Massive Text Indices

Florian Kurpicz

Departments of Computer Science
KIT & TU Dortmund

Algorithms for Big Data
DFG Schwerpunktprogramm 1736

Frankfurt, September 29th – October 01st, 2014

Karlsruhe Institute of Technology

Texts are Important

A Showcase for Big Data

- World Wide Web
- digital libraries
- DNA and proteins

Texts are Important

A Showcase for Big Data

- World Wide Web
- digital libraries
- DNA and proteins

What to do with the Text

- Store
- Structure
- Search
- Compress

And all that efficiently.

Texts are Important

A Showcase for Big Data

- World Wide Web
- digital libraries
- DNA and proteins

What to do with the Text

- Store
- Structure
- Search
- Compress

And all that efficiently.

Texts are Important

A Showcase for Big Data

- World Wide Web
- digital libraries
- DNA and proteins

What to do with the Text

- Store
- Structure
- Search
- Compress

And all that efficiently.

Texts are Important

A Showcase for Big Data

- World Wide Web
- digital libraries
- DNA and proteins

What to do with the Text

- Store
- Structure
- Search
- Compress

And all that efficiently.

The Amount of Data

- Increases faster that the storage and computation capacities.
- 10 years ago: The human genome (~ 5.8 Gigabits ≈ 725 Megabyte).
- Today: Analyse 1000 human genomes simultaneously.

Texts are Important

A Showcase for Big Data

- World Wide Web
- digital libraries
- DNA and proteins

What to do with the Text

- Store
- Structure
- Search
- Compress

And all that efficiently.

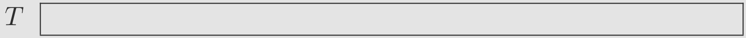
The Amount of Data

- Increases faster that the storage and computation capacities.
- 10 years ago: The human genome (~ 5.8 Gigabits ≈ 725 Megabyte).
- Today: Analyse 1000 human genomes simultaneously.
- Tomorrow: Analyse 1,000,000 human genomes simultaneously?

What do we want?

The Problem

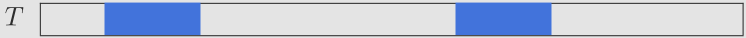
Given a text T and a pattern P find all occurrences of P in T.



What do we want?

The Problem

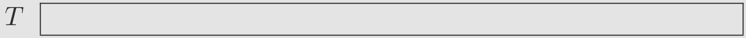
Given a text T and a pattern P find all occurrences of P in T.



What do we want?

The Problem

Given a text T and a pattern P find all occurrences of P in T.



What do we want?

The Problem

Given a text T and a pattern P find all occurrences of P in T.

T

Terminology

- Let T be a string over an alphabet Σ of length n.
- $S_i = T[i, n]$ is a suffix of T starting at index i.
- $P_i = T[1, i]$ is a prefix of T of length i.

What do we want?

The Problem

Given a text T and a pattern P find all occurrences of P in T.

T

Terminology

- Let T be a string over an alphabet Σ of length n.
- $S_i = T[i, n]$ is a suffix of T starting at index i.
- $P_i = T[1, i]$ is a prefix of T of length i.

What do we want?

The Problem

Given a text T and a pattern P find all occurrences of P in T.

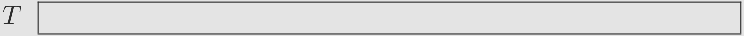
Terminology

- Let T be a string over an alphabet Σ of length n.
- $S_i = T[i, n]$ is a suffix of T starting at index i.
- $P_i = T[1, i]$ is a prefix of T of length i.

What do we want?

The Problem

Given a text T and a pattern P find all occurrences of P in T.



Terminology

- Let T be a string over an alphabet Σ of length n.
- $S_i = T[i, n]$ is a suffix of T starting at index i.
- $P_i = T[1, i]$ is a prefix of T of length i.

What do we want?

The Problem

Given a text T and a pattern P find all occurrences of P in T.

T

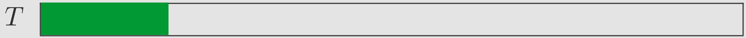
Terminology

- Let T be a string over an alphabet Σ of length n.
- $S_i = T[i, n]$ is a suffix of T starting at index i.
- $P_i = T[1, i]$ is a prefix of T of length i.

What do we want?

The Problem

Given a text T and a pattern P find all occurrences of P in T.



Terminology

- Let T be a string over an alphabet Σ of length n.
- $S_i = T[i, n]$ is a suffix of T starting at index i.
- $P_i = T[1, i]$ is a prefix of T of length i.

What do we want?

The Problem

Given a text T and a pattern P find all occurrences of P in T.

T

Terminology

- Let T be a string over an alphabet Σ of length n.
- $S_i = T[i, n]$ is a suffix of T starting at index i.
- $P_i = T[1, i]$ is a prefix of T of length i.

What do we want?

The Problem

Given a text T and a pattern P find all occurrences of P in T.

T

Terminology

- Let T be a string over an alphabet Σ of length n.
- $S_i = T[i, n]$ is a suffix of T starting at index i.
- $P_i = T[1, i]$ is a prefix of T of length i.

How Fast?

- Without preprocessing: $\Theta(|T| + |P|)$ [Knuth, Morris, Pratt, 1977]
- With Preprocessing: $\Theta(|P|)$ + Preprocessing [Weiner, 1973]

What do we want?

The Problem

Given a text T and a pattern P find all occurrences of P in T.

T

Terminology

- Let T be a string over an alphabet Σ of length n.
- $S_i = T[i, n]$ is a suffix of T starting at index i.
- $P_i = T[1, i]$ is a prefix of T of length i.

How Fast?

- Without preprocessing: $\Theta(|T| + |P|)$ [Knuth, Morris, Pratt, 1977]
- With Preprocessing: $\Theta(|P|)$ + Preprocessing [Weiner, 1973]

Build an Index

Indices and Texts

Structured Text vs. Unstructured Text

- Linguistic text
- Source Code
- DNA sequences
- Executables

Indices and Texts

Structured Text vs. Unstructured Text

- Linguistic text
- Source Code
- DNA sequences
- Executables

Word-Based vs. Full-Text Indices

	Unstructured	Structured
Word-Based		
Full-Text		

Indices and Texts

Structured Text vs. Unstructured Text

- Linguistic text
- Source Code
- DNA sequences
- Executables

Word-Based vs. Full-Text Indices

	Unstructured	Structured
Word-Based		
Full-Text		

Indices and Texts

Structured Text vs. Unstructured Text

- Linguistic text
- Source Code
- DNA sequences
- Executables

Word-Based vs. Full-Text Indices

	Unstructured	Structured
Word-Based		
Full-Text		

A Word-Based Index

Index of "Algorithms on String" (Crochemore, 2007)

A Word-Based Index

Index of "Algorithms on String" (Crochemore, 2007)

No Match = No Reference

- We want every occurrence of $P = \text{tree}$.

A Word-Based Index

Index of "Algorithms on String" (Crochemore, 2007)

No Match = No Reference

- We want every occurrence of $P = \text{tree}$.

A Word-Based Index

Index of "Algorithms on String" (Crochemore, 2007)

No Match = No Reference

- We want every occurrence of $P = \mathrm{tree}$.

A Word-Based Index

Index of "Algorithms on String" (Crochemore, 2007)

No Match = No Reference

- We want every occurrence of $P = \mathrm{tree}$.
- **Build an index containing all words?**

A Word-Based Index

Index of "Algorithms on String" (Crochemore, 2007)

No Match = No Reference

- We want every occurrence of $P = \mathrm{tree}$.
- **Build an index containing all ~~words~~?**

A Word-Based Index

Index of "Algorithms on String" (Crochemore, 2007)

No Match = No Reference

- We want every occurrence of $P = \mathrm{tree}$.
- **Build an index containing all ~~words~~ suffixes?**

Unstructured Text

DNA as Text

A C G T T G T T C T G A A G A A A A T T T A T G
A A C G T T A T A C A A C G G T C G A C C A T A
G G A T T A C A C G G C A G A G G T G G T T G T
C T A A G G C G T T A C C C C A A T C G T T A T
A A C G C T A C A T G C A C G A A A C C A A G T
G G A C A T A G C C T T G T A G C G T T A A T G
G A G C C T T C A C C G G C A T T C T G T T T A
A T C A T A T G C A G A T C C G T T A T G C G C
C T C T G T A A T A G G G A C T A A A A A A G T

Unstructured Text

DNA as Text

A C G T T G T T C T G A A G A A A T T T A T G

A A C G T T A T A C A A C G G T C G A C C A T A

G G A T T A C A C G G C A G A G G T G G T T G T

C T A A G G C G T T A C C C C A A T C G T T A T

A A C G C T A C A T G C A C G A A A C C A A G T

G G A C A T A G C C T T G T A G C G T T A A T G

G A G C C T T C A C C G G C A T T C T G T T T A

A T C A T A T G C A G A T C C G T T A T G C G C

C T C T G T A A T A G G G A C T A A A A A A G T

$P_0 = \text{CGTTA}$

Unstructured Text

DNA as Text

A C G T T G T T C T G A A G A A A A T T T A T G
A A C G T T A T A C A A C G G T C G A C C A T A
G G A T T A C A C G G C A G A G G T G G T T G T
C T A A G G C G T T A C C C C A A T C G T T A T
A A C G C T A C A T G C A C G A A A C C A A G T
G G A C A T A G C C T T G T A G C G T T A A T G
G A G C C T T C A C C G G C A T T C T G T T T A
A T C A T A T G C A G A T C C G T T A T G C G C
C T C T G T A A T A G G G A C T A A A A A A G T

$P_0 =$ CGTTA

Unstructured Text

DNA as Text

A C G T T G T T C T G A A G A A A A T T T A T G
A A C G T T A T A C A A C G G T C G A C C A T A
G G A T T A C A C G G C A G A G G T G G T T G T
C T A A G G C G T T A C C C C A A T C G T T A T
A A C G C T A C A T G C A C G A A A C C A A G T
G G A C A T A G C C T T G T A G C G T T A A T G
G A G C C T T C A C C G G C A T T C T G T T T A
A T C A T A T G C A G A T C C G T T A T G C G C
C T C T G T A A T A G G G A C T A A A A A A G T

$$P_0 = \text{CGTTA}$$
$$P_1 = \dots$$

Combining Index- and Text-Types

Word-Based vs. Full-Text Indices

	Unstructured	Structured
Word-Based	✗	
Full-Text		

Combining Index- and Text-Types

Word-Based vs. Full-Text Indices

	Unstructured	Structured
Word-Based	✗	✓
Full-Text		

Combining Index- and Text-Types

Word-Based vs. Full-Text Indices

	Unstructured	Structured
Word-Based	✗	✓
Full-Text	✓	✓

Combining Index- and Text-Types

Word-Based vs. Full-Text Indices

	Unstructured	Structured
Word-Based	✗	✓
Full-Text	✓	✓

We want full-text indices!

