String Synchronizing Sets

Sublinear-Time BWT Construction and Optimal LCE Data Structure

Dominik Kempa and **Tomasz Kociumaka**

THE UNIVERSITY OF WARWICK

UNIVERSITY OF WARSAW

Bar-Ilan University

STOC 2019

Phoenix, AZ June 25th, 2019

Burrows–Wheeler Transform: Definition

Burrows & Wheeler, 1994

T 1 1 0 1 0 0 1 0 1 0 1 0 1 0 0 1 0 1 0 0

BWT	
0	0
1	00
1	0010100
1	0010101010010100
1	0100
1	010010100
1	010010101010010100
0	010100
1	01010010100
1	0101010010100
0	010101010010100
0	100
0	10010100
0	10010101010010100
0	10100
0	1010010100
1	1010010101010010100
0	101010010100
0	10101010010100
0	11010010101010010100

Burrows–Wheeler Transform: Definition

Burrows & Wheeler, 1994

T 1 1 0 1 0 0 1 0 1 0 1 0 1 **0** 0 1 0 1 0 0

BWT	
0	0
1	00
1	0010100
1	0010101010010100
1	0100
1	010010100
1	0100101010010100
0	**010100**
1	01010010100
1	0101010010100
0	010101010010100
0	100
0	10010100
0	100101010010100
0	10100
0	1010010100
1	10100101010010100
0	101010010100
0	10101010010100
0	110100101010010100

Burrows–Wheeler Transform: Definition

Burrows & Wheeler, 1994

T 1 1 0 1 0 0 1 0 1 0 1 0 1 0 0 1 0 1 0 **0**

BWT	
0	0
1	00
1	0010100
1	0010101010010100
1	0100
1	010010100
1	0100101010010100
0	010100
1	01010010100
1	0101010010100
0	010101010010100
0	100
0	10010100
0	10010101010010100
0	10100
0	1010010100
1	101001010101010010100
0	101010010100
0	10101010010100
0	**11010010101010010100**

Burrows–Wheeler Transform: Applications

$$T : 1101001010101010010100$$
$$\mathrm{BWT}(T) : 0111111011000000001000$$

- First step in compression schemes, e.g., `bzip2`
  - If T is compressible, then $\mathrm{BWT}(T)$ has long **runs** of equal symbols.
  - **Simple** methods on $\mathrm{BWT}(T)$ instead of **difficult** methods on T.
- Main component of indexes solving many tasks in small space:
  - MinimalAbsentWord
  - LongestBorder
  - MaximalRepeats
  - MatchingStatistics
  - TandemRepeats
  - ApproxShortestSuperstring
  - LongestCommonSubstring
  - MaximalUniqueMatches
  - ShortestUniqueSubstring
  - LongestRepeatedFactor

BWT Construction

Selected construction algorithms:

Algorithm	Space (words)	Time
Classic (suffix trees)	$\mathcal{O}(n)$	$\mathcal{O}(n \log \sigma)$
Farach (FOCS'97)	$\mathcal{O}(n)$	$\mathcal{O}(n)$

BWT Construction

Selected construction algorithms:

Algorithm	Space (words)	Time
Classic (suffix trees)	$\mathcal{O}(n)$	$\mathcal{O}(n \log \sigma)$
Farach (FOCS'97)	$\mathcal{O}(n)$	$\mathcal{O}(n)$
Hon et al. (FOCS'03)	$\mathcal{O}(n/\log_\sigma n)$	$\mathcal{O}(n \log\log \sigma)$
Belazzougui (STOC'14)	$\mathcal{O}(n/\log_\sigma n)$	$\mathcal{O}(n)$ randomized
Munro et al. (SODA'17)	$\mathcal{O}(n/\log_\sigma n)$	$\mathcal{O}(n)$

BWT Construction

Selected construction algorithms:

Algorithm	Space (words)	Time
Classic (suffix trees)	$\mathcal{O}(n)$	$\mathcal{O}(n \log \sigma)$
Farach (FOCS'97)	$\mathcal{O}(n)$	$\mathcal{O}(n)$
Hon et al. (FOCS'03)	$\mathcal{O}(n/\log_\sigma n)$	$\mathcal{O}(n \log \log \sigma)$
Belazzougui (STOC'14)	$\mathcal{O}(n/\log_\sigma n)$	$\mathcal{O}(n)$ randomized
Munro et al. (SODA'17)	$\mathcal{O}(n/\log_\sigma n)$	$\mathcal{O}(n)$
This work	$\mathcal{O}(n/\log_\sigma n)$	$\mathcal{O}(n \log \sigma / \sqrt{\log n})$

