Broadword Computing and Fibonacci Code speed up Compressed Suffix Arrays

Simon Gog

Institut of Theoretical Computer Science
Ulm University

June 5, 2009

Suffix Array

Definition

- Given: Text $\mathcal{T}$ of length n over alphabet Σ. Last character ,,$“
- Substring $\mathcal{T}[i..n]$ is called ith suffix of $\mathcal{T}$.
- The suffix array SA of $\mathcal{T}$ is an array of length n.
  $SA[i]$ equals the lex. ith smallest suffix of $\mathcal{T}$.

Example

$\mathcal{T}$	=	u	m	u	l	m	u	m	u	l	m	$
		0	1	2	3	4	5	6	7	8	9	10
SA	=	10	8	3	9	6	1	4	7	2	5	0
		0	1	2	3	4	5	6	7	8	9	10

Suffix Array

$\mathcal{T}$= `umulmumulm$`

5 10

i	$SA[i]$	$\mathcal{T}_{SA[i]}$
0	10	`$`
1	8	`lm$`
2	3	`lmumulm$`
3	9	`m$`
4	6	`mulm$`
5	1	`mulmumulm$`
6	4	`mumulm$`
7	7	`ulm$`
8	2	`ulmumulm$`
9	5	`umulm$`
10	0	`umulmumulm$`

Properties

- Time to calculate: $\mathcal{O}(n)$
- Space: $\mathcal{O}(n \log n)$ bits $= 4n$ bytes in practice

Applications

- String matching
- Compression of strings (BWT,LZ-factorization),...

Suffix Array

$\mathcal{T}$= `umulmumulm$`

5 10

i	$SA[i]$	$\mathcal{T}_{SA[i]}$
0	10	`$`
1	8	`lm$`
2	3	`lmumulm$`
3	9	`m$`
4	6	`mulm$`
5	1	`mulmumulm$`
6	4	`mumulm$`
7	7	`ulm$`
8	2	`ulmumulm$`
9	5	`umulm$`
10	0	`umulmumulm$`

Properties

- Time to calculate: $\mathcal{O}(n)$
- Space: $\mathcal{O}(n \log n)$ bits $= 4n$ bytes in practice

Applications

- String matching
- Compression of strings (BWT,LZ-factorization),...

Drawback

Text $\mathcal{T}$ occupies $n \log |\Sigma|$ bits, SA $n \log n$ bits

Motivation for Compressed Suffix Arrays (CSAs)

Example 1

The human genome *HG* consists of approx. 3 billion DNA base pairs over the alphabet $\Sigma = \{A, C, G, T\}$.

- *HG* takes $2 \cdot 3 \cdot 10^9$ bits $\approx$ 715 MBytes
- *SA* of *HG* takes $3 \cdot 10^9 \log(10^9)$ bits $\approx$ 3.48 GBytes

Example 2 (Grossi et al., 2003)

„If we index a **4 GByte** ASCII file of Associated Press news in this manner (compressed suffix tree in $2n \log |\Sigma|$ bits), it requires **12 GBytes**, which includes explicit storage of the text. [...] If we index the Associated Press file using Sadakane's index, we need roughly **1.6 GBytes** of storage, since we no longer have to store the text."

Basics of Compressed Suffix Arrays

$\mathcal{T}$= `umulmumulm$`

`|5 |10`

i	$SA[i]$	$SA[i]^{-1}$	$\Psi(i)$	$\mathcal{T}[SA[i]]$
0	10	10	10	\$
1	8	5	3	l
2	3	8	6	l
3	9	2	0	m
4	6	6	7	m
5	1	9	8	m
6	4	4	9	m
7	7	7	1	u
8	2	1	2	u
9	5	3	4	u
10	0	0	5	u

Ψ-function

$$\begin{aligned} & SA[\Psi(i)] &= SA[i] + 1 \\ \Leftrightarrow \quad & \Psi(i) &= SA^{-1}[SA[i] + 1] \end{aligned}$$

Generalization

$$SA[i] = SA[\Psi^k(i)] - k$$

Lemma

If $\mathcal{T}[SA[i]] = \mathcal{T}[SA[i-1]]$

$$\Rightarrow \Psi(i) > \Psi(i-1)$$

Ψ consists of $|\Sigma|$ increasing sequences.

