Succinct Data Structures for All

Daniel Saad Nogueira Nunes Mauricio Ayala-Rincón

Grupo de Teoria da Computação
Universidade de Brasília

5^{th} February 2015

VII Workshop de Matemática Aplicada — MAT/UnB

XII Seminário informal (,mas formal!) do Grupo de Teoria da Computação da UnB — GTC/UnB

Summary

Summary

> There are more problems than people.
>
> Ricardo Baeza-Yates

Summary

1. Introduction
   - Memory Hierarchy
   - Basic Concepts

Memory Hierachy

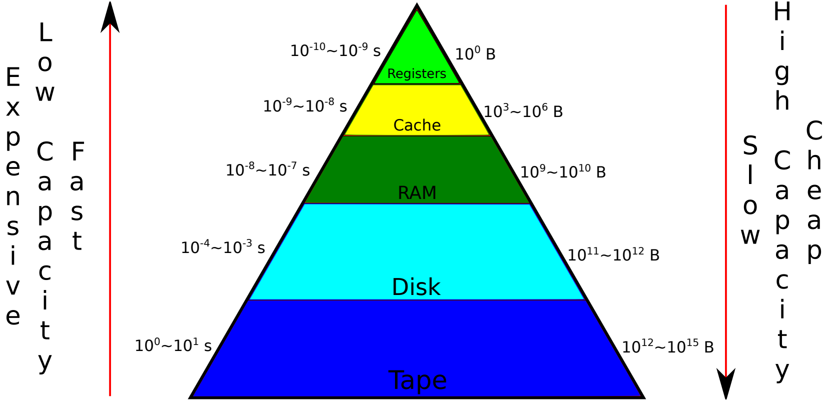

Figure : Memory Hierarchy.

Memory Hierachy

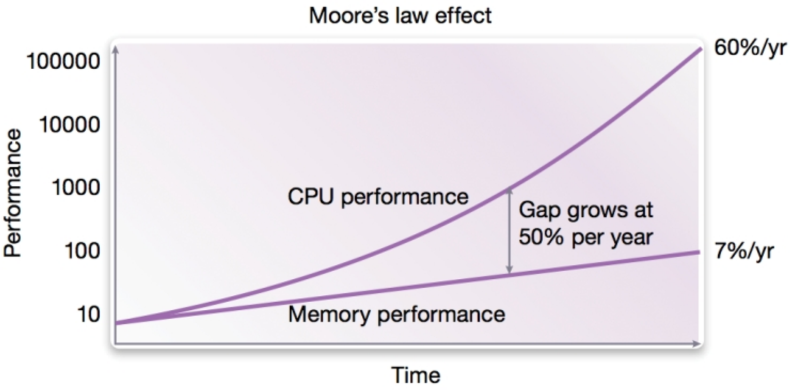

Figure : Gap between memory and cpu performance.

Memory Hierachy

Motivation

- Moore's law: # transistors grow exponentially.
- CPU speed and Memory capacity grows as well.
- Memory Access does not share the same result!
- We should use faster memories until we can!

Introduction

Definition (Succinct Data Structures)

- A Succinct Data Structure ($\mathcal{SDS}$) uses an amount of space that is "close" to the information-theoretic lower bound. [Jac89]
- But also allows efficient queries!
- The use of $\mathcal{SDS}$s is encouraged in the actual scenario.

Summary

1. Introduction
   - Memory Hierarchy
   - Basic Concepts

Basic Concepts

Empirical Entropy

- Defined for every finite and individual string.
- Can be used to measure the performance of compression algorithms.
- Does not take in account the input distribution.

Basic Concepts

Definition (Zeroth Order Empirical Entropy)

- Zeroth order entropy can be defined as:

$$H_0(S) = -\sum_{c \in \Sigma} \frac{n_c}{n} \log \frac{n_c}{n} \tag{1}$$

  where n_c stands for the frequency of c in S.
- Does not consider any context to encode a symbol.
- nH_0: lower bound to a zero-order compressor.
- Huffman compression is bounded at nH_0 bits.

Summary

I take a whole life story and compress it into three minutes.

Harlan Howard

Bitmaps

- Bitmaps: the core of $\mathcal{SDS}$.
- A sequence $S \in \{0,1\}^*$.
- Can represent information in compact space.
- Often three operations are supported:
  - $Rank$.
  - $Select$.
  - $Access$.

Rank Queries

Definition (Rank)

$$Rank_1(B,i) = \text{number of 1's in } B[0,i]$$

$$Rank_0(B,i) = \text{number of 0's in } B[0,i]$$

- Focus on $Rank_1$, since $Rank_0(B,i) = n - Rank_1(B,i)$.