BWT Construction

Selected construction algorithms for $\sigma = \mathcal{O}(1)$:

Algorithm	Space (words)	Time
Classic (suffix trees)	$\mathcal{O}(n)$	$\mathcal{O}(n \log \sigma)$
Farach (FOCS'97)	$\mathcal{O}(n)$	$\mathcal{O}(n)$
Hon et al. (FOCS'03)	$\mathcal{O}(n/\log_\sigma n)$	$\mathcal{O}(n \log\log \sigma)$
Belazzougui (STOC'14)	$\mathcal{O}(n/\log_\sigma n)$	$\mathcal{O}(n)$ randomized
Munro et al. (SODA'17)	$\mathcal{O}(n/\log_\sigma n)$	$\mathcal{O}(n)$
This work	$\mathcal{O}(n/\log_\sigma n)$	$\mathcal{O}(n \log \sigma/\sqrt{\log n})$

BWT Construction

Selected construction algorithms for $\sigma = \mathcal{O}(1)$:

Algorithm	Space (words)	Time
Classic (suffix trees)	$\mathcal{O}(n)$	$\mathcal{O}(n \log \sigma)$
Farach (FOCS'97)	$\mathcal{O}(n)$	$\mathcal{O}(n)$
Hon et al. (FOCS'03)	$\mathcal{O}(n/\log_\sigma n)$	$\mathcal{O}(n \log\log \sigma)$
Belazzougui (STOC'14)	$\mathcal{O}(n/\log_\sigma n)$	$\mathcal{O}(n)$ randomized
Munro et al. (SODA'17)	$\mathcal{O}(n/\log_\sigma n)$	$\mathcal{O}(n)$
This work	$\mathcal{O}(n/\log_\sigma n)$	$\mathcal{O}(n \log \sigma/\sqrt{\log n})$

Why $\mathcal{O}(n/\sqrt{\log n})$ rather than $\mathcal{O}(n/\log n)$ time?

BWT Construction

Selected construction algorithms for $\sigma = \mathcal{O}(1)$:

Algorithm	Space (words)	Time
Classic (suffix trees)	$\mathcal{O}(n)$	$\mathcal{O}(n \log \sigma)$
Farach (FOCS'97)	$\mathcal{O}(n)$	$\mathcal{O}(n)$
Hon et al. (FOCS'03)	$\mathcal{O}(n/\log_\sigma n)$	$\mathcal{O}(n \log\log \sigma)$
Belazzougui (STOC'14)	$\mathcal{O}(n/\log_\sigma n)$	$\mathcal{O}(n)$ randomized
Munro et al. (SODA'17)	$\mathcal{O}(n/\log_\sigma n)$	$\mathcal{O}(n)$
This work	$\mathcal{O}(n/\log_\sigma n)$	$\mathcal{O}(n \log \sigma/\sqrt{\log n})$

Why $\mathcal{O}(n/\sqrt{\log n})$ rather than $\mathcal{O}(n/\log n)$ time?

BWT construction in $o(n/\sqrt{\log n})$ time for binary length-n strings

$\Downarrow$

Counting inversions in $o(m\sqrt{\log m})$ time for length-m permutations

Would improve upon the algorithm by Chan and Pătrașcu (SODA 2010).

Longest Common Extension Queries

Landau & Vishkin, J. Comput. Syst. Sci. 1988

Definition

The **Longest Common Extension** $\mathrm{LCE}(i, j)$ is the length of the longest common prefix of $T[i \mathinner{.\,.} n]$ and $T[j \mathinner{.\,.} n]$.

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
1	1	0	1	0	0	1	0	1	0	1	0	1	0	0	1	0	1	0	0

$$\mathrm{LCE}(4, 13) = 7$$

Longest Common Extension Queries

Landau & Vishkin, J. Comput. Syst. Sci. 1988

Definition

The **Longest Common Extension** $\mathrm{LCE}(i, j)$ is the length of the longest common prefix of $T[i \mathinner{..} n]$ and $T[j \mathinner{..} n]$.