Compressing the Ψ-function

$\mathcal{T}$= `umulmumulm$`

|5 |10

i	$\Psi(i)$	$d_\Psi = \Psi(i) - \Psi(i-1)$	d_Ψ mod 11	$c_\Phi(d_\Psi$ mod 11)
0	10	10	10	010011
1	3	-7	4	1011
2	6	3	3	0011
3	0	-6	5	00011
4	7	7	7	01011
5	8	1	1	11
6	9	1	1	11
7	1	-8	3	0011
8	2	1	1	11
9	4	2	2	011
10	5	1	1	11

Compress the Ψ-function

- Determine d_Ψ

Compressing the Ψ-function

$\mathcal{T}$= `umulmumulm$`

|5 |10

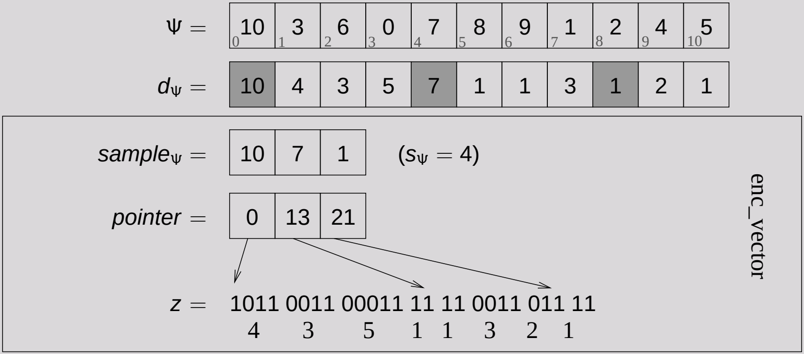
i	$\Psi(i)$	$d_\Psi = \Psi(i) - \Psi(i-1)$	d_Ψ mod 11	$c_\Phi(d_\Psi$ mod 11)
0	10	10	10	010011
1	3	-7	4	1011
2	6	3	3	0011
3	0	-6	5	00011
4	7	7	7	01011
5	8	1	1	11
6	9	1	1	11
7	1	-8	3	0011
8	2	1	1	11
9	4	2	2	011
10	5	1	1	11

Compress the Ψ-function

- Determine d_Ψ
- Encode d_Ψ with a self-delimiting code to a bitstring z
- Store every $s_\Psi th$ value of Ψ (Ψ-samples)
- Store for every Ψ-sample a pointer to the corresponding position in z

Compressing the Ψ-function: Example

Example (z encoded with Fibonacci code)



$\Psi =$

10	3	6	0	7	8	9	1	2	4	5
0	1	2	3	4	5	6	7	8	9	10

$d_\Psi =$

10	4	3	5	7	1	1	3	1	2	1

$sample_\Psi =$

10	7	1

$(s_\Psi = 4)$

$pointer =$

0	13	21

$z =$ 1011 0011 00011 11 11 0011 011 11

4 3 5 1 1 3 2 1

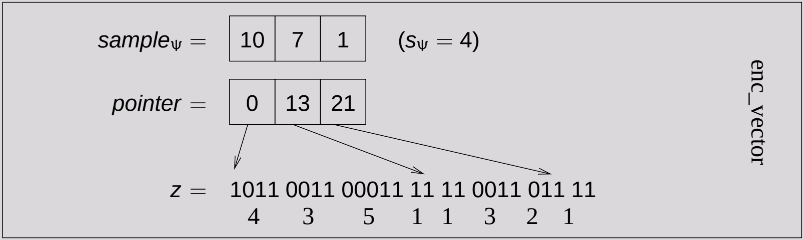
enc_vector

$$\Psi[7] = sample_\Psi[\lfloor 7/4 \rfloor] + decode(z, pointer[\lfloor 7/4 \rfloor], 7 \mod 4)$$

