In-Place BWT

Maxime Crochemore Roberto Grossi
Juha Kärkkäinen Gad M. Landau

CPM 2013

Burrows-Wheeler Transform (BWT)

text: BANANA$

BWT	sorted suffixes
A	$
N	A$
N	ANA$
B	ANANA$
$	BANANA$
A	NA$
A	NANA$

- Many applications in text compression, indexing, mining etc.

Burrows-Wheeler Transform (BWT)

text: B A N A N A $

BWT	sorted suffixes
A	$
N	A $
N	A N A $
B	A N A N A $
$	B A N A N A $
A	N A $
A	N A N A $

- Many applications in text compression, indexing, mining etc.

Burrows-Wheeler Transform (BWT)

text: B A N A N A $

BWT	sorted suffixes
A	$
N	A $
N	A N A $
B	A N A N A $
$	B A N A N A $
A	N A $
A	N A N A $

- Many applications in text compression, indexing, mining etc.

Burrows-Wheeler Transform (BWT)

text: BANANA$

BWT	sorted suffixes
A	$
N	A$
N	ANA$
B	ANANA$
$	BANANA$
A	NA$
A	NANA$

- Many applications in text compression, indexing, mining etc.

Burrows-Wheeler Transform (BWT)

text: BANANA$

BWT	sorted suffixes
A	$
N	A$
N	ANA$
B	ANANA$
$	BANANA$
A	NA$
A	NANA$

- Many applications in text compression, indexing, mining etc.
- BWT is a permutation of text
  - in-place BWT
- BWT is invertible transform
  - in-place inverse BWT

Computing BWT

Standard construction using suffix array

	time	space (bits)
Suffix array construction		
many algorithms	$\mathcal{O}(n)$	$> n \log n + n \log \sigma$
Franceschini & Muthu, 2007	$\mathcal{O}(n \log n)$	$n \log n + n \log \sigma + \mathcal{O}(\log n)$

Computing BWT

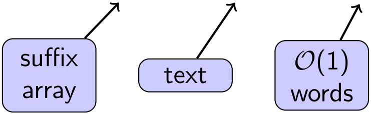
Standard construction using suffix array

	time	space (bits)
Suffix array construction		
many algorithms	$\mathcal{O}(n)$	$> n \log n + n \log \sigma$
Franceschini & Muthu, 2007	$\mathcal{O}(n \log n)$	$\underbrace{n \log n}_{\text{suffix array}} + \underbrace{n \log \sigma}_{\text{text}} + \underbrace{\mathcal{O}(\log n)}_{\mathcal{O}(1)\text{ words}}$

Computing BWT

Standard construction using suffix array

	time	space (bits)
Suffix array construction		
many algorithms	$\mathcal{O}(n)$	$> n \log n + n \log \sigma$
Franceschini & Muthu, 2007	$\mathcal{O}(n \log n)$	$n \log n + n \log \sigma + \mathcal{O}(\log n)$



suffix array

text

$\mathcal{O}(1)$ words

Human genome

- $n \log n = 34n$
- $n \log \sigma = 2n$

Computing BWT

	time	space (bits)
Suffix array construction		
many algorithms	$\mathcal{O}(n)$	$> n \log n + n \log \sigma$
Franceschini & Muthu, 2007	$\mathcal{O}(n \log n)$	$n \log n + n \log \sigma + \mathcal{O}(\log n)$
Direct BWT construction		
K, 2007	$\mathcal{O}(n/\epsilon^2)$	$2n \log \sigma + \mathcal{O}(\epsilon n \log n)$
Okanohara & Sadakane, 2009	$\mathcal{O}(n)$	$\mathcal{O}(n \log \sigma \log \log_\sigma n)$
Succinct index construction		
Hon & al., 2007	$\mathcal{O}(n \log n)$	$n \log \sigma + \mathcal{O}(nH_0)$
Hon & al., 2009	$\mathcal{O}(n \log \log \sigma)$	$\mathcal{O}(n \log \sigma)$