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
1	1	0	1	0	0	1	0	1	0	1	0	1	0	0	1	0	1	0	0

$$\mathrm{LCE}(4, 13) = 7$$

Used as a **subroutine** in many algorithms and data structures such as for:

- approximate pattern matching (the **kangaroo method**),
- discovery of repetitions in strings,
- construction of text indexing data structures.

Data Structures for LCE Queries

Data structures supporting constant-time LCE queries:

Algorithm	Space (words)	Construction Time
Landau & Vishkin (JCSS'88)	$\mathcal{O}(n)$	$\mathcal{O}(n \log \sigma)$
Farach (FOCS'97)	$\mathcal{O}(n)$	$\mathcal{O}(n)$

Data Structures for LCE Queries

Data structures supporting constant-time LCE queries:

Algorithm	Space (words)	Construction Time
Landau & Vishkin (JCSS'88)	$\mathcal{O}(n)$	$\mathcal{O}(n \log \sigma)$
Farach (FOCS'97)	$\mathcal{O}(n)$	$\mathcal{O}(n)$
Tanimura et al. (MFCS'17)	$\mathcal{O}(n\sqrt{\log\sigma}/\sqrt{\log_\sigma n})$	–
Munro et al. (arXiv'17)	$\mathcal{O}(n/\sqrt{\log_\sigma n})$	$\mathcal{O}(n/\sqrt{\log_\sigma n})$
Birenzwige et al. (arXiv'18)	$\mathcal{O}(n/\log_\sigma n)$	$\mathcal{O}(n)$ randomized

Data Structures for LCE Queries

Data structures supporting constant-time LCE queries:

Algorithm	Space (words)	Construction Time
Landau & Vishkin (JCSS'88)	$\mathcal{O}(n)$	$\mathcal{O}(n \log \sigma)$
Farach (FOCS'97)	$\mathcal{O}(n)$	$\mathcal{O}(n)$
Tanimura et al. (MFCS'17)	$\mathcal{O}(n\sqrt{\log \sigma}/\sqrt{\log_\sigma n})$	–
Munro et al. (arXiv'17)	$\mathcal{O}(n/\sqrt{\log_\sigma n})$	$\mathcal{O}(n/\sqrt{\log_\sigma n})$
Birenzwige et al. (arXiv'18)	$\mathcal{O}(n/\log_\sigma n)$	$\mathcal{O}(n)$ randomized
this work	$\mathcal{O}(n/\log_\sigma n)$	$\mathcal{O}(n/\log_\sigma n)$

Asymptotically optimal for each alphabet size!

BWT construction algorithm

Toy special case:

T is binary except for 2's at $\Theta(\frac{n}{\tau})$ positions, at least one every τ positions.

Example for $\tau = 4$:

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
2	1	0	1	2	0	1	2	1	0	1	2	1	0	2	1	0	2	0	2

Toy special case:

T is binary except for 2's at $\Theta(\frac{n}{\tau})$ positions, at least one every τ positions.

Example for $\tau = 4$:

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
2	1	0	1	2	0	1	2	1	0	1	2	1	0	2	1	0	2	0	2

Fact: Suffixes starting with a 2 can be sorted in $\mathcal{O}(n/\tau)$ time.

Strings with a Planted Synchronizing Set

Toy special case:

T is binary except for 2's at $\Theta(\frac{n}{\tau})$ positions, at least one every τ positions.

Example for $\tau = 4$:

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
2	1	0	1	2	0	1	2	1	0	1	2	1	0	2	1	0	2	0	2

Fact: Suffixes starting with a 2 can be sorted in $\mathcal{O}(n/\tau)$ time.

20	2
5	2012101210210202
18	202
1	21012012101210210202
8	2101210210202
15	210202
12	210210202

Strings with a Planted Synchronizing Set

Toy special case:

T is binary except for 2's at $\Theta(\frac{n}{\tau})$ positions, at least one every τ positions.

Example for $\tau = 4$:

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
2	1	0	1	2	0	1	2	1	0	1	2	1	0	2	1	0	2	0	2

Fact: Suffixes starting with a 2 can be sorted in $\mathcal{O}(n/\tau)$ time.

20	2
5	2012101210210202
18	202
1	21012012101210210202
8	2101210210202
15	210202
12	210210202

Strings with a Planted Synchronizing Set

Toy special case:

T is binary except for 2's at $\Theta(\frac{n}{\tau})$ positions, at least one every τ positions.