Select Queries

Definition (Select)

$$Select_1(B, i) = \text{Position of the } i^{th} \text{ 1 in } B$$

$$Select_0(B, i) = \text{Position of the } i^{th} \text{ 0 in } B$$

- Notice that $Rank(Select(x)) = x \wedge Select(Rank(x)) = x$ iff $B[x] = 1$.
- Can be answered in $O(1)$ time as well.

Rank and Select

- Time is short!
- I need you to believe that $Rank$ and $Select$ queries can be answered in $O(1)$ time and $n + o(n)$ bits.

Summary

If you think in terms of a year, plant a seed; if in terms of ten years, plant trees; if in terms of 100 years, teach the people.

Confucius

Wavelet Trees

Bitmaps Limitations

- Support only binary Σ.
- What if one wanted to answer general $Rank/Select$ queries over another alphabet?
- Example: $Rank_c(ACTAGACCTAGACGAC, 7) = 3$.
- Solution: Wavelet Trees ($\mathcal{WT}$) [GGV03].
- $\mathcal{WT}$s reduce general $Rank/Select$ queries on binary queries.

Wavelet Trees

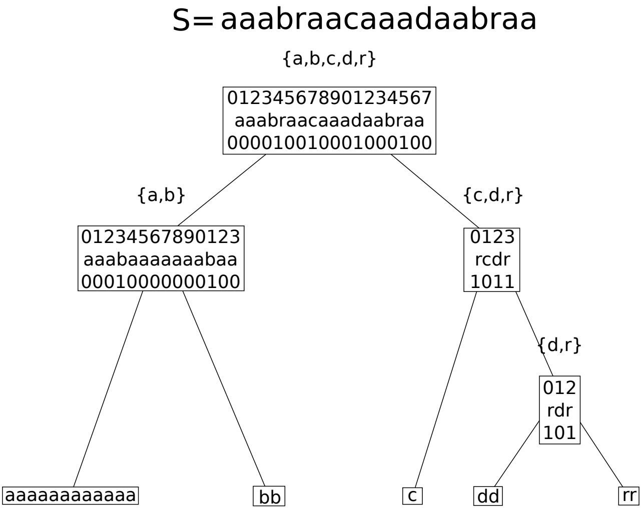

Figure : Wavelet tree for $S = aaabraacaaadaabraa$

Wavelet Trees

Bitmaps Limitations

- $Rank_b(S, 10) = ?$
- $Select_a(S, 11) = ?$
- $Access(S, 14) = ?$

Wavelet Trees

S= aaabraacaaadaabraa

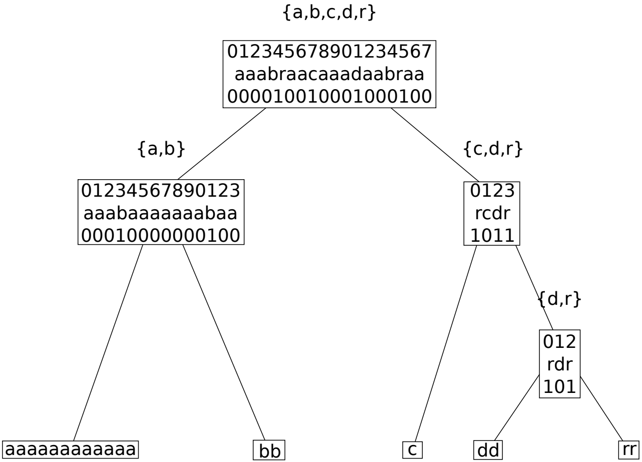

Figure : Answering $Rank_b(S, 10)$

Wavelet Trees

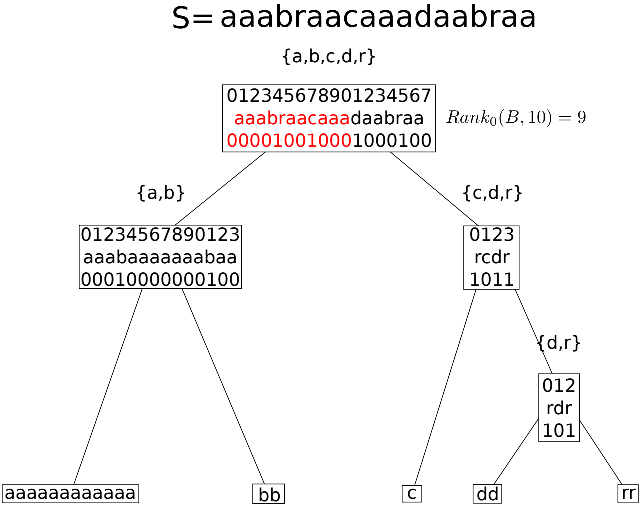

Figure : Answering $Rank_b(S, 10)$

Wavelet Trees

S= aaabraacaaadaabraa

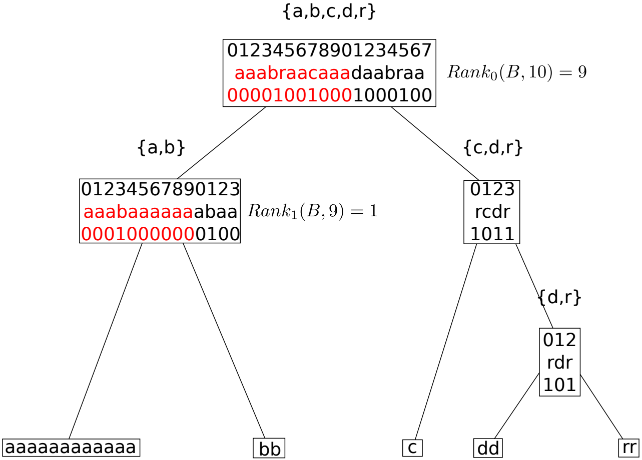

Figure : Answering $Rank_b(S,10)$

Wavelet Trees

S= aaabraacaaadaabraa

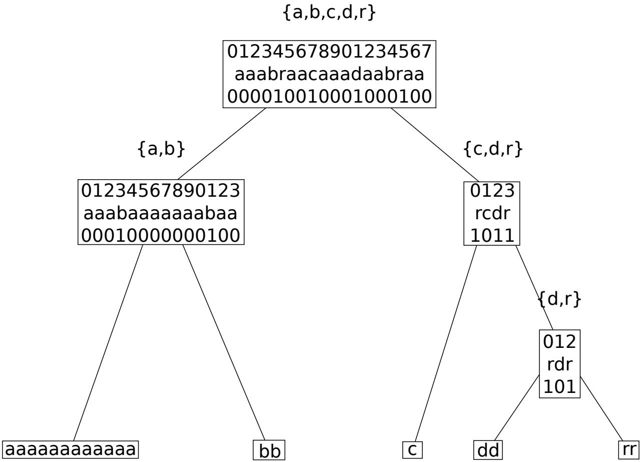

Figure : Answering $Select_a(S, 11)$

Wavelet Trees

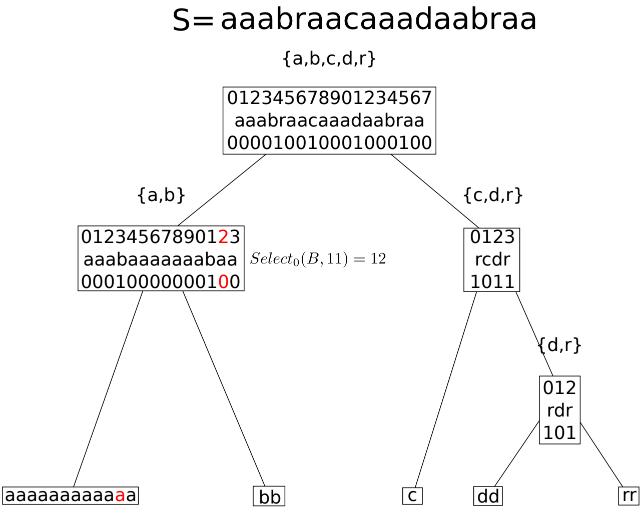

Figure : Answering $Select_a(S, 11)$

Wavelet Trees

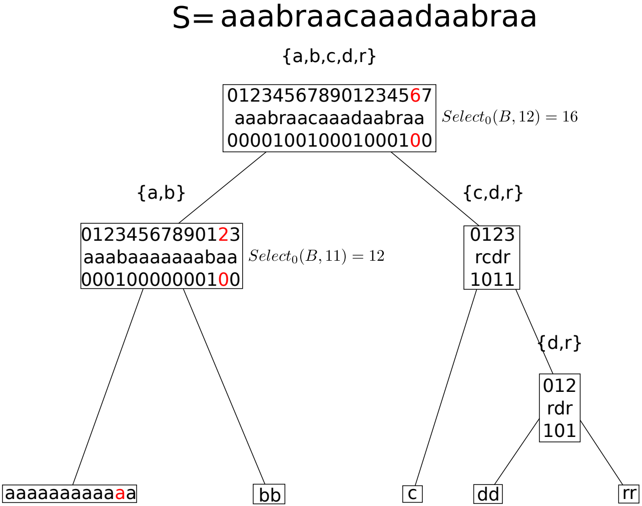

