing either [<-|->] or [-> | <-] LZ type of parsing while the left and right streams will share matches and will be dealt back-to-back at a loop level - hoping to improve compression (there will be flags for using left-or-right stream and using offset from last offset, plus keeping 1TB in both directions thus having 2TB window) and reduce latency during fetching uncached RAM ...

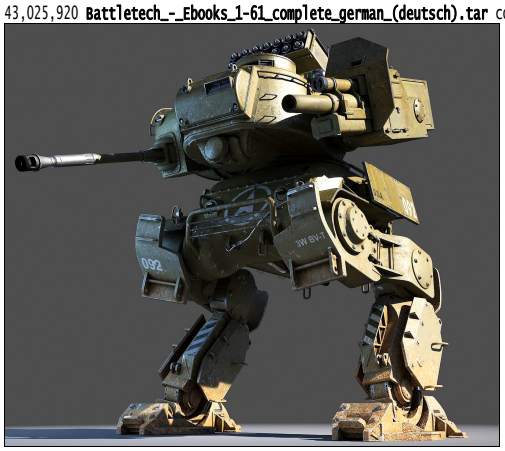
Incoming... Nakamichi 'Otonari', to put it poetically, 'The-Whisper-Of-Sword's-Soul' or '**Sword's Breath**' since soul etymology is breath. "Could there be a hidden wordplay with 乙成, "duplicate, strange, the latter, witty" and "become, elapse, get, grow, reach, turn into"? My goal is to use 'Otonari' as a name of one new compression algorithm based on Lempel-Ziv but cutting the data in half and since DUPLICATE GET/REACH depicts the process I wanna go with 乙成, a pun I want to have, that is." Source: https://www.quora.com/Having-seen-a-Japanese-saber-master-saying-otonari-meaning-the-hissing-sound-coming-from-cutting-the-air-I-ask-how-to-write-it-Why-when-searching-for-otonari-wiktionary-the-definition-is-a-fay-a-fairy

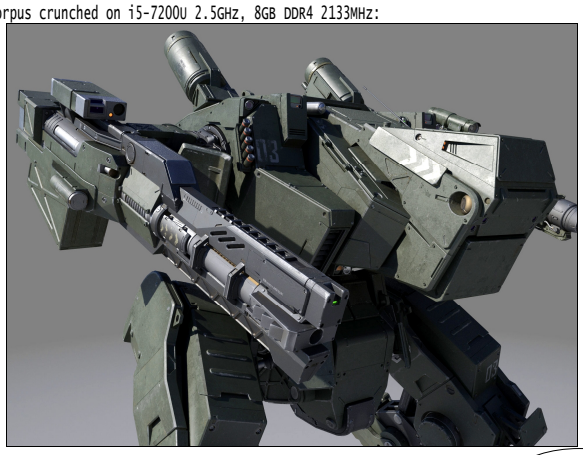


Nakamichi 'Dragoneye' LITE a.k.a. **Satanichi** highlights:
- The Zennish LZSS Microdeduplicator, written in plain C;
- The first Lempel-Ziv-Storer-Szymanski (LZSS) compressor using 1TB sliding window;
- It targets superfast decompression of huge/tera corpora of textual data;
- Compressing at Linear Rate due to Matchfinding based on B-trees only;
- The Internal/External B-trees order 3 (2 keys MAX) are brutally-optimized;
- Ability to deduplicate (as little as) 64 bytes long chunks 1TB backwards;
- Single-threaded Non-SIMD console tool written entirely by yours truly Sanmayce;
- SCALABLE! Gets faster when more Physical or/and External RAM is available;
- Dynamical (as command line parameters) selection of HASH & B-trees memory blocks;
- Compileable under Windows and Linux, most suitable to run on 3TB machines;
- Many thanks go to: Haruhiko Okumura, Nobuo Ito, Hamid Buzidi, Landon Noll.

TEXTORAMIC Decompression Showdown — SNAPPY vs LZO vs ZLIB vs ZSTD vs LZTURBO vs OODLE vs LIZARD vs Nakamichi

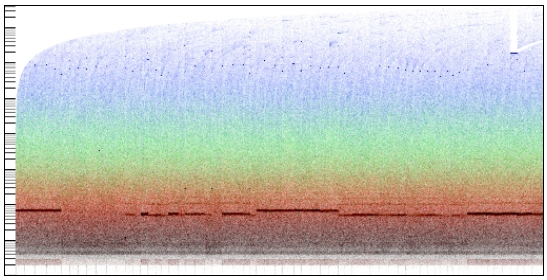
43,025,920 **Battletech_-_Ebooks_1-61_complete_german_(deutsch).tar** corpus crunched on i5-7200U 2.5GHz, 8GB DDR4 2133MHz:





中道乙成

```
D:\TEXTUAL_MADNESS_bare-minimum_2020_May_22cc-TIMER64 Satanichi_GCC730_64bit.exe ""Battletech_-_Ebooks_1-
61_complete_german_(deutsch).tar"" ""Battletech_-_Ebooks_1-61_complete_german_(deutsch).tar".Nakamichi" 20 6000 i
...
Leprechaun: Total Searches-n-Inserts Per Second: 33,806,011 SNIPS = 41,818,036,590 keys in 1,237 seconds
Leprechaun: RAM needed to house B-trees (relative to the file being ripped): 44N = 1,827MB
Leprechaun: RAM needed to build B-trees IN ONE PASS: (Target-Buffer = 73 MB) x 128 passes = 9,344MB
Note: In case of RAM in spades then may use compile option: -DSpeedupBuilding=9344
```



The struture of the corpus is shown in the diagram above. Each pixel represents a match between consecutive occurrences of a string. The color of the pixel represents the length of the match: black for 1 byte, red for 2, green for 4 and blue for 8. The horizontal axis represents the position of the second occurrence of the string. The vertical axis represents the distance back to the match on a logarithmic scale. (The image was generated by the fv program).
http://www.mattmahoney.net/dc/dce.html

```
D:\TEXTUAL_MADNESS_bare-minimum_2020_May_22>lzbench18 -c4 -i1,15 -o3 -esnappy/lizard,49/zlib,9/lzsse2,17/lzo1b,999/libdeflate,12 ""Battletech_-_Ebooks_1-61_complete_german_(de
lzbench 1.8 (64-bit Windows)   Assembled by P.Skibinski
Compressor name         Compress. Decompress.  Compr. size  Ratio Filename
lizard 1.0 -49           1.28 MB/s   679 MB/s    12533423  29.13 Battletech_-_Ebooks_1-61_complete_german_(deuts
Nakamichi 2020-May-09                873 MB/s    13345810
libdeflate 1.3 -12       5.38 MB/s   648 MB/s    15241772  35.42 Battletech_-_Ebooks_1-61_complete_german_(deuts
lzsse2 2019-04-18 -17    5.90 MB/s  3035 MB/s    15819561  36.77 Battletech_-_Ebooks_1-61_complete_german_(deuts
zlib 1.2.11 -9           8.09 MB/s   240 MB/s    15963537  37.10 Battletech_-_Ebooks_1-61_complete_german_(deuts
lzo1b 2.10 -999          6.18 MB/s   425 MB/s    17413727  40.47 Battletech_-_Ebooks_1-61_complete_german_(deuts
snappy 2019-09-30         201 MB/s   717 MB/s    26233979  60.97 Battletech_-_Ebooks_1-61_complete_german_(deuts
memcpy                  10648 MB/s 10773 MB/s    43025920 100.00 Battletech_-_Ebooks_1-61_complete_german_(deuts
```

```
D:\TEXTUAL_MADNESS_bare-minimum_2020_May_22>"turbobench_v18.05_-_build_04_May_2018" ""Battletech_-_Ebooks_1-61_co
      C Size  ratio%    C MB/s    D MB/s  Name          File
     8395845    19.5      7.68     39.30  lzturbo 59    Battletech_-_Ebooks_1-61_complete_german_(deutsch).t
    10582769    24.6      0.95    628.62  lzturbo 39    Battletech_-_Ebooks_1-61_complete_german_(deutsch).t
    10671186    24.8      0.33    548.95  oodle 139     Battletech_-_Ebooks_1-61_complete_german_(deutsch).t
    11875204    27.6      0.97   1018.56  lzturbo 29    Battletech_-_Ebooks_1-61_complete_german_(deutsch).t
    12716329    29.6      0.88   1369.95  oodle 99      Battletech_-_Ebooks_1-61_complete_german_(deutsch).t
    13345810                      873     Nakamichi 2020-May-09
    17924969    41.7      1.12   3108.14  lzturbo 19    Battletech_-_Ebooks_1-61_complete_german_(deutsch).t
```

```
D:\TEXTUAL_MADNESS_bare-minimum_2020_May_22cc>zstd-v1.4.5-w
--priority=rt ""Battletech_-_Ebooks_1-61_complete_german_(d
Note : switching to real-time priority te_german_(deutsch).
 1#man_(deutsch).tar :  43025920 ->  18070755 (2.381), 212.
 2#man_(deutsch).tar :  43025920 ->  15993619 (2.690), 149.5 MB/s , 639.1 MB/s
 3#man_(deutsch).tar :  43025920 ->  14843099 (2.899), 127.0 MB/s , 648.2 MB/s
 4#man_(deutsch).tar :  43025920 ->  14511168 (2.965), 100.7 MB/s , 637.5 MB/s
 5#man_(deutsch).tar :  43025920 ->  14680369 (2.931),  59.7 MB/s , 606.4 MB/s
 6#man_(deutsch).tar :  43025920 ->  14244992 (3.020),  36.4 MB/s , 630.9 MB/s
 7#man_(deutsch).tar :  43025920 ->  13734197 (3.133),  25.9 MB/s , 675.7 MB/s
 8#man_(deutsch).tar :  43025920 ->  13551218 (3.175),  20.0 MB/s , 699.8 MB/s
Nakamichi 2020-May-09            13345810                          873 MB/s
 9#man_(deutsch).tar :  43025920 ->  13317192 (3.231),  12.3 MB/s , 716.7 MB/s
10#man_(deutsch).tar :  43025920 ->  13132102 (3.276),  8.79 MB/s , 683.5 MB/s
11#man_(deutsch).tar :  43025920 ->  13029770 (3.302),  7.13 MB/s , 656.0 MB/s
12#man_(deutsch).tar :  43025920 ->  12800482 (3.361),  3.62 MB/s , 662.5 MB/s
13#man_(deutsch).tar :  43025920 ->  12441548 (3.458),  3.62 MB/s , 703.7 MB/s
14#man_(deutsch).tar :  43025920 ->  12165424 (3.537),  2.84 MB/s , 584.9 MB/s
15#man_(deutsch).tar :  43025920 ->  11966072 (3.596),  2.17 MB/s , 506.7 MB/s
16#man_(deutsch).tar :  43025920 ->  11492843 (3.744),  2.40 MB/s , 659.5 MB/s
17#man_(deutsch).tar :  43025920 ->  11239388 (3.828),  1.83 MB/s , 552.5 MB/s
18#man_(deutsch).tar :  43025920 ->  11082855 (3.882),  1.60 MB/s , 475.7 MB/s
19#man_(deutsch).tar :  43025920 ->  10927066 (3.938),  1.46 MB/s , 435.4 MB/s
20#man_(deutsch).tar :  43025920 ->  10653250 (4.039),  1.30 MB/s , 582.9 MB/s
21#man_(deutsch).tar :  43025920 ->  10589193 (4.063),  1.23 MB/s , 591.2 MB/s
22#man_(deutsch).tar :  43025920 ->  10585862 (4.064),  1.20 MB/s , 583.4 MB/s
```