Example for $\tau = 4$:

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
2	1	0	1	2	0	1	2	1	0	1	2	1	0	2	1	0	2	0	2
D				B			D				E			E			C		A

Fact: Suffixes starting with a 2 can be sorted in $\mathcal{O}(n/\tau)$ time.

20	2
5	2012101210210202
18	202
1	21012012101210210202
8	2101210210202
15	210202
12	210210202

Toy special case:

T is binary except for 2's at $\Theta(\frac{n}{\tau})$ positions, at least one every τ positions.

Example for $\tau = 4$:

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
2	1	0	1	2	0	1	2	1	0	1	2	1	0	2	1	0	2	0	2
D				B			D				E			E			C		A

Fact: Suffixes starting with a 2 can be sorted in $\mathcal{O}(n/\tau)$ time.

7	A	20	2
2	BDEECA	5	2012101210210202
6	CA	18	202
1	DBDEECA	1	21012012101210210202
3	DEECA	8	2101210210202
5	ECA	15	210202
4	EECA	12	210210202

Strings with a Planted Synchronizing Set

Toy special case:

T is binary except for 2's at $\Theta(\frac{n}{\tau})$ positions, at least one every τ positions.

Example for $\tau = 4$:

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
2	1	0	1	2	0	1	2	1	0	1	2	1	0	2	1	0	2	0	2
D				B			D				E			E			C		A

Fact: Suffixes starting with a 2 can be sorted in $\mathcal{O}(n/\tau)$ time.

7	A	20	2
2	BDEECA	5	2012101210210202
6	CA	18	202
1	DBDEECA	1	21012012101210210202
3	DEECA	8	2101210210202
5	ECA	15	210202
4	EECA	12	210210202

Toy special case:

T is binary except for 2's at $\Theta(\frac{n}{\tau})$ positions, at least one every τ positions.

Example for $\tau = 4$:

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
2	1	0	1	2	0	1	2	1	0	1	2	1	0	2	1	0	2	0	2
D				B			D				E			E			C		A

Fact: Suffixes starting with a 2 can be sorted in $\mathcal{O}(n/\tau)$ time.

7	A	20	2
2	BDEECA	5	2012101210210202
6	CA	18	202
1	DBDEECA	1	21012012101210210202
3	DEECA	8	2101210210202
5	ECA	15	210202
4	EECA	12	210210202

Structure of the Burrows–Wheeler Transform

012012101210210202
012101210210202
01210210202
02
0202
0210202
1012012101210210202
101210210202
10202
10210202
12012101210210202
12101210210202
1210210202
2
2012101210210202
202
21012012101210210202
2101210210202
210202
210210202

Structure of the Burrows–Wheeler Transform

012012101210210202
012101210210202
01210210202
02
0202
0210202
1012012101210210202
101210210202
10202
10210202
12012101210210202
12101210210202
1210210202
2
2012101210210202
202
21012012101210210202
2101210210202
210202
210210202

Structure of the Burrows–Wheeler Transform

012012101210210202 012101210210202 01210210202
02 0202 0210202
1012012101210210202 101210210202
10202 10210202
12012101210210202 12101210210202 1210210202
2 2012101210210202 202 21012012101210210202 2101210210202 210202 210210202

Structure of the Burrows–Wheeler Transform

	012012101210210202
	012101210210202
	01210210202
	02
	0202
	0210202
	1012012101210210202
	101210210202
	10202
	10210202
	12012101210210202
	12101210210202
	1210210202
20	2
2101	2012101210210202
210	202
2	21012012101210210202
201	2101210210202
210	210202
2101	210210202

Structure of the Burrows–Wheeler Transform

	012012101210210202
	012101210210202
	01210210202
2	02
21	0202
21	0210202
	1012012101210210202
	101210210202
	10202
	10210202
	12012101210210202
	12101210210202
	1210210202
20	**2**
2101	2012101210210202
210	**202**
2	21012012101210210202
201	2101210210202
210	**210202**
2101	210210202

0

Structure of the Burrows–Wheeler Transform

		012012101210210202
		0121012102 10202
		01210210202
	2	02
	21	**0202**
	21	**0210202**
1		1012012101210210202
		101210210202
	2	10202
	2	10210202
0		12012101210210202
		121012102 10202
		1210210202
	20	2
	2101	2012101210210202
	210	202
	2	21012012101210210202
	201	21012102 10202
	210	210202
	2101	210210202