Figure : Answering $Select_a(S, 11)$

Wavelet Trees

S= aaabraacaaadaabraa

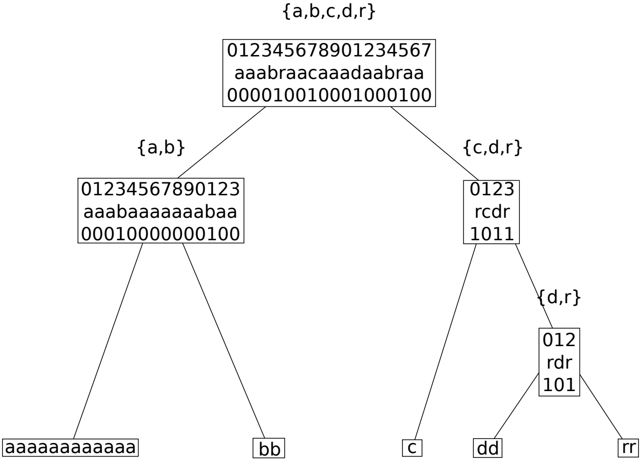

Figure : Answering $Access(S, 14)$

Wavelet Trees

S= aaabraacaaadaabraa

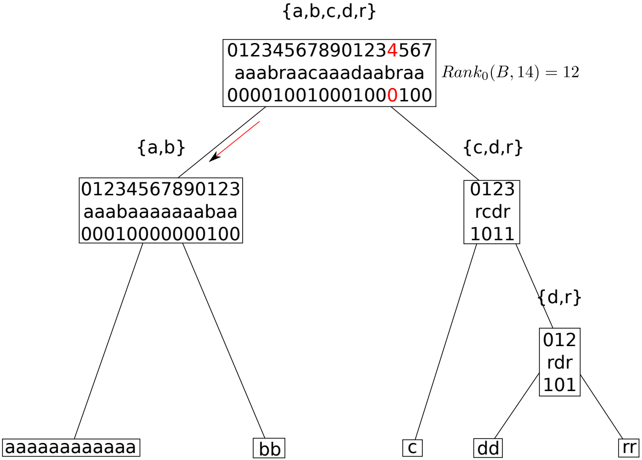

Figure : Answering $Access(S, 14)$

Wavelet Trees

S= aaabraacaaadaabraa

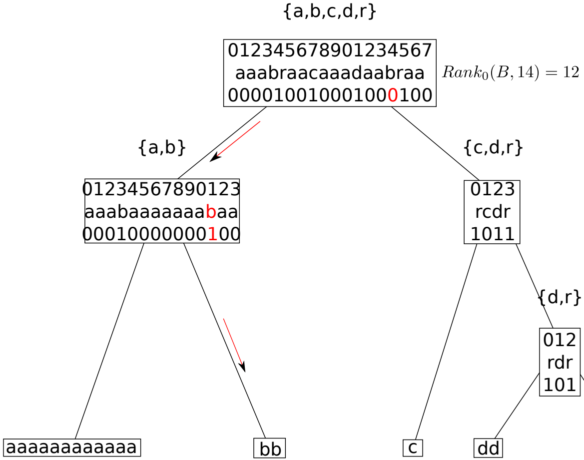

Figure : Answering $Access(S, 14)$

Wavelet Trees

Proprerties

- σ leaves and $\sigma - 1$ internal nodes.
- Height: $\lceil \log \sigma \rceil$.
- $Rank$, $Select$ and $Access$ in $O(\log \sigma)$ time.
- Space: $O(n \log \sigma + \sigma \log n)$.
- Space (with no pointers): $O(n \log \sigma)$.
- Can achieve $O(nH_0)$ bits if shaped as a Huffman tree.
- It is a **self-index**: entirely replaces the original sequence.

Wavelet Trees

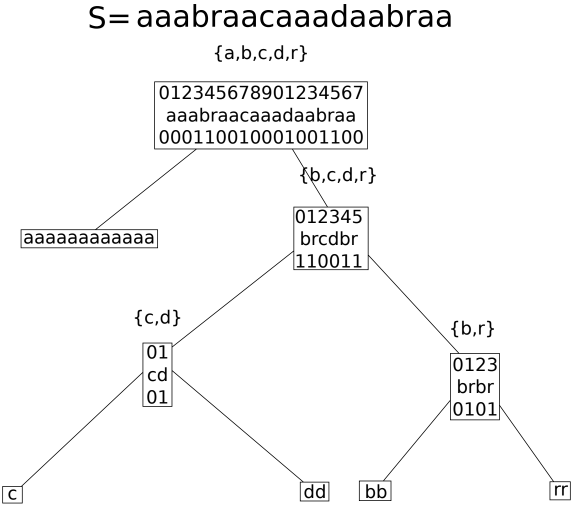

Figure : Huffman-shaped $\mathcal{WT}$.