How to Represent a Fulltext Index

How to Represent a Fulltext Index

A AAAAAAGT, 208
AAAAAGT, 209
AAAAGT, 210
...

C CAACGGTCGACCATAGGATTAC[...]AAAAAAGT, 33
CAAGTGGACATAGCCTTGTAGC[...]AAAAAAGT, 115
CAATCGTTATAACGCTACATGC[...]AAAAAAGT, 86
...

G GAAAATTTATGAACGTTATACA[...]AAAAAAGT, 13
GAAACCAAGTGGACATAGCCTT[...]AAAAAAGT, 110
GAACGTTATACAACGGTCGACC[...]AAAAAAGT, 23
...

T T, 215
TAAAAAAGT, 207
TAACGCTACATGCACGAAACCA[...]AAAAAAGT, 95
...

How to Represent a Fulltext Index

A AAAAAAGT, 208
AAAAAGT, 209
AAAAGT, 210
...

Space $O\left(|T|^2\right)$

C CAACGGTCGACCATAGGATTAC[...]AAAAAAGT, 33
CAAGTGGACATAGCCTTGTAGC[...]AAAAAAGT, 115
CAATCGTTATAACGCTACATGC[...]AAAAAAGT, 86
...

G GAAAATTTATGAACGTTATACA[...]AAAAAAGT, 13
GAAACCAAGTGGACATAGCCTT[...]AAAAAAGT, 110
GAACGTTATACAACGGTCGACC[...]AAAAAAGT, 23
...

T T, 215
TAAAAAAGT, 207
TAACGCTACATGCACGAAACCA[...]AAAAAAGT, 95
...

How to Represent a Fulltext Index

Space $O\left(|T|^2\right)$

A AAAAAAGT, 208
AAAAAGT, 209
AAAAGT, 210

C CAA GTCGACCATAGGATTAC[. AAAAAGT, 33
CAAGTG CATAGCCTTGTAC . .]AAAAAAGT, 115
CAATCGTTA ACGCTAC GC[. . .]AAAAAAGT, 86
. . .

G GAAAATTTATC CGT TACA[. . .]AAAAAAGT, 13
GAAACCAA GGACATAGC T[. . .]AAAAAAGT, 110
GAACG ATACAACGGTCGACC[ ]AAAAAAGT, 23
. .

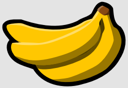
T, 215
TAAAAAAGT, 207
TAACGCTACATGCACGAAACCA[. . .]AAAAAAGT, 9
. . .

Ordered Suffixes

The Typical Example

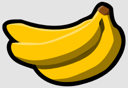
Ordered Suffixes

The Typical Example



Ordered Suffixes

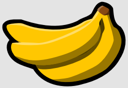
The Typical Example



$$T = \overset{0}{\text{B}}\overset{1}{\text{A}}\overset{2}{\text{N}}\overset{3}{\text{A}}\overset{4}{\text{N}}\overset{5}{\text{A}}\overset{6}{\text{\$}}$$

Ordered Suffixes

The Typical Example



$$T = \overset{0}{\text{B}}\overset{1}{\text{A}}\overset{2}{\text{N}}\overset{3}{\text{A}}\overset{4}{\text{N}}\overset{5}{\text{A}}\overset{6}{\text{\$}}$$

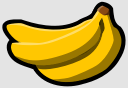
Suffixarray (SA)

i	0	1	2	3	4	5	6
	$	A	A	A	B	N	N
		$	N	N	A	A	A
			A	A	N	$	N
			$	N	A		A
				A	N		$
				$	A		
					$		

- Array containing the suffixes of a string T in lexicographical order.

Ordered Suffixes

The Typical Example



$$T = \overset{0}{\text{B}}\overset{1}{\text{A}}\overset{2}{\text{N}}\overset{3}{\text{A}}\overset{4}{\text{N}}\overset{5}{\text{A}}\overset{6}{\$}$$

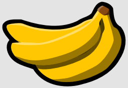
Suffixarray (SA)

i	0	1	2	3	4	5	6
	$	A	A	A	B	N	N
		$	N	N	A	A	A
			A	A	N	$	N
			$	N	A		A
				A	N		$
				$	A		
					$		

- Array containing the suffixes of a string T in lexicographical order.
- Suffixes are represented by their index.

Ordered Suffixes

The Typical Example



$$T = \overset{0}{\mathrm{B}}\overset{1}{\mathrm{A}}\overset{2}{\mathrm{N}}\overset{3}{\mathrm{A}}\overset{4}{\mathrm{N}}\overset{5}{\mathrm{A}}\overset{6}{\$}$$

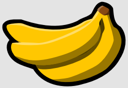
Suffixarray (SA)

i	0	1	2	3	4	5	6
$\mathrm{SA}[i]$	6	5	3	1	0	4	2
	$	A	A	A	B	N	N
		$	N	N	A	A	A
			A	A	N	$	N
			$	N	A		A
				A	N		$
				$	A		
					$		

- Array containing the suffixes of a string T in lexicographical order.
- Suffixes are represented by their index.

Ordered Suffixes

The Typical Example



$$T = \overset{0}{\text{B}}\overset{1}{\text{A}}\overset{2}{\text{N}}\overset{3}{\text{A}}\overset{4}{\text{N}}\overset{5}{\text{A}}\overset{6}{\text{\$}}$$

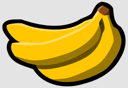
Suffixarray (SA)

i	0	1	2	3	4	5	6
$\mathrm{SA}[i]$	6	5	3	1	0	4	2
	$	A	A	A	B	N	N
		$	N	N	A	A	A
			A	A	N	$	N
			$	N	A		A
				A	N		$
				$	A		
					$		

- Array containing the suffixes of a string T in lexicographical order.
- Suffixes are represented by their index.
- Answer queries with binary search.

Ordered Suffixes

The Typical Example



$$T = \overset{0}{\text{B}}\overset{1}{\text{A}}\overset{2}{\text{N}}\overset{3}{\text{A}}\overset{4}{\text{N}}\overset{5}{\text{A}}\overset{6}{\text{\$}}$$

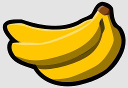
Suffixarray (SA)

i	0	1	2	3	4	5	6
SA[i]	6	5	3	1	0	4	2
	$	A	A	A	B	N	N
		$	N	N	A	A	A
			A	A	N	$	N
			$	N	A		A
				A	N		$
				$	A		
					$		

- Array containing the suffixes of a string T in lexicographical order.
- Suffixes are represented by their index.
- Answer queries with binary search.
- $P =$ BAN

Ordered Suffixes

The Typical Example



$$T = \overset{0}{\text{B}}\overset{1}{\text{A}}\overset{2}{\text{N}}\overset{3}{\text{A}}\overset{4}{\text{N}}\overset{5}{\text{A}}\overset{6}{\$}$$

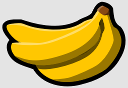
Suffixarray (SA)

i	0	1	2	↓ 3	4	5	6
SA[i]	6	5	3	1	0	4	2
	$	A	A	A	B	N	N
		$	N	N	A	A	A
			A	A	N	$	N
			$	N	A		A
				A	N		$
				$	A		
					$		

- Array containing the suffixes of a string T in lexicographical order.
- Suffixes are represented by their index.
- Answer queries with binary search.
- $P =$ BAN

Ordered Suffixes

The Typical Example



$$T = \overset{0}{\mathrm{B}}\overset{1}{\mathrm{A}}\overset{2}{\mathrm{N}}\overset{3}{\mathrm{A}}\overset{4}{\mathrm{N}}\overset{5}{\mathrm{A}}\overset{6}{\$}$$

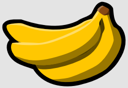
Suffixarray (SA)

i	0	1	2	3	4	5 ↓	6
SA[i]	6	5	3	1	0	4	2
	$	A	A	A	B	N	N
		$	N	N	A	A	A
			A	A	N	$	N
			$	N	A		A
				A	N		$
				$	A		
					$		

- Array containing the suffixes of a string T in lexicographical order.
- Suffixes are represented by their index.
- Answer queries with binary search.
- $P =$ BAN

Ordered Suffixes

The Typical Example



$$T = \overset{0}{\text{B}}\overset{1}{\text{A}}\overset{2}{\text{N}}\overset{3}{\text{A}}\overset{4}{\text{N}}\overset{5}{\text{A}}\overset{6}{\$}$$

Suffixarray (SA)

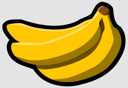
i	0	1	2	3	4	5	6
SA[i]	6	5	3	1	0	4	2
	$	A	A	A	B	N	N
		$	N	N	A	A	A
			A	A	N	$	N
			$	N	A		A
				A	N		$
				$	A		
					$		

- Array containing the suffixes of a string T in lexicographical order.
- Suffixes are represented by their index.
- Answer queries with binary search.
- $P =$ BAN