Structure of the Burrows–Wheeler Transform

	012012101210210202
	012101210210202
	01210210202
2	02
21	0202
21	0210202
	1012012101210210202
	101210210202
2	10202
2	10210202
210	12012101210210202
20	12101210210202
210	1210210202
20	2
2101	**2012101210210202**
210	202
2	21012012101210210202
201	**2101210210202**
210	210202
2101	**210210202**

Structure of the Burrows–Wheeler Transform

21	012012101210210202
2	012101210210202
21	01210210202
2	02
21	0202
21	0210202
	1012012101210210202
	101210210202
2	10202
2	10210202
210	**12012101210210202**
20	**12101210210202**
210	**1210210202**
20	2
2101	2012101210210202
210	202
2	21012012101210210202
201	2101210210202
210	210202
2101	210210202

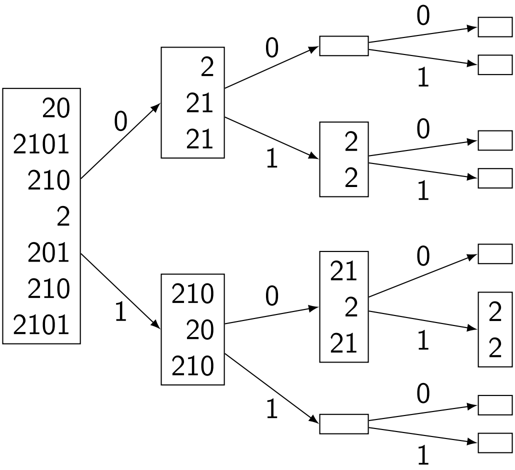
Structure of the Burrows–Wheeler Transform

21	**012**012101210210202
2	012101210210202
21	**012**10210202
2	02
21	0202
21	0210202
2	1012012101210210202
2	101210210202
2	10202
2	10210202
210	12012101210210202
20	12101210210202
210	1210210202
20	2
2101	2012101210210202
210	202
2	21012012101210210202
201	2101210210202
210	210202
2101	210210202

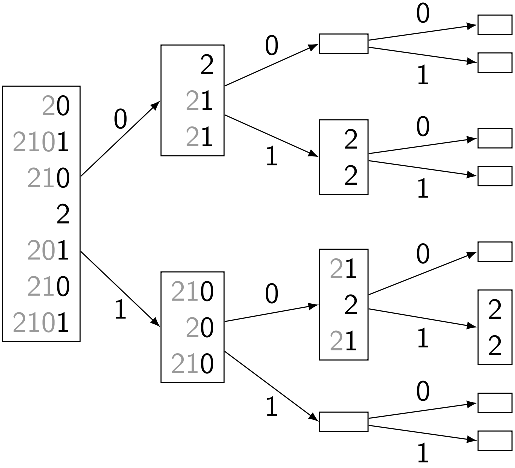
Structure of the Burrows–Wheeler Transform

21	012012101210210202
2	012101210210202
21	01210210202
2	02
21	0202
21	0210202
2	1012012101210210202
2	101210210202
2	10202
2	10210202
210	12012101210210202
20	12101210210202
210	1210210202
20	2
2101	2012101210210202
210	202
2	21012012101210210202
201	2101210210202
210	210202
2101	210210202