Compressing the Ψ-function: Example

Example (z encoded with Fibonacci code)

	0	1	2	3	4	5	6	7	8	9	10
$\Psi =$	10	3	6	0	7	8	9	1	2	4	5
$d_\Psi =$	▨ 10	4	3	5	▨ 7	1	1	3	▨ 1	2	1



$sample_\Psi =$	10	7	1
$pointer =$	0	13	21

($s_\Psi = 4$)

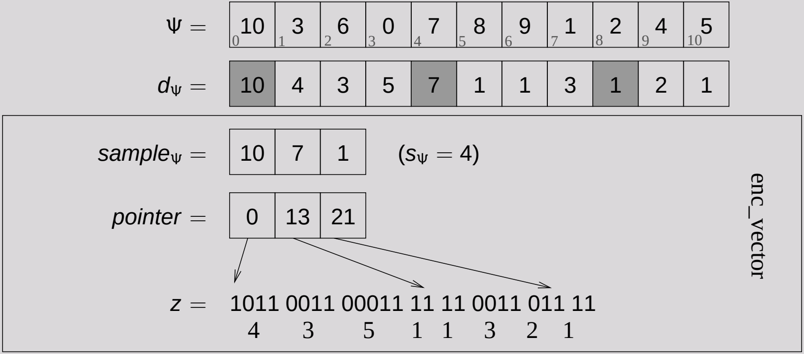
$z =$ 1011 0011 00011 11 11 0011 011 11

4 3 5 1 1 3 2 1

$$\Psi[7] = sample_\Psi[1] + decode(z, pointer[1], 3)$$

Compressing the Ψ-function: Example

Example (z encoded with Fibonacci code)



$\Psi =$	10	3	6	0	7	8	9	1	2	4	5
	0	1	2	3	4	5	6	7	8	9	10
$d_\Psi =$	10	4	3	5	7	1	1	3	1	2	1

$sample_\Psi =$	10	7	1
$pointer =$	0	13	21

($s_\Psi = 4$)