Common Prefixes

Data Structures

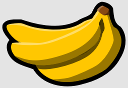
The Typical Example



$$T = \overset{0}{\mathrm{B}}\overset{1}{\mathrm{A}}\overset{2}{\mathrm{N}}\overset{3}{\mathrm{A}}\overset{4}{\mathrm{N}}\overset{5}{\mathrm{A}}\overset{6}{\$}$$

Common Prefixes

The Typical Example



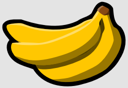
$$T = \overset{0\,1\,2\,3\,4\,5\,6}{\text{BANANA\$}}$$

Longest-Common-Prefix Array (LCP)

i	0	1	2	3	4	5	6
$\text{SA}[i]$	6	5	3	1	0	4	2
$\text{LCP}[i]$							
	$	A	A	A	B	N	N
		$	N	N	A	A	A
			A	A	N	$	N
			$	N	A		A
				A	N		$
				$	A		
					$		

Common Prefixes

The Typical Example



$$T = \overset{0}{\mathrm{B}}\overset{1}{\mathrm{A}}\overset{2}{\mathrm{N}}\overset{3}{\mathrm{A}}\overset{4}{\mathrm{N}}\overset{5}{\mathrm{A}}\overset{6}{\$}$$

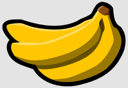
Longest-Common-Prefix Array (LCP)

i	0	1	2	3	4	5	6
$\mathrm{SA}[i]$	6	5	3	1	0	4	2
$\mathrm{LCP}[i]$							
	$	A	A	A	B	N	N
		$	N	N	A	A	A
			A	A	N	$	N
			$	N	A		A
				A	N		$
				$	A		
					$		

- Array containing the size of the longest common prefix of $S_{\mathrm{SA}[i-1]}$ and $S_{\mathrm{SA}[i]}$ at position i.

Common Prefixes

The Typical Example



$$T = \overset{0}{\mathrm{B}}\overset{1}{\mathrm{A}}\overset{2}{\mathrm{N}}\overset{3}{\mathrm{A}}\overset{4}{\mathrm{N}}\overset{5}{\mathrm{A}}\overset{6}{\$}$$

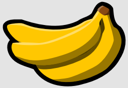
Longest-Common-Prefix Array (LCP)

i	0	1	2	3	4	5	6
$\mathrm{SA}[i]$	6	5	3	1	0	4	2
$\mathrm{LCP}[i]$							
	$	A	A	A	B	N	N
		$	N	N	A	A	A
			A	A	N	$	N
			$	N	A		A
				A	N		$
				$	A		
					$		

- Array containing the size of the longest common prefix of $S_{\mathrm{SA}[i-1]}$ and $S_{\mathrm{SA}[i]}$ at position i.

Common Prefixes

The Typical Example



$$T = \overset{0\,1\,2\,3\,4\,5\,6}{\text{BANANA\$}}$$

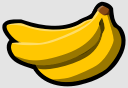
Longest-Common-Prefix Array (LCP)

i	0	1	2	3	4	5	6
$\mathrm{SA}[i]$	6	5	3	1	0	4	2
$\mathrm{LCP}[i]$							2
	$	A	A	A	B	N	N
		$	N	N	A	A	A
			A	A	N	$	N
			$	N	A		A
				A	N		$
				$	A		
					$		

- Array containing the size of the longest common prefix of $S_{\mathrm{SA}[i-1]}$ and $S_{\mathrm{SA}[i]}$ at position i.

Common Prefixes

The Typical Example



$$T = \overset{0}{\text{B}}\overset{1}{\text{A}}\overset{2}{\text{N}}\overset{3}{\text{A}}\overset{4}{\text{N}}\overset{5}{\text{A}}\overset{6}{\text{\$}}$$

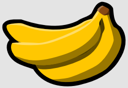
Longest-Common-Prefix Array (LCP)

i	0	1	2	3	4	5	6
$\mathrm{SA}[i]$	6	5	3	1	0	4	2
$\mathrm{LCP}[i]$							2
	$	A	A	A	B	N	N
		$	N	N	A	A	A
			A	A	N	$	N
			$	N	A		A
				A	N		$
				$	A		
					$		

- Array containing the size of the longest common prefix of $S_{\mathrm{SA}[i-1]}$ and $S_{\mathrm{SA}[i]}$ at position i.

Common Prefixes

The Typical Example



$$T = \overset{0}{\text{B}}\overset{1}{\text{A}}\overset{2}{\text{N}}\overset{3}{\text{A}}\overset{4}{\text{N}}\overset{5}{\text{A}}\overset{6}{\$}$$

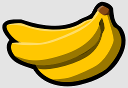
Longest-Common-Prefix Array (LCP)

i	0	1	2	3	4	5	6
$\mathrm{SA}[i]$	6	5	3	1	0	4	2
$\mathrm{LCP}[i]$							2
	$	A	A	A	B	N	N
		$	N	N	A	A	A
			A	A	N	$	N
			$	N	A		A
				A	N		$
				$	A		
					$		

- Array containing the size of the longest common prefix of $S_{\mathrm{SA}[i-1]}$ and $S_{\mathrm{SA}[i]}$ at position i.

Common Prefixes

The Typical Example



$$T = \overset{0}{\text{B}}\overset{1}{\text{A}}\overset{2}{\text{N}}\overset{3}{\text{A}}\overset{4}{\text{N}}\overset{5}{\text{A}}\overset{6}{\$}$$

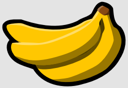
Longest-Common-Prefix Array (LCP)

i	0	1	2	3	4	5	6
$\text{SA}[i]$	6	5	3	1	0	4	2
$\text{LCP}[i]$						0	2
	$	A	A	A	B	N	N
		$	N	N	A	A	A
			A	A	N	$	N
			$	N	A		A
				A	N		$
				$	A		
					$		

- Array containing the size of the longest common prefix of $S_{\text{SA}[i-1]}$ and $S_{\text{SA}[i]}$ at position i.

Common Prefixes

The Typical Example



$$T = \overset{0}{\text{B}}\overset{1}{\text{A}}\overset{2}{\text{N}}\overset{3}{\text{A}}\overset{4}{\text{N}}\overset{5}{\text{A}}\overset{6}{\$}$$

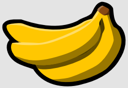
Longest-Common-Prefix Array (LCP)

i	0	1	2	3	4	5	6
$\text{SA}[i]$	6	5	3	1	0	4	2
$\text{LCP}[i]$						0	2
	$	A	A	A	B	N	N
		$	N	N	A	A	A
			A	A	N	$	N
			$	N	A		A
				A	N		$
				$	A		
					$		

- Array containing the size of the longest common prefix of $S_{\text{SA}[i-1]}$ and $S_{\text{SA}[i]}$ at position i.

Common Prefixes

The Typical Example



$$T = \overset{0\,1\,2\,3\,4\,5\,6}{\text{BANANA\$}}$$

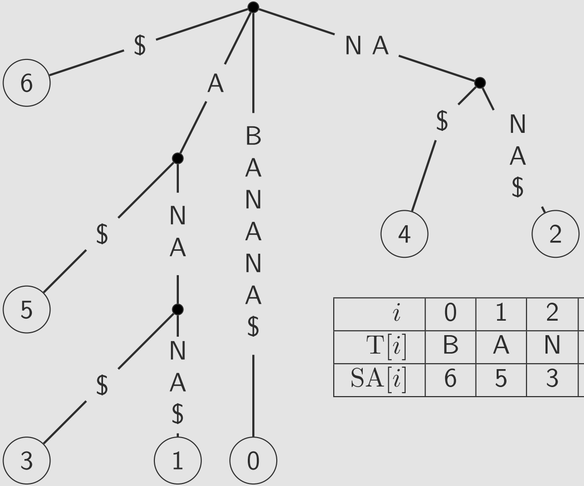
Longest-Common-Prefix Array (LCP)

i	0	1	2	3	4	5	6
SA$[i]$	6	5	3	1	0	4	2
LCP$[i]$	-1	0	1	3	0	0	2
	$	A	A	A	B	N	N
		$	N	N	A	A	A
			A	A	N	$	N
			$	N	A		A
				A	N		$
				$	A		
					$		

- Array containing the size of the longest common prefix of $S_{\mathrm{SA}[i-1]}$ and $S_{\mathrm{SA}[i]}$ at position i.