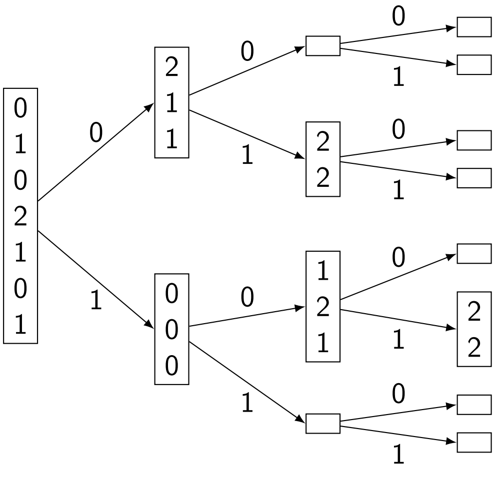
Wavelet Trees



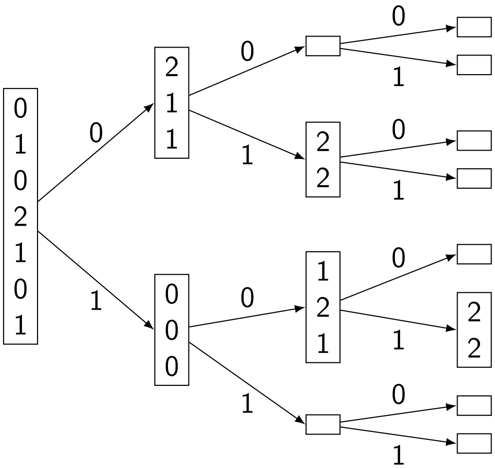
Wavelet Trees



Wavelet Trees



Wavelet Trees



Theorem (Munro et al., SPIRE'14; Babenko et al., SODA'15)

The wavelet tree of a sequence of m items with b bits each can be computed in $\mathcal{O}(mb/\sqrt{\log m})$ time using $\mathcal{O}(mb/\log m)$ space.

BWT Construction

2 1 0 1 2 0 1 2 1 0 1 2 1 0 2 1 0 2 0 2

BWT Construction

```
2 1 0 1 2 0 1 2 1 0 1 2 1 0 2 1 0 2 0 2
D       B   D       E     E     C   A
```

BWT Construction

2 1 0 1 2 0 1 2 1 0 1 2 1 0 2 1 0 2 0 2
D B D E E C A

7 A
2 BDEECA
6 CA
1 DBDEECA
3 DEECA
5 ECA
4 EECA

BWT Construction

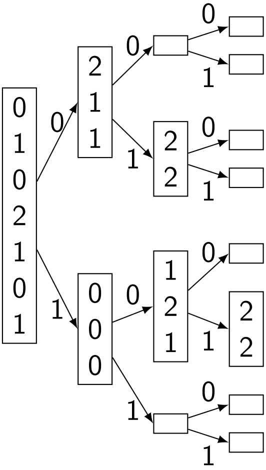
```
2 1 0 1 2 0 1 2 1 0 1 2 1 0 2 1 0 2 0 2
D       B   D       E     E     C   A
```

20	7	A
2101	2	BDEECA
210	6	CA
2	1	DBDEECA
201	3	DEECA
210	5	ECA
2101	4	EECA

BWT Construction

2 1 0 1 2 0 1 2 1 0 1 2 1 0 2 1 0 2 0 2
D B D E E C A

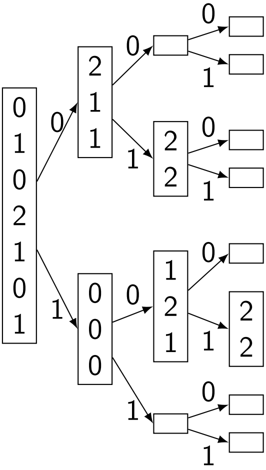
20	7	A
2101	2	BDEECA
210	6	CA
2	1	DBDEECA
201	3	DEECA
210	5	ECA
2101	4	EECA