$z =$ 1011 0011 00011 11 11 0011 011 11

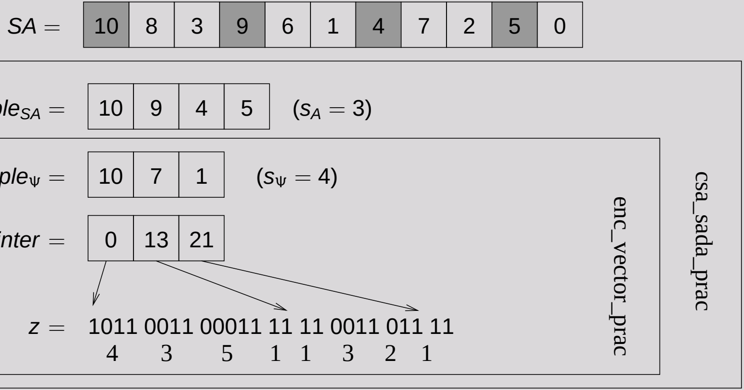
4 3 5 1 1 3 2 1

enc_vector

$$\Psi[7] = 7 + (1 + 1 + 3) = 12 = 1 \mod 11$$

Compressed Suffix Array: Example

Example for $\mathcal{T}$= `umulmumulm$`



$SA =$

10	8	3	9	6	1	4	7	2	5	0

$sample_{SA} =$

10	9	4	5

$(s_A = 3)$

$sample_{\Psi} =$

10	7	1

$(s_{\Psi} = 4)$

$pointer =$

0	13	21

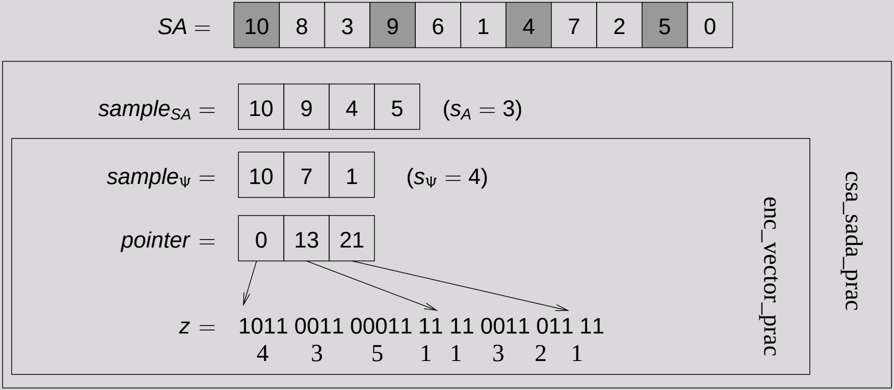
$z =$

1011	0011	00011	11	11	0011	011	11
4	3	5	1	1	3	2	1

$$SA[2] = SA[\Psi(2)] - 1$$

Compressed Suffix Array: Example

Example for $\mathcal{T}$= `umulmumulm$`



$SA =$

10	8	3	9	6	1	4	7	2	5	0

$sample_{SA} =$

10	9	4	5

$(s_A = 3)$

$sample_\Psi =$

10	7	1

$(s_\Psi = 4)$

$pointer =$

0	13	21

$z =$

1011	0011	00011	11	11	0011	011	11
4	3	5	1	1	3	2	1

$$SA[2] = SA[sample_\psi[0] + 4 + 3] - 1$$

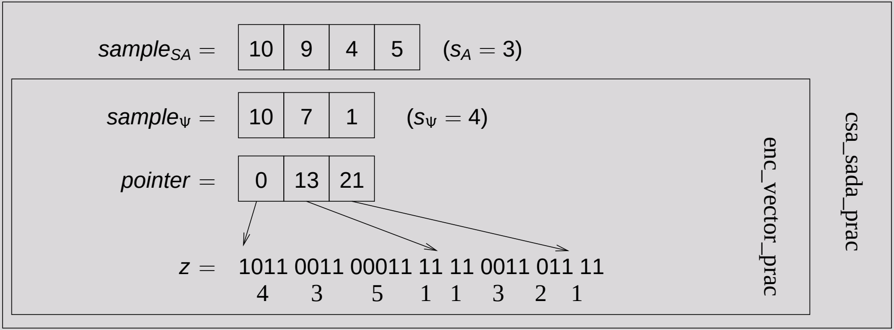
Compressed Suffix Array: Example

Example for $\mathcal{T}$= `umulmumulm$`

|5 |10

$SA =$

10	8	3	9	6	1	4	7	2	5	0



$sample_{SA} =$

10	9	4	5

$(s_A = 3)$

$sample_{\Psi} =$

10	7	1

$(s_{\Psi} = 4)$

$pointer =$

0	13	21

$z =$

1011	0011	00011	11	11	0011	011	11
4	3	5	1	1	3	2	1

$$SA[2] = SA[10 + 4 + 3 \mod 11] - 1$$