Summary

Suffix Arrays. . . The permutation in Stringology

Roberto Grossi

Suffix Tree

- The Suffix Tree ($\mathcal{ST}$) is a well-known index in the literature which represents all the suffixes in $O(n \log n)$ bits or $O(n)$ words.
- Can be constructed in $O(n)$ time.
- Demands too much space in practice.

Suffix Tree

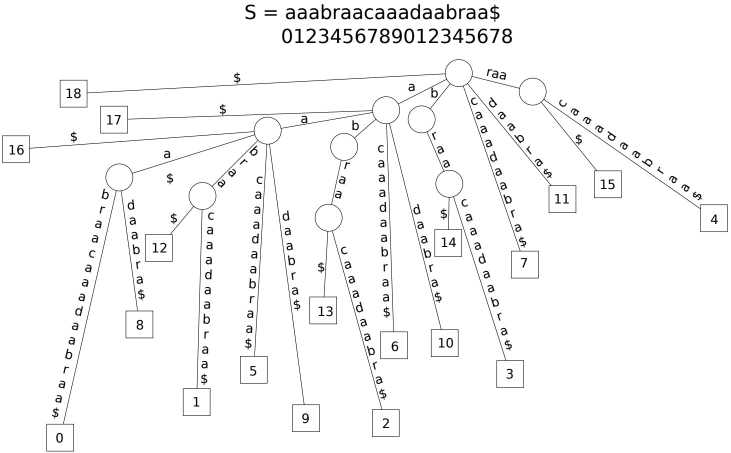

Figure : Suffix Tree for $aaabraacaaadaabraa\$$.

Suffix Array

- Suffix Array ($\mathcal{SA}$): compact alternative to $\mathcal{ST}$s.
- Integer array containing the position of suffixes in lexicographical order induced by Σ.
- Can be built in $O(n)$ time.
- Can handle bigger texts.
- Inverse $\mathcal{SA}$: Integer array containing the lexicographical order of the i^{th} suffix.

Suffix Array

Table : Suffix Array for $aaabraacaaadaabraa\$$.

i	$SA[i]$	$SA^{-1}[i]$	$T_{SA[i]}$
0	18	3	$\$$
1	17	6	$a\$$
2	16	10	$aa\$$
3	0	14	$aaabraacaaadaabraa\$$
4	8	18	$aaadaabraa\$$
5	12	7	$aabraa\$$
6	1	11	$aabraacaaadaabraa\$$
7	5	15	$aacaaadaabraa\$$
8	9	4	$aadaabraa\$$
9	13	8	$abraa\$$
10	2	12	$abraacaaadaabraa\$$
11	6	16	$acaaadaabraa\$$
12	10	5	$adaabraa\$$
13	14	9	$braa\$$
14	3	13	$braacaaadaabraa\$$
15	7	17	$caaadaabraa\$$
16	11	2	$daabraa\$$
17	15	1	$raa\$$
18	4	0	$raacaaadaabraa\$$

Suffix Array

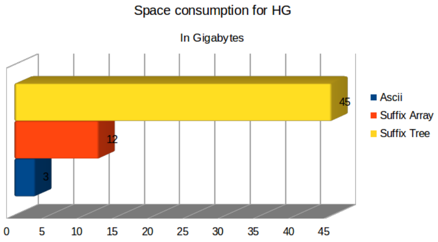

Figure : Space Consumption of Indices for HG.

- With $\mathcal{SA}$s: space consumption is still an issue.
- In order to manipulate **huge** texts one need more space-efficient data-structures!
- Compressed Indices for All!

Summary

4. Compressed Indices
   - Basic Concepts
   - $\mathcal{CSA}$
   - $\mathcal{FM}$-Index

$\mathcal{CSA}$

- The Compressed Suffix Array ($\mathcal{CSA}$) was originally developed by Grossi and Vitter [GV00].
- Main idea: sample some suffix array entries and recover other by computation.
- $\mathcal{CSA}$ core: Ψ function.

$$\Psi(i) = SA^{-1}[SA[i] + 1 \mod n]$$

- $\Psi(i)$ is piecewise crescent for suffixes starting with the same symbol.
- Allows compression by differential encoding.