Notes:
- Latest available '**oo2core_8_win64.dll**' was used;
- **Oodle** Compressors: Kraken = 8 (Default); Leviathan = 13 (Best); Mermaid = 9 (Crazy fast); Selkie = 11 (Fastest)
- **Oodle** Levels: HyperFast = -4..-1; None = 0; SuperFast = 1; VeryFast = 2; Fast = 3; Normal = 4; Optimal1,2,3,4 =
- The tweak value **spacespeedtradeoff**=[64..1024] "tweak size vs decode time" has not been played with!
- If you wanna see more performers/options, write me to sanmayce@sanmayce.com, will try to add them.

By the way, is someone aware of existing an LZ approach dealing with the stream bi-directionally (not from left-to-right but from left-to-the-right and right-to-the-left until they converge)? Inspired by the kanji 中 denoting center [|] and 乙成, 音成 I look forward to implementing either [<-|->] or [-> | <-] LZ type of parsing while the left and right streams will share matches and will be dealt back-to-back at a loop level - hoping to improve compression (there will be flags for using left-or-right stream and using offset from last offset, plus seeing 1TB in both directions thus having 2TB window) and reduce latency during fetching uncached RAM ...

Incoming... Nakamichi 'Otonari', to put it poetically, 'The-Whisper-Of-Sword's-Soul' or '**Sword's Breath**' since soul etymology is breath. "Could there be a hidden wordplay with 乙成, "duplicate, strange, the latter, witty" and "become, elapse, get, grow, reach, turn into"? My goal is to use 'Otonari' as a name of one new compression algorithm based on Lempel-Ziv but cutting the data in half and since DUPLICATE GET/REACH depicts the process I wanna go with 乙成, a pun I want to have, that is." Source: https://www.quora.com/Having-seen-a-Japanese-saber-master-saying-otonari-meaning-the-hissing-sound-coming-from-cutting-the-air-I-ask-how-to-write-it-Why-when-searching-for-otonari-wiktionary-the-definition-is-a-fay-a-fairy

Nakamichi 'Dragoneye' LITE a.k.a. **Satanichi** highlights:
- The Zennish LZSS Microdeduplicator, written in plain C;
- The first Lempel-Ziv-Storer-Szymanski (LZSS) compressor using 1TB sliding window;
- It targets superfast decompression of huge/tera corpora of textual data;
- Compressing at Linear Rate due to Matchfinding based on B-trees only;
- The Internal/External B-trees order 3 (2 keys MAX) are brutally-optimized;
- Ability to deduplicate (as little as) 64 bytes long chunks 1TB backwards;
- Single-threaded Non-SIMD console tool written entirely by yours truly Sanmayce;
- SCALABLE! Gets faster when more Physical or/and External RAM is available;
- Dynamical (as command line parameters) selection of HASH & B-trees memory blocks;
- Compileable under Windows and Linux, most suitable to run on 3TB machines;
- Many thanks go to: Haruhiko Okumura, Nobuo Ito, Hamid Buzidi, Landon Noll.