BWT Construction

2 1 0 1 2 0 1 2 1 0 1 2 1 0 2 1 0 2 0 2
D B D E E C A

20	7	A
2101	2	BDEECA
210	6	CA
2	1	DBDEECA
201	3	DEECA
210	5	ECA
2101	4	EECA



012

02

1012

102

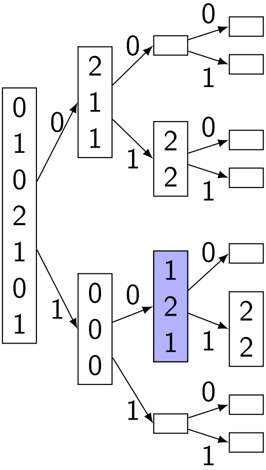
12

2

BWT Construction

2 1 0 1 2 0 1 2 1 0 1 2 1 0 2 1 0 2 0 2
D B D E E C A

20	7	A
2101	2	BDEECA
210	6	CA
2	1	DBDEECA
201	3	DEECA
210	5	ECA
2101	4	EECA



1
2 012
1

02

1012

102

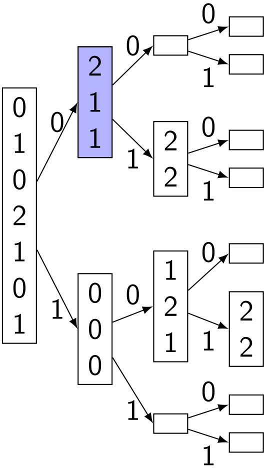
12

2

BWT Construction

2 1 0 1 2 0 1 2 1 0 1 2 1 0 2 1 0 2 0 2
D B D E E C A

20	7	A
2101	2	BDEECA
210	6	CA
2	1	DBDEECA
201	3	DEECA
210	5	ECA
2101	4	EECA



1
2 012
1

2
1 02
1

1012

102

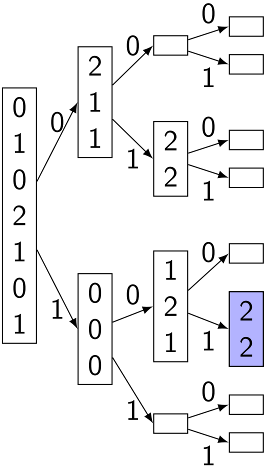
12

2

BWT Construction

2 1 0 1 2 0 1 2 1 0 1 2 1 0 2 1 0 2 0 2
D B D E E C A

20	7	A
2101	2	BDEECA
210	6	CA
2	1	DBDEECA
201	3	DEECA
210	5	ECA
2101	4	EECA

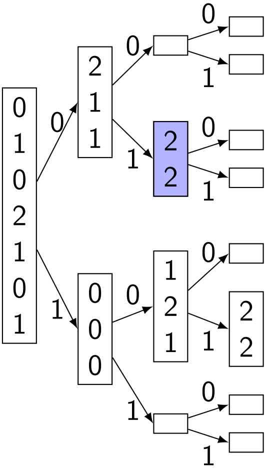


1, 2, 1	012
2, 1, 1	02
2, 2	1012
	102
	12
	2

BWT Construction

2 1 0 1 2 0 1 2 1 0 1 2 1 0 2 1 0 2 0 2
D B D E E C A

20	7	A
2101	2	BDEECA
210	6	CA
2	1	DBDEECA
201	3	DEECA
210	5	ECA
2101	4	EECA

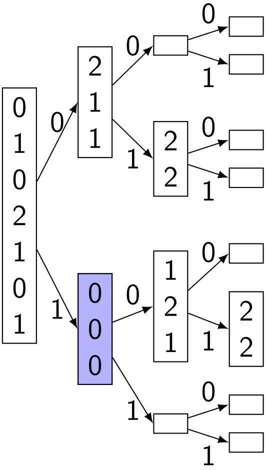


1 2 1	012
2 1 1	02
2 2	1012
2 2	102
	12
	2

BWT Construction

2 1 0 1 2 0 1 2 1 0 1 2 1 0 2 1 0 2 0 2
D B D E E C A

20	7	A
2101	2	BDEECA
210	6	CA
2	1	DBDEECA
201	3	DEECA
210	5	ECA
2101	4	EECA

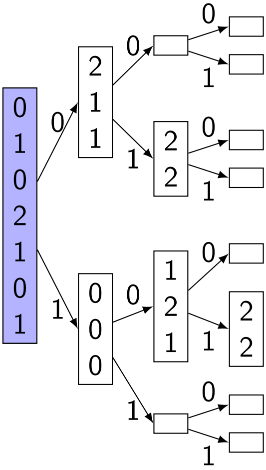


1 2 1	012
2 1 1	02
2 2	1012
2 2	102
0 0 0	12
	2

BWT Construction

2 1 0 1 2 0 1 2 1 0 1 2 1 0 2 1 0 2 0 2
D B D E E C A

20	7	A
2101	2	BDEECA
210	6	CA
2	1	DBDEECA
201	3	DEECA
210	5	ECA
2101	4	EECA



1 2 1	012
2 1 1	02
2 2	1012
2 2	102
0 0 0	12
0 1 0 2 1 0 1	2

String Synchronizing Sets

A τ-**synchronizing set** of T is a set of positions S that is

small: $|\mathsf{S}| = \mathcal{O}(\frac{n}{\tau})$;

consistent: whether $i \in \mathsf{S}$ depends only on $T[i \mathinner{.\,.} i + 2\tau - 1]$,

dense: $\mathsf{S} \cap [i \mathinner{.\,.} i + \tau - 1] \neq \emptyset$ for $i \in [1 \mathinner{.\,.} n - 3\tau + 2]$.