Suffix Array

Table : Ψ function for $aaabraacaaadaabraa\$$.

i	$SA[i]$	$SA^{-1}[i]$	$\Psi(i)$	$T_{SA[i]}$
0	18	3	3	$\$$
1	17	6	0	$a\$$
2	16	10	1	$aa\$$
3	0	14	6	$aaabraacaaadaabraa\$$
4	8	18	8	$aaadaabraa\$$
5	12	7	9	$aabraa\$$
6	1	11	10	$aabraacaaadaabraa\$$
7	5	15	11	$aacaaadaabraa\$$
8	9	4	12	$aadaabraa\$$
9	13	8	13	$abraa\$$
10	2	12	14	$abraacaaadaabraa\$$
11	6	16	15	$acaaadaabraa\$$
12	10	5	16	$adaabraa\$$
13	14	9	17	$braa\$$
14	3	13	18	$braacaaadaabraa\$$
15	7	17	4	$caaadaabraa\$$
16	11	2	5	$daabraa\$$
17	15	1	2	$raa\$$
18	4	0	7	$raacaaadaabraa\$$

The Ψ function

- How Ψ is efficiently coded?
- Basic idea: take $\Psi(i) - \Psi(i-1)$ and apply Rice code.
- For increasing sequence $(0, 2, 5, 7, 9)$ one would have the bitmap:

$$B = \underbrace{1}_{0}\underbrace{001}_{2}\underbrace{0001}_{5}\underbrace{001}_{7}\underbrace{001}_{9}$$

- $\Psi(i) = Select_1(B, i)$.

CSA

- How recover SA entries?
- Basic idea: store explicitly only $k = \frac{n}{\log^{\epsilon} n}$ entries of SA.
  - $o(n)$ bits of space if $\epsilon > 1$
- Mark sampled entries with a 1 in a bitmap.
- Apply Ψ at maximum $l \leq k$ times until finding a sampled entry.
- $SA[i] = SA[\Psi^{l}(i)] - l \mod n$

Suffix Array

i	$SA[i]$	$SA^{-1}[i]$	$\Psi(i)$	$T_{SA[i]}$
0	18	3	3	$\$$
1	17	6	0	$a\$$
2	16	10	1	$aa\$$
3	0	14	6	$aaabraacaaadaabraa\$$
4	8	18	8	$aaadaabraa\$$
5	12	7	9	$aabraa\$$
6	1	11	10	$aabraacaaadaabraa\$$
7	5	15	11	$aacaaadaabraa\$$
8	9	4	12	$aadaabraa\$$
9	13	8	13	$abraa\$$
10	2	12	14	$abraacaaadaabraa\$$
11	6	16	15	$acaaadaabraa\$$
12	10	5	16	$adaabraa\$$
13	14	9	17	$braa\$$
14	3	13	18	$braacaaadaabraa\$$
15	7	17	4	$caaadaabraa\$$
16	11	2	5	$daabraa\$$
17	15	1	2	$raa\$$
18	4	0	7	$raacaaadaabraa\$$

- Suppose that we want to recover $SA[2]$ but only $SA[6]$ is sampled.
- $\Psi(2) = 1 \rightarrow \Psi(1) = 0 \rightarrow \Psi(0) = 3 \rightarrow \Psi(3) = 6 \rightarrow SA[6] = 1$
- $\therefore SA[2] = 1 - 4 \mod 19 = 16.$

Summary

4 Compressed Indices
- Basic Concepts
- $\mathcal{CSA}$
- $\mathcal{FM}$-Index

$\mathcal{FM}$-Index

- The $\mathcal{FM}$ family of indices is based on the Burrows-Wheeler Transform ($\mathcal{BWT}$) [FM05].
- Same idea as $\mathcal{CSA}$: sample some entries and recover others by computation.
- The $\mathcal{BWT}$ is computed by sorting the cyclical suffixes and taking the last column.

BWT

		BWT
aaabraacaaadaabraa$		$aaabraacaaadaabraa
aabraacaaadaabraa$a		a$aaabraacaaadaabra
abraacaaadaabraa$aa		aa$aaabraacaaadaabr
braacaaadaabraa$aaa		aaabraacaaadaabraa$
raacaaadaabraa$aaab		aaadaabraa$aaabraac
aacaaadaabraa$aaabr		aabraa$aaabraacaaad
acaaadaabraa$aaabra		aabraacaaadaabraa$a
caaadaabraa$aaabraa		aacaaadaabraa$aaabr
aaadaabraa$aaabraac	$\Longrightarrow$	aadaabraa$aaabraaca
aadaabraa$aaabraaca		abraa$aaabraacaaada
adaabraa$aaabraacaa		abraacaaadaabraa$aa
daabraa$aaabraacaaa		acaaadaabraa$aaabra
aabraa$aaabraacaaad		adaabraa$aaabraacaa
abraa$aaabraacaaada		braa$aaabraacaaadaa
braa$aaabraacaaadaa		braacaaadaabraa$aaa
raa$aaabraacaaadaab		caaadaabraa$aaabraa
aa$aaabraacaaadaabr		daabraa$aaabraacaaa
a$aaabraacaaadaabra		raa$aaabraacaaadaab
$aaabraacaaadaabraa		raacaaadaabraa$aaab

Figure : BWT for $S = aaabraacaaadaabraa\$$.