Suffix Trees

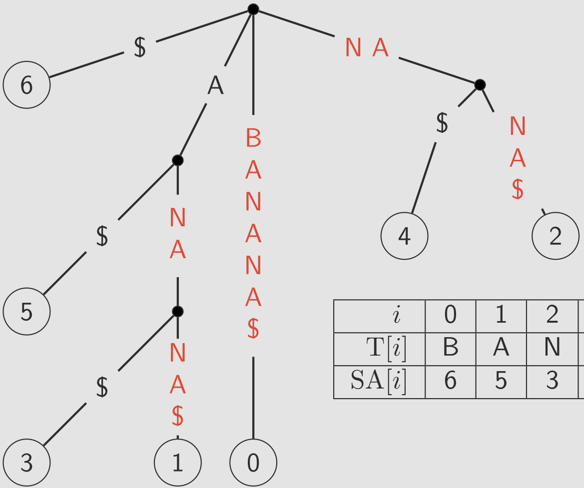
A Compressed Trie



i	0	1	2	3	4	5	6
T[i]	B	A	N	A	N	A	$
SA[i]	6	5	3	1	0	4	2

Suffix Trees

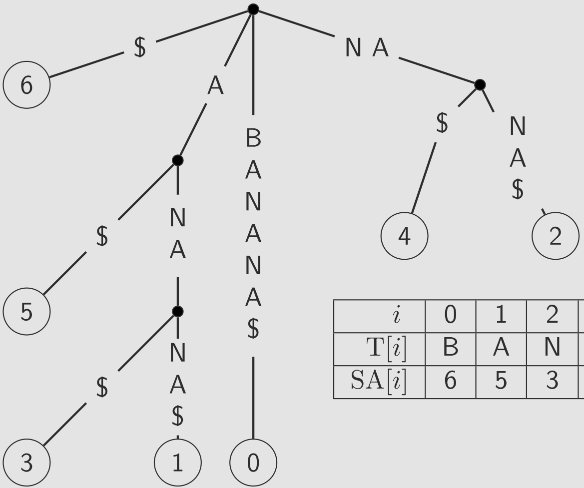
A Compressed Trie



i	0	1	2	3	4	5	6
$T[i]$	B	A	N	A	N	A	$
$SA[i]$	6	5	3	1	0	4	2

Suffix Trees

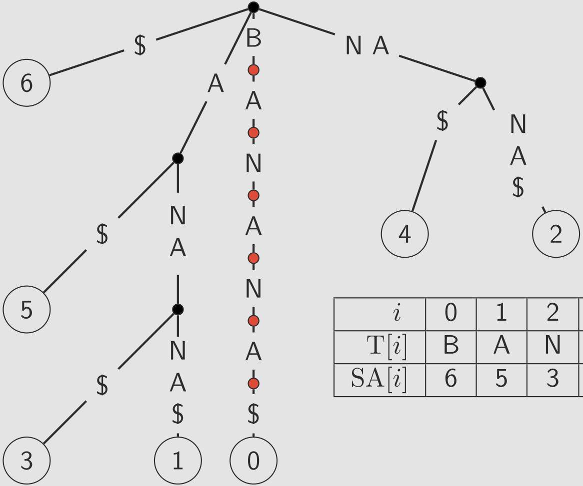
A Compressed Trie



i	0	1	2	3	4	5	6
T[i]	B	A	N	A	N	A	$
SA[i]	6	5	3	1	0	4	2

Suffix Trees

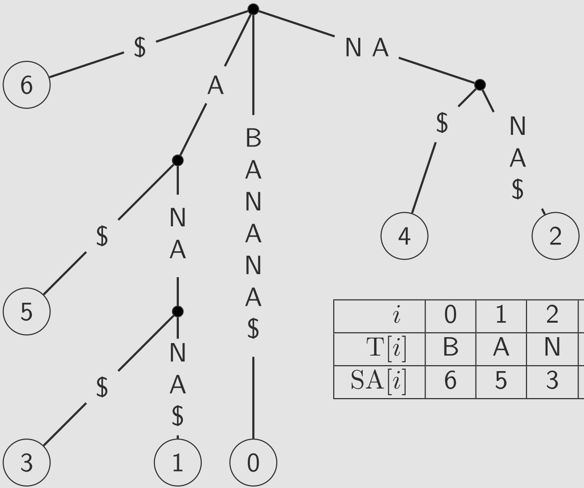
A Compressed Trie



i	0	1	2	3	4	5	6
$T[i]$	B	A	N	A	N	A	$
$SA[i]$	6	5	3	1	0	4	2

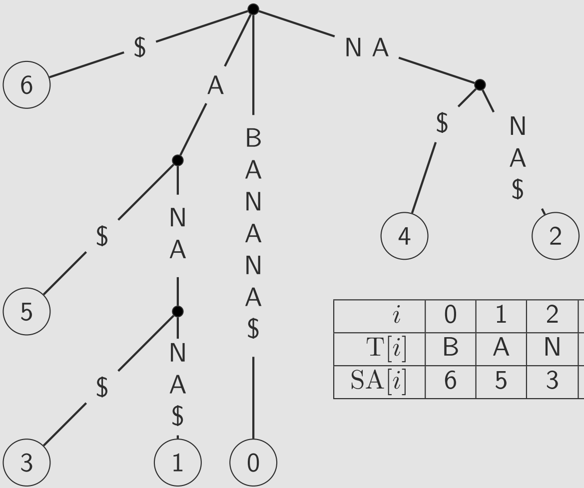
Suffix Trees

A Compressed Trie



i	0	1	2	3	4	5	6
T[i]	B	A	N	A	N	A	$
SA[i]	6	5	3	1	0	4	2

A Compressed Trie

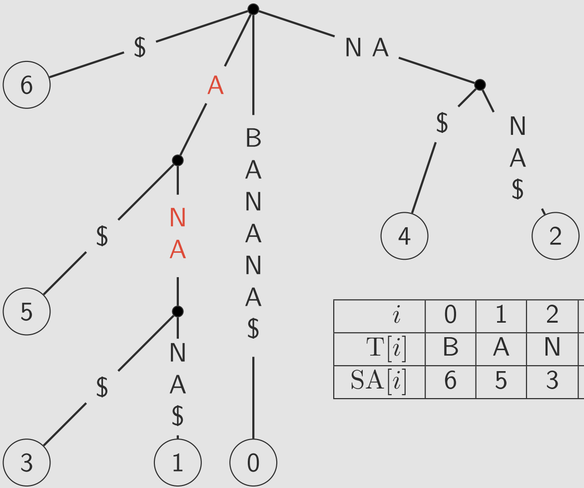


$Q = \text{ANA}$

i	0	1	2	3	4	5	6
$T[i]$	B	A	N	A	N	A	$
$SA[i]$	6	5	3	1	0	4	2

Suffix Trees

A Compressed Trie

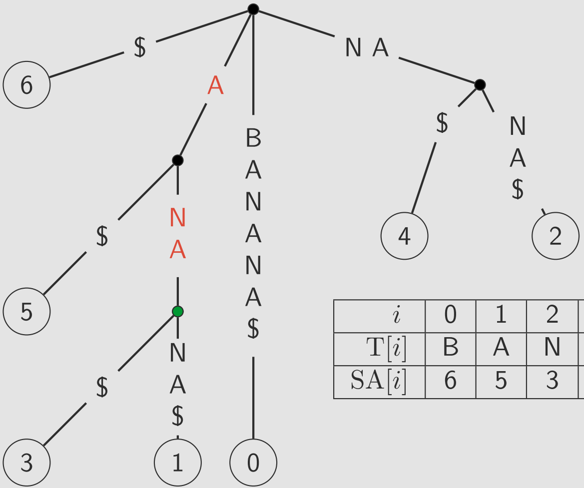


$Q = \text{ANA}$

i	0	1	2	3	4	5	6
T[i]	B	A	N	A	N	A	$
SA[i]	6	5	3	1	0	4	2

Suffix Trees

A Compressed Trie



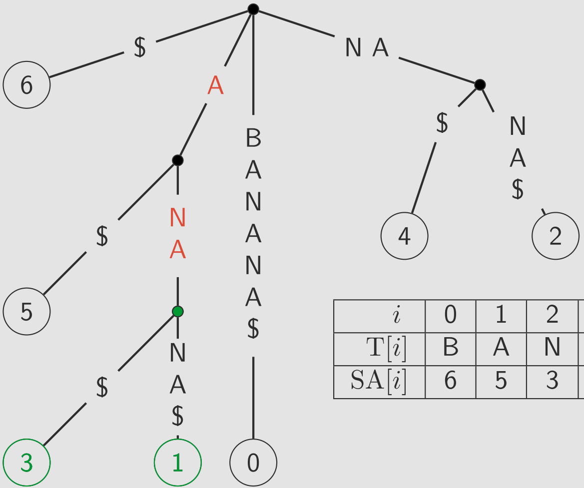
$Q = \mathrm{ANA}$

i	0	1	2	3	4	5	6
$\mathrm{T}[i]$	B	A	N	A	N	A	$
$\mathrm{SA}[i]$	6	5	3	1	0	4	2

Suffix Trees

Data Structures

A Compressed Trie

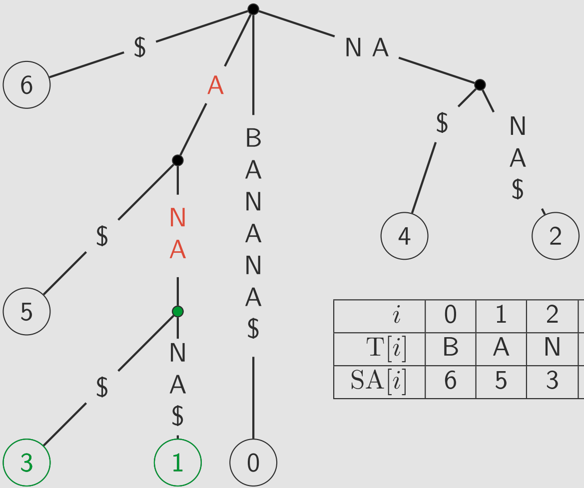


$Q = \mathrm{ANA}$

i	0	1	2	3	4	5	6
$\mathrm{T}[i]$	B	A	N	A	N	A	$
$\mathrm{SA}[i]$	6	5	3	1	0	4	2

Suffix Trees

A Compressed Trie

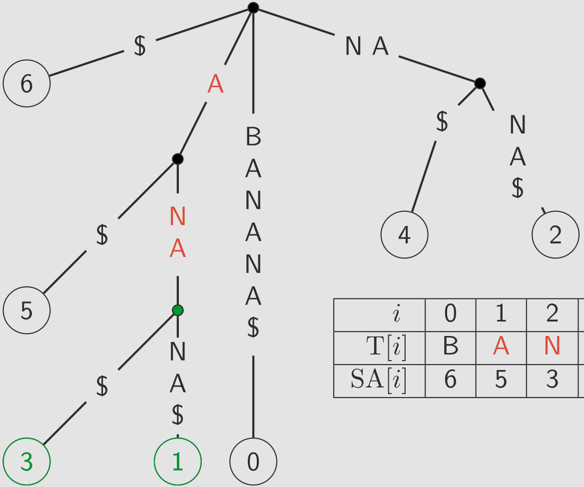


$Q = \mathrm{ANA}$

i	0	1	2	3	4	5	6
$\mathrm{T}[i]$	B	A	N	A	N	A	$
$\mathrm{SA}[i]$	6	5	3	1	0	4	2

Suffix Trees

A Compressed Trie

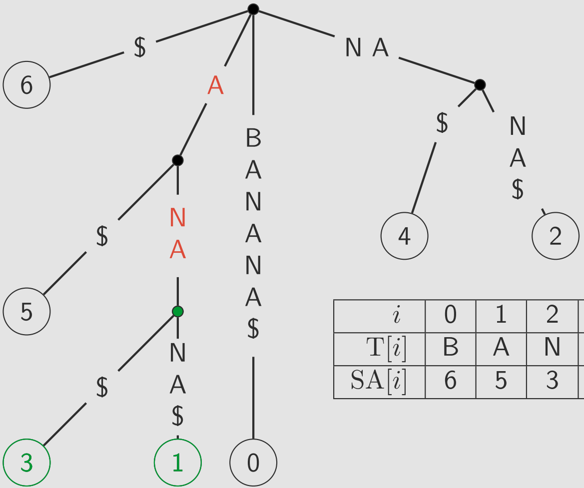


$Q = \mathrm{ANA}$

i	0	1	2	3	4	5	6
$\mathrm{T}[i]$	B	A	N	A	N	A	$
$\mathrm{SA}[i]$	6	5	3	1	0	4	2

Suffix Trees

A Compressed Trie

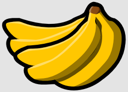


$Q = \text{ANA}$

i	0	1	2	3	4	5	6
$T[i]$	B	A	N	A	N	A	$
$SA[i]$	6	5	3	1	0	4	2

String B-Trees

The Extended Example



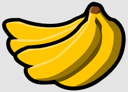
$$T = \overset{0}{\text{B}}\overset{1}{\text{A}}\overset{2}{\text{A}}\overset{3}{\text{N}}\overset{4}{\text{N}}\overset{5}{\text{A}}\overset{6}{\text{A}}\overset{7}{\text{N}}\overset{8}{\text{N}}\overset{9}{\text{A}}\overset{10}{\text{A}}\overset{11}{\$}$$

B-Tree on top of a Suffixarray

$\mathrm{SA}[i]$												
	11	10	9	5	1	6	2	0	8	4	7	3

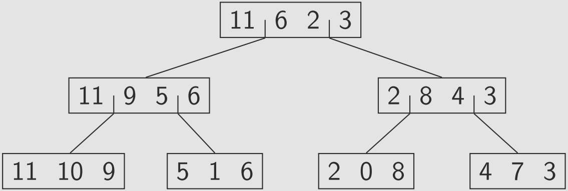
String B-Trees

The Extended Example



$$T = \overset{0}{\mathrm{B}}\overset{1}{\mathrm{A}}\overset{2}{\mathrm{A}}\overset{3}{\mathrm{N}}\overset{4}{\mathrm{N}}\overset{5}{\mathrm{A}}\overset{6}{\mathrm{A}}\overset{7}{\mathrm{N}}\overset{8}{\mathrm{N}}\overset{9}{\mathrm{A}}\overset{10}{\mathrm{A}}\overset{11}{\$}$$

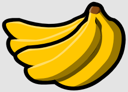