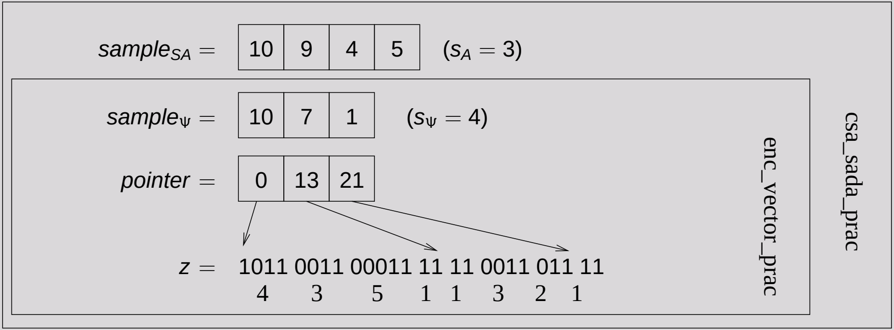
Compressed Suffix Array: Example

Example for $\mathcal{T}$= `umulmumulm$`

`|5` `|10`

$SA =$

10	8	3	9	6	1	4	7	2	5	0



$sample_{SA} =$

10	9	4	5

$(s_A = 3)$

$sample_{\Psi} =$

10	7	1

$(s_{\Psi} = 4)$

$pointer =$

0	13	21

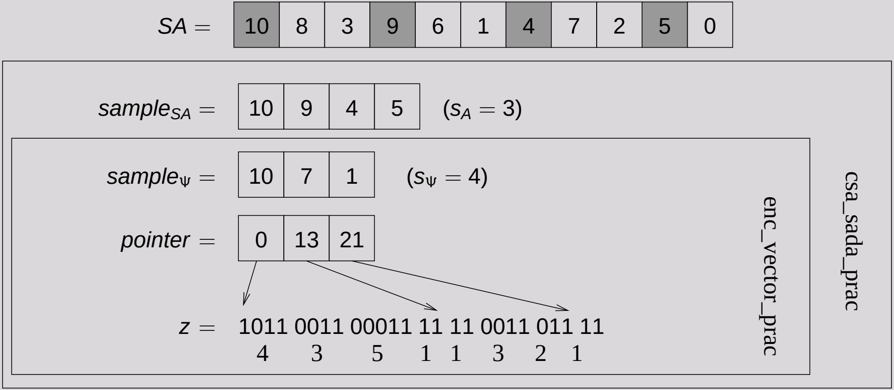
$z =$

1011	0011	00011	11	11	0011	011	11
4	3	5	1	1	3	2	1

$$SA[2] = SA[6] - 1$$

Compressed Suffix Array: Example

Example for $\mathcal{T}$= `umulmumulm$`



$SA =$

10	8	3	9	6	1	4	7	2	5	0

$sample_{SA} =$

10	9	4	5

$(s_A = 3)$

$sample_{\Psi} =$

10	7	1

$(s_{\Psi} = 4)$

$pointer =$

0	13	21

$z =$ 1011 0011 00011 11 11 0011 011 11

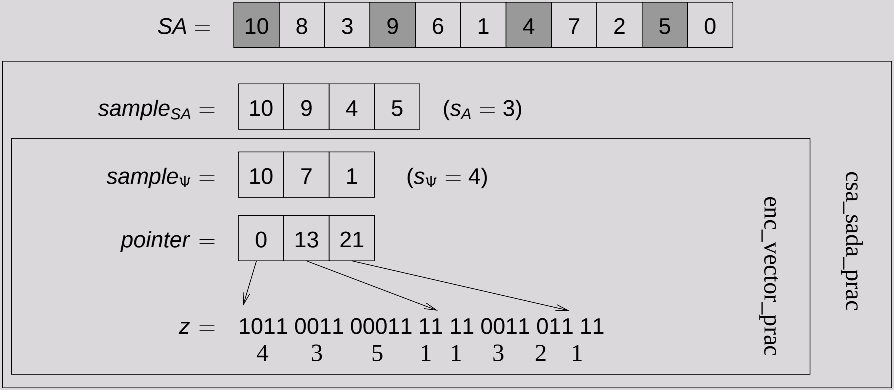
4 3 5 1 1 3 2 1

$$SA[2] = sample_{SA}[2] - 1$$

Compressed Suffix Array: Example

Example for $\mathcal{T}$= `umulmumulm$`

`|5` `|10`



$SA =$

10	8	3	9	6	1	4	7	2	5	0

$sample_{SA} =$ ($s_A = 3$)

10	9	4	5

$sample_{\Psi} =$ ($s_{\Psi} = 4$)

10	7	1

$pointer =$

0	13	21

$z =$ 1011 0011 00011 11 11 0011 011 11

4 3 5 1 1 3 2 1

$$SA[2] = 4 - 1 = 3$$

Self-Delimiting Codes

- Size and access time of CSA depend on s_A, s_Ψ and the choice of the self-delimiting code.