The $\mathcal{BWT}$

- The $\mathcal{BWT}$ is a permutation based on the original text.
- Has a close relation to $\mathcal{SA}$s.
- $BWT[i] = T[SA[i] - 1 \mod n]$.
- Same symbols tend to be grouped together.
- Eases the compression.

$\mathcal{BWT}$

- By using the $\mathcal{BWT}$ we can walk through the suffix array.

$$LF(i) = SA^{-1}[SA[i] - 1 \mod n]$$

- LF moves to the previous suffix.
- LF and Ψ are very similar: $LF(\Psi(i)) = \Psi(LF(i))$.
- $LF(i) = C[BWT[i]] + Rank_{BWT[i]}(BWT, i)$.
  - $C[i]$ contains the # of symbols in S which are lexicographically smaller than i.
  - Why?
  - Previous suffixes starting with the same symbol will retain relative order.
  - They are contiguous in the first row!

BWT

Rotations		Sorted rotations (BWT = last column)
aaabraacaaadaabraa$		$aaabraacaaadaabraa
aabraacaaadaabraa$a		a$aaabraacaaadaabra
abraacaaadaabraa$aa		aa$aaabraacaaadaabr
braacaaadaabraa$aaa		aaabraacaaadaabraa$
raacaaadaabraa$aaab		aaadaabraa$aaabraac
aacaaadaabraa$aaabr		aabraa$aaabraacaaad
acaaadaabraa$aaabra		aabraacaaadaabraa$a
caaadaabraa$aaabraa		aacaaadaabraa$aaabr
aaadaabraa$aaabraac	$\Rightarrow$	aadaabraa$aaabraaca
aadaabraa$aaabraaca		abraa$aaabraacaaada
adaabraa$aaabraacaa		abraacaaadaabraa$aa
daabraa$aaabraacaaa		acaaadaabraa$aaabra
aabraa$aaabraacaaad		adaabraa$aaabraacaa
abraa$aaabraacaaada		braa$aaabraacaaadaa
braa$aaabraacaaadaa		braacaaadaabraa$aaa
raa$aaabraacaaadaab		caaadaabraa$aaabraa
aa$aaabraacaaadaabr		daabraa$aaabraacaaa
a$aaabraacaaadaabra		raa$aaabraacaaadaab
$aaabraacaaadaabraa		raacaaadaabraa$aaab

Figure : $LF(13) = C[a] + Rank_a(BWT, 13) - 1$

BWT

			BWT
aaabraacaaadaabraa$	⇒	$aaabraacaaadaabra	a
aabraacaaadaabraa$a		a$aaabraacaaadaabr	a
abraacaaadaabraa$aa		aa$aaabraacaaadaab	r
braacaaadaabraa$aaa		aaabraacaaadaabraa	$
raacaaadaabraa$aaab		aaadaabraa$aaabraa	c
aacaaadaabraa$aaabr		aabraa$aaabraacaaa	d
acaaadaabraa$aaabra		aabraacaaadaabraa$	a
caaadaabraa$aaabraa		aacaaadaabraa$aaab	r
aaadaabraa$aaabraac		aadaabraa$aaabraac	a
aadaabraa$aaabraaca		abraa$aaabraacaaad	a
adaabraa$aaabraacaa		abraacaaadaabraa$a	a
daabraa$aaabraacaaa		acaaadaabraa$aaabr	a
aabraa$aaabraacaaad		adaabraa$aaabraaca	a
abraa$aaabraacaaada		braa$aaabraacaaada	a
braa$aaabraacaaadaa		braacaaadaabraa$aa	a
raa$aaabraacaaadaab		caaadaabraa$aaabra	a
aa$aaabraacaaadaabr		daabraa$aaabraacaa	a
a$aaabraacaaadaabra		raa$aaabraacaaadaa	b
$aaabraacaaadaabraa		raacaaadaabraa$aaa	b

Figure : $LF(13) = 1 + 9 - 1 = 9.$

$\mathcal{FM}$-Index

- What changes from one $\mathcal{FM}$-Index to another: how to represent the $\mathcal{BWT}$.
- We need to know: $C[i] + Rank_{BWT[i]}(BWT, i)$;
- Simple and fast implementation:

$$\mathcal{BWT} \rightarrow \mathcal{WT}(\mathcal{BWT})$$

- To achieve $O(nH_0)$ bits, use Huffman-shaped $\mathcal{WT}$s.

Summary

The best preparation for good work tomorrow is to do good work today.

Elbert Hubbard

Summary

5 Current Work

- Results
- To-do List

Results

- Compressed Suffix Tree with Low Peak Memory Usage [NA14].
  - Based on $\mathcal{CSA}$ and additional compressed information.
- Support of complex queries:
  - Longest Common Ancestor.
  - Suffix Link.