B-Tree on top of a Suffixarray



SA[i]	11	10	9	5	1	6	2	0	8	4	7	3

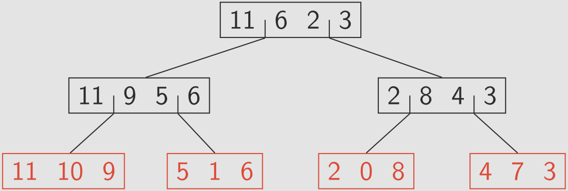
String B-Trees

The Extended Example



$$T = \overset{0}{\text{B}}\overset{1}{\text{A}}\overset{2}{\text{A}}\overset{3}{\text{N}}\overset{4}{\text{N}}\overset{5}{\text{A}}\overset{6}{\text{A}}\overset{7}{\text{N}}\overset{8}{\text{N}}\overset{9}{\text{A}}\overset{10}{\text{A}}\overset{11}{\$}$$

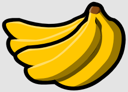
B-Tree on top of a Suffixarray



SA[i]											
11	10	9	5	1	6	2	0	8	4	7	3

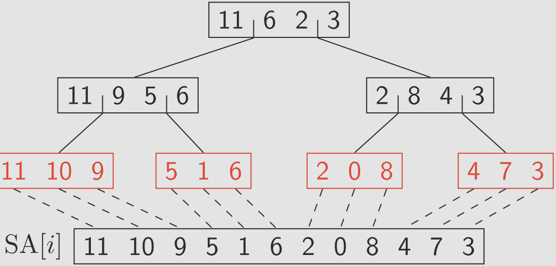
String B-Trees

The Extended Example



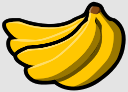
$$T = \overset{0}{\text{B}}\overset{1}{\text{A}}\overset{2}{\text{A}}\overset{3}{\text{N}}\overset{4}{\text{N}}\overset{5}{\text{A}}\overset{6}{\text{A}}\overset{7}{\text{N}}\overset{8}{\text{N}}\overset{9}{\text{A}}\overset{10}{\text{A}}\overset{11}{\$}$$

B-Tree on top of a Suffixarray



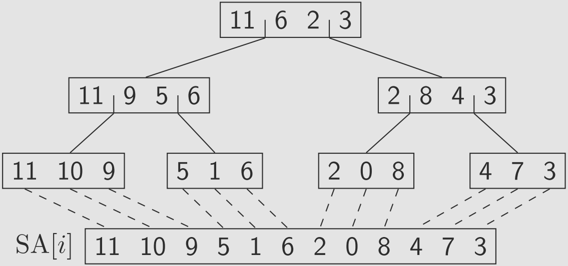
String B-Trees

The Extended Example



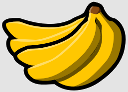
$$T = \overset{0}{\text{B}}\overset{1}{\text{A}}\overset{2}{\text{A}}\overset{3}{\text{N}}\overset{4}{\text{N}}\overset{5}{\text{A}}\overset{6}{\text{A}}\overset{7}{\text{N}}\overset{8}{\text{N}}\overset{9}{\text{A}}\overset{10}{\text{A}}\overset{11}{\$}$$

B-Tree on top of a Suffixarray



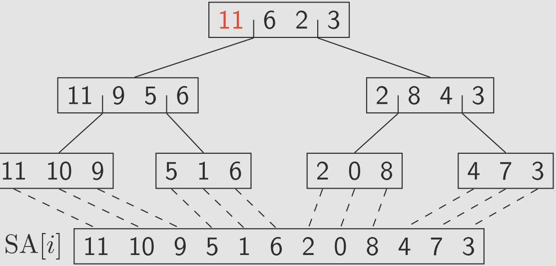
String B-Trees

The Extended Example



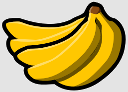
$$T = \overset{0}{\mathrm{B}}\overset{1}{\mathrm{A}}\overset{2}{\mathrm{A}}\overset{3}{\mathrm{N}}\overset{4}{\mathrm{N}}\overset{5}{\mathrm{A}}\overset{6}{\mathrm{A}}\overset{7}{\mathrm{N}}\overset{8}{\mathrm{N}}\overset{9}{\mathrm{A}}\overset{10}{\mathrm{A}}\overset{11}{\$}$$

B-Tree on top of a Suffixarray



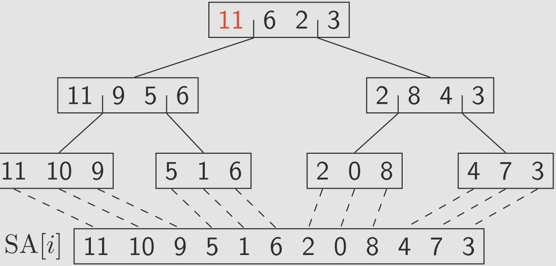
String B-Trees

The Extended Example



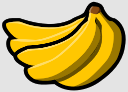
$$T = \overset{0}{\mathrm{B}}\overset{1}{\mathrm{A}}\overset{2}{\mathrm{A}}\overset{3}{\mathrm{N}}\overset{4}{\mathrm{N}}\overset{5}{\mathrm{A}}\overset{6}{\mathrm{A}}\overset{7}{\mathrm{N}}\overset{8}{\mathrm{N}}\overset{9}{\mathrm{A}}\overset{10}{\mathrm{A}}\overset{11}{\$}$$

B-Tree on top of a Suffixarray



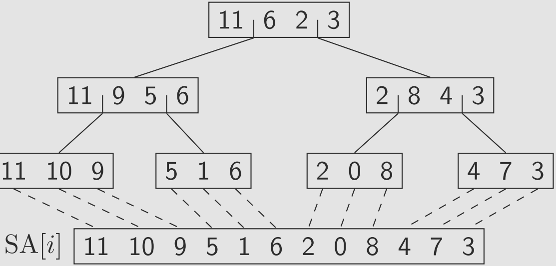
String B-Trees

The Extended Example



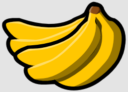
$$T = \overset{0}{\mathrm{B}}\overset{1}{\mathrm{A}}\overset{2}{\mathrm{A}}\overset{3}{\mathrm{N}}\overset{4}{\mathrm{N}}\overset{5}{\mathrm{A}}\overset{6}{\mathrm{A}}\overset{7}{\mathrm{N}}\overset{8}{\mathrm{N}}\overset{9}{\mathrm{A}}\overset{10}{\mathrm{A}}\overset{11}{\$}$$