Self-delimiting codes

- Elias δ-code: $c_\delta(x) = \underbrace{0\cdots0}_{|\tilde{b}(b(x))|}1\tilde{b}(|b(x)|)\tilde{b}(x)$
- Fibonacci code $c_\Phi(x)$: Representation to the base of Fibonacci numbers and add one 1 at the end.

Example

x	$c_\Phi(x)$	$c_\delta(x)$
1	11	1
2	011	01 0 0
3	0011	01 0 1
4	1011	01 1 00
5	00011	01 1 01

x	$c_\Phi(x)$	$c_\delta(x)$
6	10011	01 1 10
7	01011	01 1 11
8	000011	001 00 000
9	100011	001 00 001
10	010011	001 00 010

Fibonacci Code and Broadword Computing

Bitsequence z is stored in a array of 64 bit words.

Properties

- $|c_{\Phi}(x)| \leq |c_{\delta}(x)|$ for $1 < x < 6765$
- Supported by two new broadword functions
  - **`b11Cnt(x)`**, get number of Φ-encoded numbers in a 64 bit word **`x`**.
  - **`i11BP(x,i)`**, get the end position of the ith Φ-encoded number in **`x`**.

Decoding of the sum of k encoded numbers

- Check with **`b11Cnt`** how many words have to be decoded.
- Calculate with **`i11BP(x,i)`** how many bits are masked in the last word.
- Use lookup tables for 8 or 16 bits to decode the sum.

Implementation and experimental results

Implementation

- New template C++ library *sdsl* contains data structures for
    - bit vector, integer vector, rank/select
    - coders: Fibonacci coder, Elias-δ coder,...
    - 2 Compressed Suffix Arrays
    - 2 Compressed Suffix Trees
    - ...

Experiments

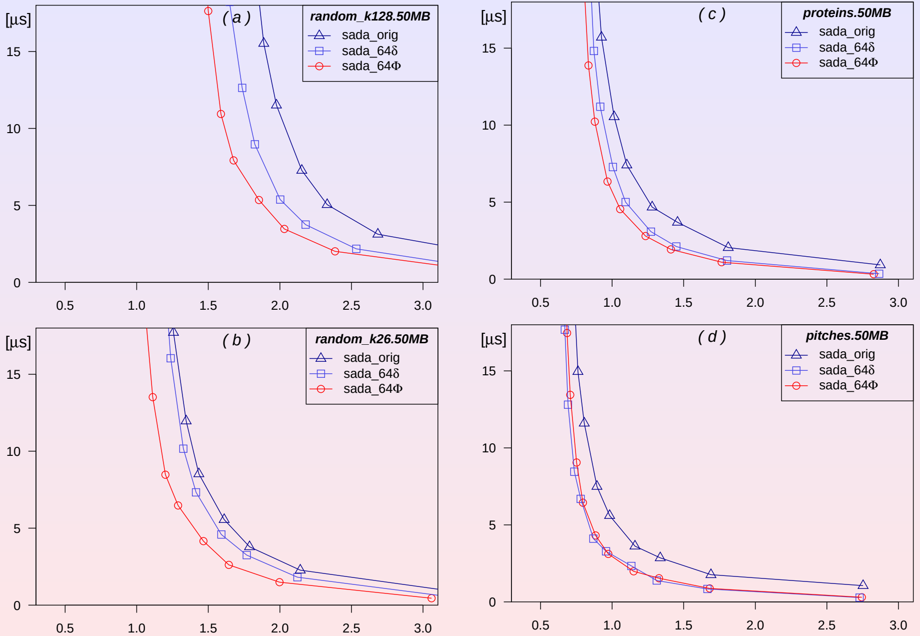
- We used test cases from *Pizza&Chili* website.
- Comparison of our CSA implementation parametrized with different coders.
- Comparison of our CSA implementation with Sadakane's implementation.
- Access time / space tradeoff