Build Time

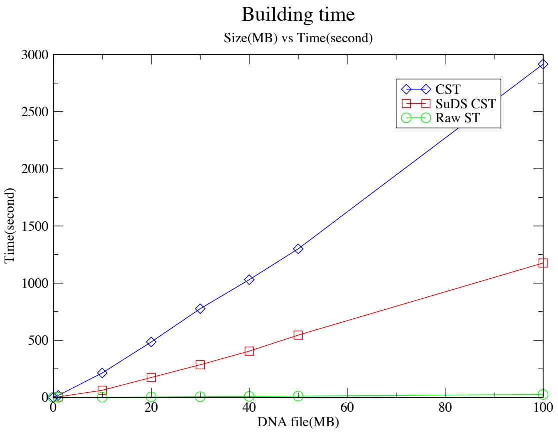

Figure : Proposed $\mathcal{CST}$ using 13 bits per symbol.

Space and Memory Peak

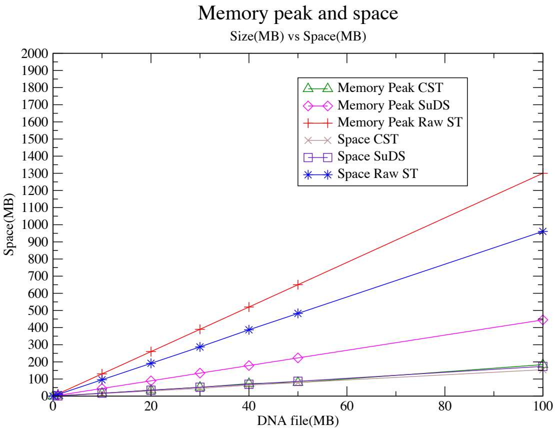

Figure : Proposed $\mathcal{CST}$ using 13 bits per symbol.

Operations

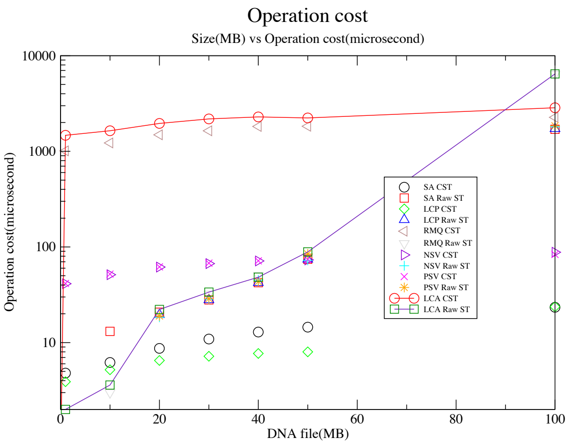

Figure : Proposed $\mathcal{CST}$ using 13 bits per symbol.

Summary

5 Current Work
- Results
- To-do List

To-do List

- Search for theoretical improvements in $\mathcal{SDS}$s, which can lead to practical usage.
- Design fast and more efficient $\mathcal{SDS}$s with low peak memory usage.
- Compare to others implementations. [ACN13, GBMP13]

Summary

References

[ACN13] Andrés Abeliuk, Rodrigo Cánovas, and Gonzalo Navarro.
Practical compressed suffix trees.
Algorithms, 6(2):319–351, 2013.

[FM05] Paolo Ferragina and Giovanni Manzini.
Indexing compressed text.
J. ACM, 52(4):552–581, 2005.

[GBMP13] Simon Gog, Timo Beller, Alistair Moffat, and Matthias Petri.
From theory to practice: Plug and play with succinct data structures.
CoRR, abs/1311.1249, 2013.

References

[GGV03] Roberto Grossi, Ankur Gupta, and Jeffrey Scott Vitter.
High-order entropy-compressed text indexes.
In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms, January 12-14, 2003, Baltimore, Maryland, USA.*, pages 841–850. ACM/SIAM, 2003.

[GV00] Roberto Grossi and Jeffrey Scott Vitter.
Compressed suffix arrays and suffix trees with applications to text indexing and string matching (extended abstract).
In F. Frances Yao and Eugene M. Luks, editors, *Proceedings of the Thirty-Second Annual ACM Symposium on Theory of Computing, May 21-23, 2000, Portland, OR, USA*, pages 397–406. ACM, 2000.

References

[Jac89] Guy Jacobson.
Space-efficient static trees and graphs.
In *30th Annual Symposium on Foundations of Computer Science, Research Triangle Park, North Carolina, USA, 30 October - 1 November 1989*, pages 549–554. IEEE Computer Society, 1989.

[NA14] Daniel Saad Nogueira Nunes and Mauricio Ayala-Rincón.
A compressed suffix tree based implementation with low peak memory usage.
Electr. Notes Theor. Comput. Sci., 302:73–94, 2014.