B-Tree on top of a Suffixarray



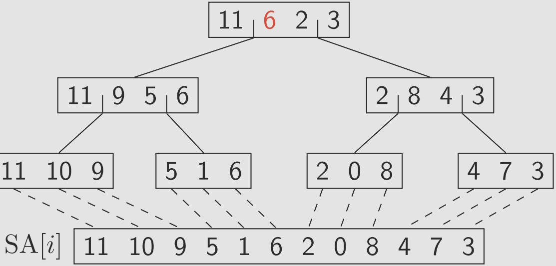
String B-Trees

The Extended Example



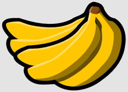
$$T = \overset{0}{\text{B}}\overset{1}{\text{A}}\overset{2}{\text{A}}\overset{3}{\text{N}}\overset{4}{\text{N}}\overset{5}{\text{A}}\overset{6}{\text{A}}\overset{7}{\text{N}}\overset{8}{\text{N}}\overset{9}{\text{A}}\overset{10}{\text{A}}\overset{11}{\$}$$

B-Tree on top of a Suffixarray



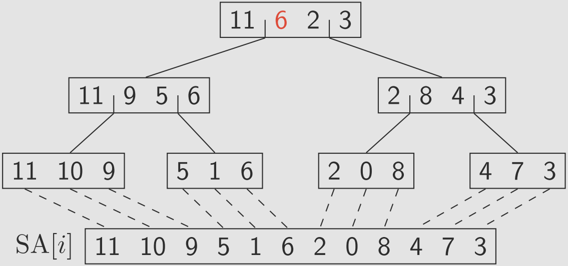
String B-Trees

The Extended Example



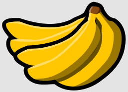
$$T = \overset{0}{\text{B}}\overset{1}{\text{A}}\overset{2}{\text{A}}\overset{3}{\text{N}}\overset{4}{\text{N}}\overset{5}{\text{A}}\overset{6}{\text{A}}\overset{7}{\text{N}}\overset{8}{\text{N}}\overset{9}{\text{A}}\overset{10}{\text{A}}\overset{11}{\$}$$

B-Tree on top of a Suffixarray



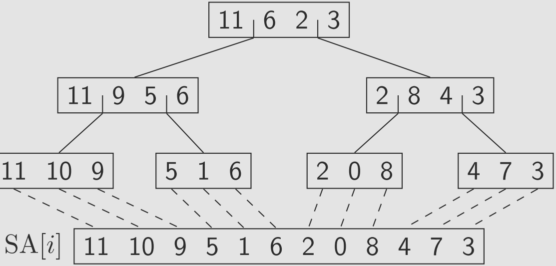
String B-Trees

The Extended Example



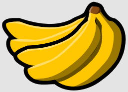
$$T = \overset{0}{\mathrm{B}}\overset{1}{\mathrm{A}}\overset{2}{\mathrm{A}}\overset{3}{\mathrm{N}}\overset{4}{\mathrm{N}}\overset{5}{\mathrm{A}}\overset{6}{\mathrm{A}}\overset{7}{\mathrm{N}}\overset{8}{\mathrm{N}}\overset{9}{\mathrm{A}}\overset{10}{\mathrm{A}}\overset{11}{\$}$$

B-Tree on top of a Suffixarray



String B-Trees

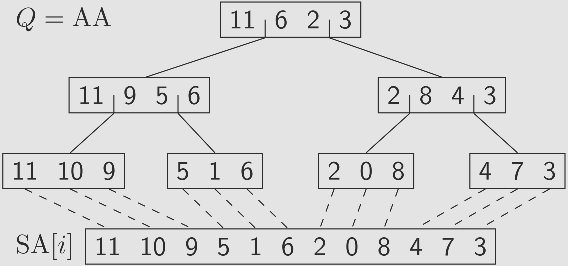
The Extended Example



$$T = \overset{0}{\mathrm{B}}\overset{1}{\mathrm{A}}\overset{2}{\mathrm{A}}\overset{3}{\mathrm{N}}\overset{4}{\mathrm{N}}\overset{5}{\mathrm{A}}\overset{6}{\mathrm{A}}\overset{7}{\mathrm{N}}\overset{8}{\mathrm{N}}\overset{9}{\mathrm{A}}\overset{10}{\mathrm{A}}\overset{11}{\$}$$

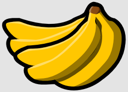
B-Tree on top of a Suffixarray

$Q = \mathrm{AA}$



String B-Trees

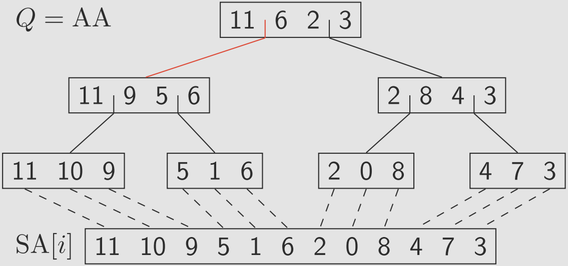
The Extended Example



$$T = \overset{0}{\mathrm{B}}\overset{1}{\mathrm{A}}\overset{2}{\mathrm{A}}\overset{3}{\mathrm{N}}\overset{4}{\mathrm{N}}\overset{5}{\mathrm{A}}\overset{6}{\mathrm{A}}\overset{7}{\mathrm{N}}\overset{8}{\mathrm{N}}\overset{9}{\mathrm{A}}\overset{10}{\mathrm{A}}\overset{11}{\$}$$

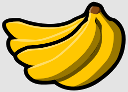
B-Tree on top of a Suffixarray

$Q = \mathrm{AA}$



String B-Trees

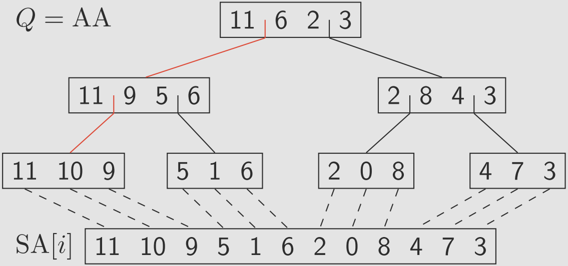
The Extended Example



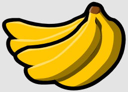
$$T = \overset{0}{\mathrm{B}}\overset{1}{\mathrm{A}}\overset{2}{\mathrm{A}}\overset{3}{\mathrm{N}}\overset{4}{\mathrm{N}}\overset{5}{\mathrm{A}}\overset{6}{\mathrm{A}}\overset{7}{\mathrm{N}}\overset{8}{\mathrm{N}}\overset{9}{\mathrm{A}}\overset{10}{\mathrm{A}}\overset{11}{\$}$$

B-Tree on top of a Suffixarray

$Q = \mathrm{AA}$



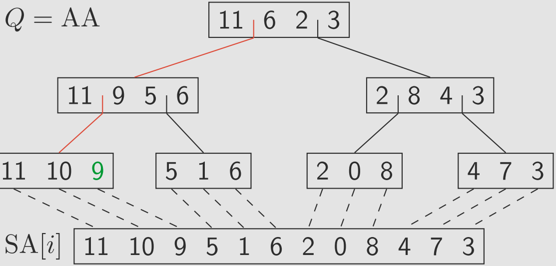
The Extended Example



$$T = \overset{0}{\mathrm{B}}\overset{1}{\mathrm{A}}\overset{2}{\mathrm{A}}\overset{3}{\mathrm{N}}\overset{4}{\mathrm{N}}\overset{5}{\mathrm{A}}\overset{6}{\mathrm{A}}\overset{7}{\mathrm{N}}\overset{8}{\mathrm{N}}\overset{9}{\mathrm{A}}\overset{10}{\mathrm{A}}\overset{11}{\$}$$

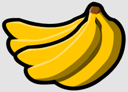
B-Tree on top of a Suffixarray

$Q = \mathrm{AA}$



String B-Trees

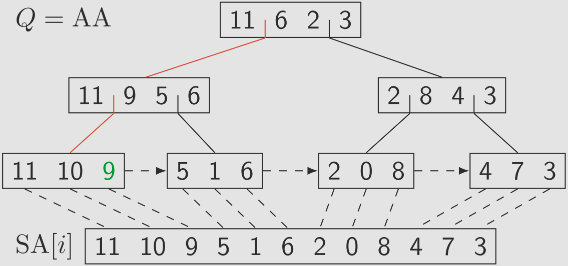
The Extended Example



$$T = \overset{0}{\mathrm{B}}\overset{1}{\mathrm{A}}\overset{2}{\mathrm{A}}\overset{3}{\mathrm{N}}\overset{4}{\mathrm{N}}\overset{5}{\mathrm{A}}\overset{6}{\mathrm{A}}\overset{7}{\mathrm{N}}\overset{8}{\mathrm{N}}\overset{9}{\mathrm{A}}\overset{10}{\mathrm{A}}\overset{11}{\$}$$

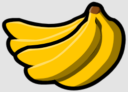
B-Tree on top of a Suffixarray

$Q = \mathrm{AA}$



String B-Trees

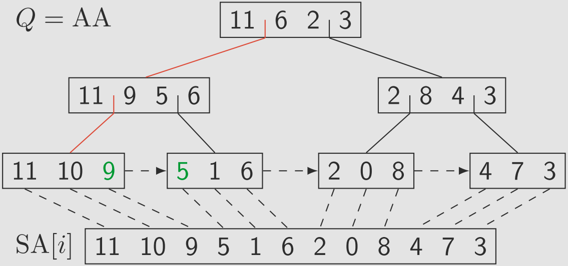
The Extended Example



$$T = \overset{0}{\mathrm{B}}\overset{1}{\mathrm{A}}\overset{2}{\mathrm{A}}\overset{3}{\mathrm{N}}\overset{4}{\mathrm{N}}\overset{5}{\mathrm{A}}\overset{6}{\mathrm{A}}\overset{7}{\mathrm{N}}\overset{8}{\mathrm{N}}\overset{9}{\mathrm{A}}\overset{10}{\mathrm{A}}\overset{11}{\$}$$