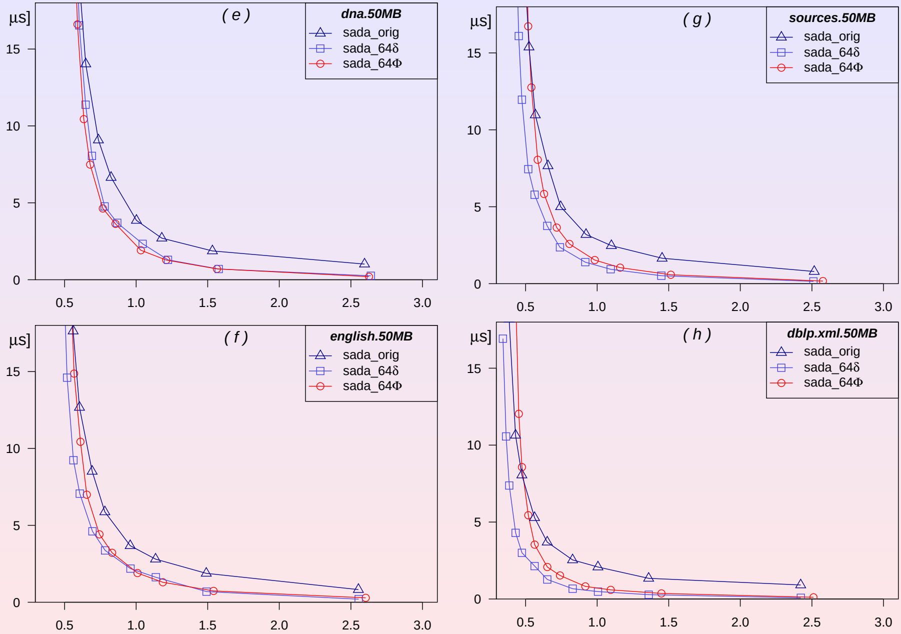
Test cases

Test case	$\|\Sigma\|$	Compression by `ppmdi -l 9`	Encoded ones in z	Encoded values < 32 in z
`random_k128.50MB`	128	0.894	< 0.01	≈ 0.21
`random_k26.50MB`	26	0.698	≈ 0.03	≈ 0.70
`proteins.50MB`	27	0.421	≈ 0.36	≈ 0.82
`pitches.50MB`	133	0.305	≈ 0.37	≈ 0.83
`dna.50MB`	16	0.243	≈ 0.57	≈ 0.99
`english.50MB`	239	0.242	≈ 0.68	≈ 0.94
`sources.50MB`	230	0.167	≈ 0.76	≈ 0.94
`dblp.xml.50MB`	97	0.092	≈ 0.85	≈ 0.97

Parameter choice for CSAs

$s_\Psi = 128$ and $s_A = \{2, 4, 6, 8, 12, 16, 24, 32, 48, 64, 96\}$


[µs]
(a)
random_k128.50MB
sada_orig
sada_64δ
sada_64Φ
15
10
5
0
0.5
1.0
1.5
2.0
2.5
3.0
(c)
proteins.50MB
(b)
random_k26.50MB
(d)
pitches.50MB


(e)
dna.50MB
sada_orig
sada_64δ
sada_64Φ
µs]
15
10
5
0
0.5
1.0
1.5
2.0
2.5
3.0
(g)
sources.50MB
sada_orig
sada_64δ
sada_64Φ
µs]
15
10
5
0
0.5
1.0
1.5
2.0
2.5
3.0
(f)
english.50MB
sada_orig
sada_64δ
sada_64Φ
µs]
15
10
5
0
0.5
1.0
1.5
2.0
2.5
3.0
(h)
dblp.xml.50MB
sada_orig
sada_64δ
sada_64Φ
µs]
15
10
5
0
0.5
1.0
1.5
2.0
2.5
3.0

Summary

- Size and access time to a the CSA depend on paramter s_Ψ, s_A and the choice of the self-delimiting code for d_Ψ.
- Broadword methods accelerate decoding
  - Fibonacci code
  - Elias-δ code (case with much encoded ones)
- Rule of thumb: Use Fibonacci code for text with high entropy and Elias-δ code for highly-compressible files.