String Synchronizing Sets

A τ-**synchronizing set** of T is a set of positions S that is

small: $|\mathsf{S}| = \mathcal{O}(\frac{n}{\tau})$;

consistent: whether $i \in \mathsf{S}$ depends only on $T[i \mathinner{.\,.} i+2\tau-1]$,

dense: $\mathsf{S} \cap [i \mathinner{.\,.} i+\tau-1] \neq \emptyset$ for $i \in [1 \mathinner{.\,.} n-3\tau+2]$
if and only if $\mathrm{per}(T[i \mathinner{.\,.} i+3\tau-1]) > \frac{1}{3}\tau$.

String Synchronizing Sets

A τ-**synchronizing set** of T is a set of positions S that is

- small: $|\mathsf{S}| = \mathcal{O}(\frac{n}{\tau})$;
- consistent: whether $i \in \mathsf{S}$ depends only on $T[i \mathinner{.\,.} i + 2\tau - 1]$,
- dense: $\mathsf{S} \cap [i \mathinner{.\,.} i + \tau - 1] \neq \emptyset$ for $i \in [1 \mathinner{.\,.} n - 3\tau + 2]$ if and only if $\mathrm{per}(T[i \mathinner{.\,.} i + 3\tau - 1]) > \frac{1}{3}\tau$.

Theorem

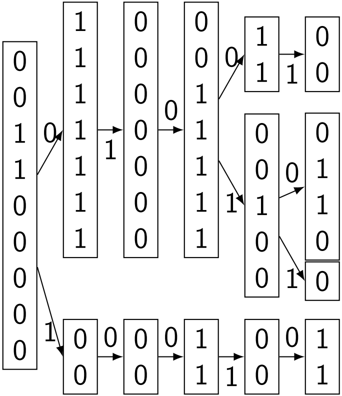
Given a text T and a positive integer τ, one can deterministically construct a τ-synchronizing set of size $\mathcal{O}(\frac{n}{\tau})$:

- *in $\mathcal{O}(n)$ time in general,*
- *in $\mathcal{O}(\frac{n}{\tau})$ time if $\tau \leqslant \frac{1}{5}\log_\sigma n$.*

BWT Construction

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
1	<u>1</u>	0	<u>1</u>	<u>0</u>	0	<u>1</u>	0	<u>1</u>	0	<u>1</u>	0	<u>1</u>	<u>0</u>	0	<u>1</u>	0	1	0	0
	D		B	A		E		E		D		B	A		C				

01	0010	14	AC
01	0010	5	AEEDBAC
10	1001	13	BAC
10	1001	4	BAEEDBAC
00	1010	16	C
10	1010	11	DBAC
01	1010	2	DBAEEDBAC
10	1010	9	EDBAC
00	1010	7	EEDBAC



0	0
1	00
1 1	0010
1	0100
1 1	01001
0 1 1 0	01010
0	100
0 0	1001
0 0 1 0 0	1010
0	11010

Conclusions

Our contributions:

1. BWT construction: $\mathcal{O}(n \log \sigma / \sqrt{\log n})$ time, $\mathcal{O}(n / \log_\sigma n)$ space.
2. LCE queries in $\mathcal{O}(1)$ time after $\mathcal{O}(n / \log_\sigma n)$-time preprocessing.
3. The notion of τ-synchronizing sets.

Conclusions

Our contributions:

1. BWT construction: $\mathcal{O}(n \log \sigma / \sqrt{\log n})$ time, $\mathcal{O}(n / \log_\sigma n)$ space.
2. LCE queries in $\mathcal{O}(1)$ time after $\mathcal{O}(n / \log_\sigma n)$-time preprocessing.
3. The notion of τ-synchronizing sets.

Further work:

1. Lower bounds for BWT construction with any alphabet size σ.
2. Sublinear-time construction of further objects.
3. External-memory counterpart.
4. Improve the running time wrt. more subtle instance size measures.

Conclusions

Our contributions:

1. BWT construction: $\mathcal{O}(n \log \sigma / \sqrt{\log n})$ time, $\mathcal{O}(n / \log_\sigma n)$ space.
2. LCE queries in $\mathcal{O}(1)$ time after $\mathcal{O}(n / \log_\sigma n)$-time preprocessing.
3. The notion of τ-synchronizing sets.

Further work:

1. Lower bounds for BWT construction with any alphabet size σ.
2. Sublinear-time construction of further objects.
3. External-memory counterpart.
4. Improve the running time wrt. more subtle instance size measures.

Thank you for your attention!