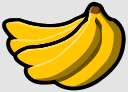
B-Tree on top of a Suffixarray

$Q = \mathrm{AA}$



String B-Trees

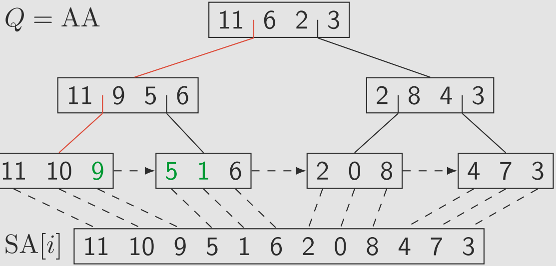
The Extended Example



$$T = \overset{0}{\mathrm{B}}\overset{1}{\mathrm{A}}\overset{2}{\mathrm{A}}\overset{3}{\mathrm{N}}\overset{4}{\mathrm{N}}\overset{5}{\mathrm{A}}\overset{6}{\mathrm{A}}\overset{7}{\mathrm{N}}\overset{8}{\mathrm{N}}\overset{9}{\mathrm{A}}\overset{10}{\mathrm{A}}\overset{11}{\$}$$

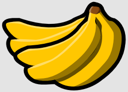
B-Tree on top of a Suffixarray

$Q = \mathrm{AA}$



String B-Trees

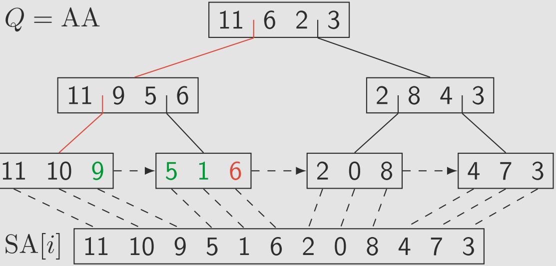
The Extended Example



$$T = \overset{0}{\mathrm{B}}\overset{1}{\mathrm{A}}\overset{2}{\mathrm{A}}\overset{3}{\mathrm{N}}\overset{4}{\mathrm{N}}\overset{5}{\mathrm{A}}\overset{6}{\mathrm{A}}\overset{7}{\mathrm{N}}\overset{8}{\mathrm{N}}\overset{9}{\mathrm{A}}\overset{10}{\mathrm{A}}\overset{11}{\$}$$

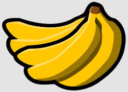
B-Tree on top of a Suffixarray

$Q = \mathrm{AA}$



String B-Trees

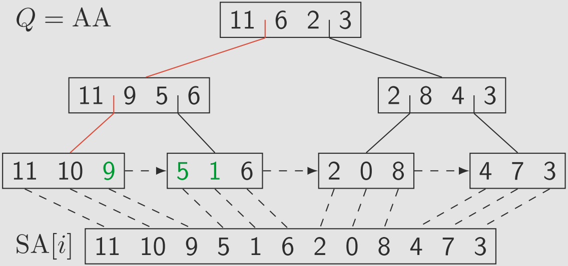
The Extended Example



$$T = \overset{0}{\text{B}}\overset{1}{\text{A}}\overset{2}{\text{A}}\overset{3}{\text{N}}\overset{4}{\text{N}}\overset{5}{\text{A}}\overset{6}{\text{A}}\overset{7}{\text{N}}\overset{8}{\text{N}}\overset{9}{\text{A}}\overset{10}{\text{A}}\overset{11}{\$}$$

B-Tree on top of a Suffixarray

$Q = \text{AA}$



Succinct Data Structures

The Idea

- Using minimum space close to the information-theoretic lower bound.
- Handle large problem sizes in the internal memory.

Succinct Data Structures

The Idea

- Using minimum space close to the information-theoretic lower bound.
- Handle large problem sizes in the internal memory.

Constant Time Operations on Bit Vectors

$$\text{rank}_q(i) = \#q's \text{ up to position } i$$
$$\text{select}_q(i) = \text{position of } i^{\text{th}}\ q$$

0 1 1 1 0 0 1 0 1 0 0 1

Succinct Data Structures

The Idea

- Using minimum space close to the information-theoretic lower bound.
- Handle large problem sizes in the internal memory.

Constant Time Operations on Bit Vectors

$$\mathrm{rank}_q(i) = \#q's \text{ up to position } i \rightarrow \mathrm{rank}_1(4) = 3$$
$$\mathrm{select}_q(i) = \text{position of } i^{\text{th}}\ q$$

0 1 1 1 0 0 1 0 1 0 0 1

Succinct Data Structures

The Idea

- Using minimum space close to the information-theoretic lower bound.
- Handle large problem sizes in the internal memory.

Constant Time Operations on Bit Vectors

$$\text{rank}_q(i) = \#q's \text{ up to position } i \rightarrow \text{rank}_1(4) = 3$$

$$\text{select}_q(i) = \text{position of } i^{\text{th}}\ q$$

0 1 1 1 0 0 1 0 1 0 0 1

Succinct Data Structures

The Idea

- Using minimum space close to the information-theoretic lower bound.
- Handle large problem sizes in the internal memory.

Constant Time Operations on Bit Vectors

$$\text{rank}_q(i) = \#q's \text{ up to position } i \rightarrow \text{rank}_1(4) = 3$$
$$\text{select}_q(i) = \text{position of } i^{\text{th}}\ q$$

0	1	1	1	0	0	1	0	1	0	0	1
	1	2	3								

Succinct Data Structures

The Idea

- Using minimum space close to the information-theoretic lower bound.
- Handle large problem sizes in the internal memory.

Constant Time Operations on Bit Vectors

$$\text{rank}_q(i) = \#q's \text{ up to position } i \rightarrow \text{rank}_1(4) = 3$$
$$\text{select}_q(i) = \text{position of } i^{\text{th}}\ q$$

0 1 1 1 0 0 1 0 1 0 0 1

Succinct Data Structures

The Idea

- Using minimum space close to the information-theoretic lower bound.
- Handle large problem sizes in the internal memory.

Constant Time Operations on Bit Vectors

$$\mathsf{rank}_q(i) = \#q's \text{ up to position } i \rightarrow \mathsf{rank}_1(4) = 3$$

$$\mathsf{select}_q(i) = \text{position of } i^{\text{th}}\ q \rightarrow \mathsf{select}_0(3) = 5$$

0 1 1 1 0 0 1 0 1 0 0 1

Succinct Data Structures

The Idea

- Using minimum space close to the information-theoretic lower bound.
- Handle large problem sizes in the internal memory.

Constant Time Operations on Bit Vectors

$$\text{rank}_q(i) = \#q's \text{ up to position } i \rightarrow \text{rank}_1(4) = 3$$
$$\text{select}_q(i) = \text{position of } i^{\text{th}}\ q \rightarrow \text{select}_0(3) = 5$$

0 1 1 1 0 0 1 0 1 0 0 1

Succinct Data Structures

The Idea

- Using minimum space close to the information-theoretic lower bound.
- Handle large problem sizes in the internal memory.

Constant Time Operations on Bit Vectors

$\text{rank}_q(i) = \#q's$ up to position $i \rightarrow \text{rank}_1(4) = 3$

$\text{select}_q(i) =$ position of i^{th} $q \rightarrow \text{select}_0(3) = 5$

0	1	1	1	0	0	1	0	1	0	0	1
0	1	2	3	4	5						

Succinct Data Structures

The Idea

- Using minimum space close to the information-theoretic lower bound.
- Handle large problem sizes in the internal memory.

Constant Time Operations on Bit Vectors

$$\begin{aligned} \text{rank}_q(i) &= \#q's \text{ up to position } i \rightarrow \text{rank}_1(4) = 3 \\ \text{select}_q(i) &= \text{position of } i^{\text{th}}\ q \rightarrow \text{select}_0(3) = 5 \end{aligned}$$

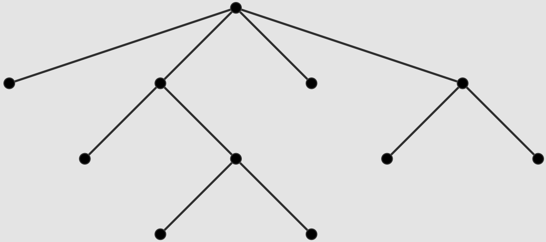
0 1 1 1 0 0 1 0 1 0 0 1

Example: Tree

- A tree $G = (V, E)$ can be represented as bit vector of length $2|V| + 1$.
- *rank* and *select* are available.

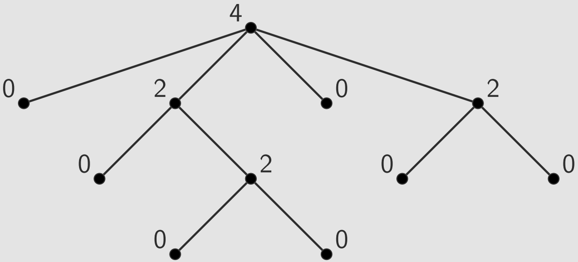
Succinct Tree

Tree as Bit Vector



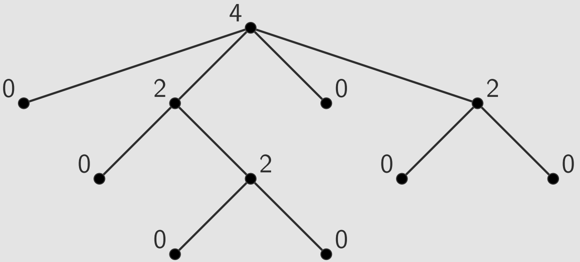
Succinct Tree

Tree as Bit Vector



Succinct Tree

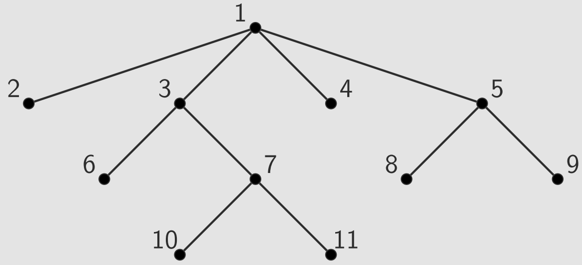
Tree as Bit Vector



4 0 2 0 2 0 2 0 0 0 0

Succinct Tree

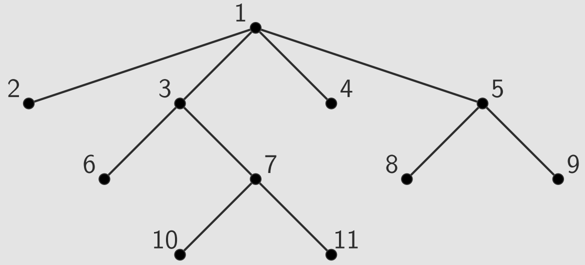
Tree as Bit Vector



4 0 2 0 2 0 2 0 0 0 0

Succinct Tree

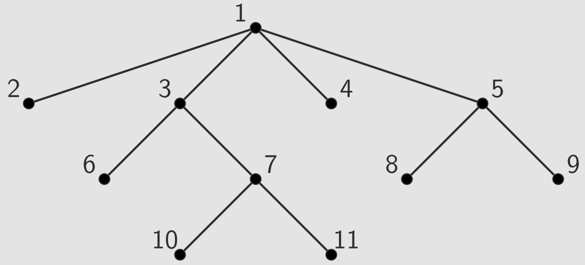
Tree as Bit Vector



4					0	2			0	2			0	2			0	0	0	0
1	1	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	0	0	0

Succinct Tree

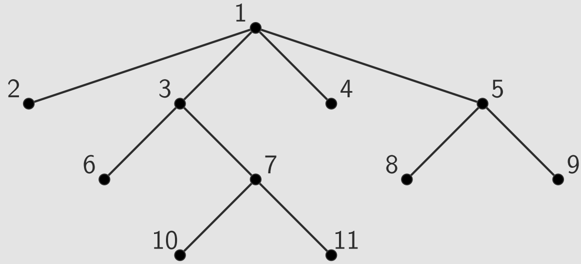
Tree as Bit Vector



		4					0	2			0	2			0	2			0	0	0	0
		1	1	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	0	0	0
1	0	1	1	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	0	0	0

Succinct Tree

Tree as Bit Vector

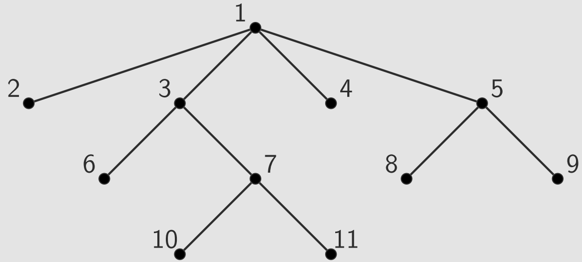


		4					0	2			0	2			0	2			0	0	0	0
		1	1	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	0	0	0
1	0	1	1	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	0	0	0

$$\text{parent}(v) = \text{rank}_0(\text{select}_1(v))$$

Succinct Tree

Tree as Bit Vector



		4					0	2			0	2			0	2			0	0	0	0
		1	1	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	0	0	0
1	0	1	1	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	0	0	0
1	1	2	3	4	5	2	3	6	7	4	5	8	9	6	7	10	11	8	9	10	11	12

$$\text{parent}(v) = \text{rank}_0(\text{select}_1(v))$$

State of the Art (1/2)

Internal Memory (RAM)

- SA, LCP and String B-trees are well studied.
- Construction in optimal time in theory.
- Well-performing practical implementations.

State of the Art (1/2)

Internal Memory (RAM)

- SA, LCP and String B-trees are well studied.
- Construction in optimal time in theory.
- Well-performing practical implementations.

External Memor

- Theoretical I/O-Optimal Algorithm exists for the SA.
- Recent work; still an active topic.
- LCP is needed for the construction of String B-Trees.

State of the Art (2/2)

Succinct Data Structures

- Working with large problem sizes.
- Almost exclusively only in internal memory.
- Practical implementation exists.
- Fast and space efficient constructions remain a major issue.

State of the Art (2/2)

Succinct Data Structures

- Working with large problem sizes.
- Almost exclusively only in internal memory.
- Practical implementation exists.
- Fast and space efficient constructions remain a major issue.

Parallel and Distributed Computing

- Parallel SA Algorithms (no implementations) at the end of the 80s.
- First practical implementations end of the end of last decade.
- GPGPU is possible and yields to better results.
- Large constant factor gap to the practical performance of the best sequential algorithms

Preliminary Work – External Memory

eSAIS [Bingmann et al., 2013]

- Based on SAIS: Linear time and fast in practice [Nong et al., 2009]
- Induced Sorting Principle.

Preliminary Work – External Memory

eSAIS [Bingmann et al., 2013]

- Based on SAIS: Linear time and fast in practice [Nong et al., 2009]
- Induced Sorting Principle.

Induced Sorting

i	0	1	2	3	4	5	6	7	8	9	10	11
$\mathrm{T}[i]$	B	A	A	N	N	A	A	N	N	A	A	\$
c-Bucket	\$				A			B			N	

Preliminary Work – External Memory

eSAIS [Bingmann et al., 2013]

- Based on SAIS: Linear time and fast in practice [Nong et al., 2009]
- Induced Sorting Principle.

Induced Sorting

i	0	1	2	3	4	5	6	7	8	9	10	11
$\mathrm{T}[i]$	B	A	A	N	N	A	A	N	N	A	A	$
c-Bucket	$	A						B	N			

Preliminary Work – External Memory

eSAIS [Bingmann et al., 2013]

- Based on SAIS: Linear time and fast in practice [Nong et al., 2009]
- Induced Sorting Principle.

Induced Sorting

i	0	1	2	3	4	5	6	7	8	9	10	11
$\mathrm{T}[i]$	B	A	A	N	N	A	A	N	N	A	A	$
c-Bucket	$	A						B	N			

Preliminary Work – External Memory

The Project

eSAIS [Bingmann et al., 2013]

- Based on SAIS: Linear time and fast in practice [Nong et al., 2009]
- Induced Sorting Principle.
- Computes SA and LCP in external memory.

Induced Sorting

<table>
<tr><td>i</td><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>10</td><td>11</td></tr>
<tr><td>$\mathrm{T}[i]$</td><td>B</td><td>A</td><td>A</td><td>N</td><td>N</td><td>A</td><td>A</td><td>N</td><td>N</td><td>A</td><td>A</td><td>$</td></tr>
<tr><td>c-Bucket</td><td>$</td><td colspan="6">A</td><td>B</td><td colspan="4">N</td></tr>
</table>

Preliminary Work – External Memory

eSAIS [Bingmann et al., 2013]

- Based on SAIS: Linear time and fast in practice [Nong et al., 2009]
- Induced Sorting Principle.
- Computes SA and LCP in external memory.

Induced Sorting

<table>
<tr><td>i</td><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>10</td><td>11</td></tr>
<tr><td>T[i]</td><td>B</td><td>A</td><td>A</td><td>N</td><td>N</td><td>A</td><td>A</td><td>N</td><td>N</td><td>A</td><td>A</td><td>$</td></tr>
<tr><td>c-Bucket</td><td>$</td><td colspan="6">A</td><td>B</td><td colspan="4">N</td></tr>
</table>

Results

- Outperforms all other external memory SA & LCP algorithms.
- 80 GigaByte XML dump with 4 Gigabyte main memory.
- 2.5 μseconds per character and 18 TiB I/O-volume.

Preliminary Work – Currently In Progress

Parallelization of DivSufSort

- Based on DivSufsort: Linear time and fast in practice [Mori, 2005]
- Induced Sorting Principle.
- Semi-parallelization possible.
- Computes SA and LCP.

Preliminary Work – Currently In Progress

Parallelization of DivSufSort

- Based on DivSufsort: Linear time and fast in practice [Mori, 2005]
- Induced Sorting Principle.
- Semi-parallelization possible.
- Computes SA and LCP.

Other Topics

- Bulk-parallel priority queue for SA and LCP construction.
- Hierarchic (integer) sorting on a really big cluster.

Objectives

Text indexing algorithms are not fit for data sizes > low terabyte regions.

Objectives

Text indexing algorithms are not fit for data sizes > low terabyte regions.

Key Objectives

1. Full use of local computing power:

Objectives

Text indexing algorithms are not fit for data sizes > low terabyte regions.

Key Objectives

1. Full use of local computing power:
   - Multicore and/or GPGPU

Objectives

Text indexing algorithms are not fit for data sizes > low terabyte regions.

Key Objectives

1. Full use of local computing power:
   - Multicore and/or GPGPU
   - Succinct Data Structures

Objectives

Text indexing algorithms are not fit for data sizes $>$ low terabyte regions.

Key Objectives

1. Full use of local computing power:
   - Multicore and/or GPGPU
   - Succinct Data Structures
   - External Memory

Objectives

Text indexing algorithms are not fit for data sizes > low terabyte regions.

Key Objectives

1. Full use of local computing power:
   - Multicore and/or GPGPU
   - Succinct Data Structures
   - External Memory
2. Computing text indices in a distributed environment.

Objectives

Text indexing algorithms are not fit for data sizes > low terabyte regions.

Key Objectives

1. Full use of local computing power:
   - Multicore and/or GPGPU
   - Succinct Data Structures
   - External Memory
2. Computing text indices in a distributed environment.
3. Later: Higher query thoughput of the resulting data structures.

Objectives

Text indexing algorithms are not fit for data sizes > low terabyte regions.

Key Objectives

1. Full use of local computing power:
   - Multicore and/or GPGPU
   - Succinct Data Structures
   - External Memory
2. Computing text indices in a distributed environment.
3. Later: Higher query thoughput of the resulting data structures.

Thank You!