Computing BWT

	time	space (bits)
Suffix array construction		
many algorithms	$\mathcal{O}(n)$	$> n\log n + n\log\sigma$
Franceschini & Muthu, 2007	$\mathcal{O}(n\log n)$	$n\log n + n\log\sigma$ $+\mathcal{O}(\log n)$
Direct BWT construction		
K, 2007	$\mathcal{O}(n/\epsilon^2)$	$2n\log\sigma + \mathcal{O}(\epsilon n\log n)$
Okanohara & Sadakane, 2009	$\mathcal{O}(n)$	$\mathcal{O}(n\log\sigma\log\log_\sigma n)$
Succinct index construction		
Hon & al., 2007	$\mathcal{O}(n\log n)$	$n\log\sigma + \mathcal{O}(nH_0)$
Hon & al., 2009	$\mathcal{O}(n\log\log\sigma)$	$\mathcal{O}(n\log\sigma)$
In-place BWT construction		
This talk	$\mathcal{O}(n^2)$	$\underbrace{n\log\sigma}_{\text{text on input, BWT on output}} + \underbrace{\mathcal{O}(\log n)}_{\mathcal{O}(1)\text{ words}}$

In-Place BWT Construction

In-place sorting

- Easy: heapsort
- Freedom to move elements
- Encode information in element order

In-place suffix sorting

- Freedom to move pointers
- Easy in $\mathcal{O}(n^2)$ time
- $\mathcal{O}(n \log n)$ time (Franceshini & Muthu, 2007)

In-place BWT

- Single character move changes many suffixes
- Non-trivial in any time

B A N A N A $

BWT	sorted suffixes
A	$
N	A $
N	A N A $
B	A N A N A $
$	B A N A N A $
A	N A $
A	N A N A $

In-Place BWT Construction

In-place sorting

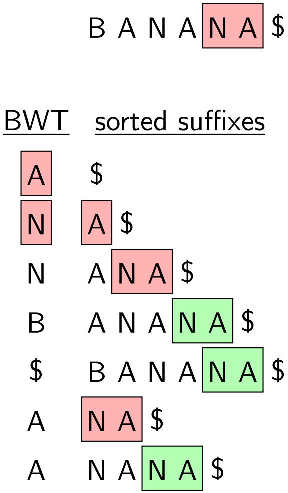
- Easy: heapsort
- Freedom to move elements
- Encode information in element order

In-place suffix sorting

- Freedom to move pointers
- Easy in $\mathcal{O}(n^2)$ time
- $\mathcal{O}(n \log n)$ time (Franceshini & Muthu, 2007)

In-place BWT

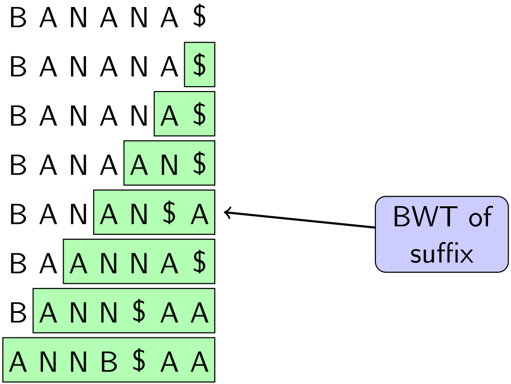
- Single character move changes many suffixes
- Non-trivial in any time



B A N A N A $

BWT	sorted suffixes
A	$
N	A $
N	A N A $
B	A N A N A $
$	B A N A N A $
A	N A $
A	N A N A $

Basic Idea: Incremental construction from end



B A N A N A $

B A N A N A $

B A N A N A $

B A N A A N $

B A N A N $ A

B A A N N A $

B A N N $ A A

A N N B $ A A

BWT of suffix

Incremental Step

N A N A $

BWT	sorted suffixes
A	$
N	A $
N	A N A $
A	N A $
$	N A N A $

A N A N A $

BWT	sorted suffixes
A	$
N	A $
N	A N A $
$	A N A N A $
A	N A $
A	N A N A $

Incremental Step

1. Replace $ with new first character

N A N A $

BWT	sorted suffixes
A	$
N	A $
N	A N A $
A	N A $
$	N A N A $

A N A N A $

BWT	sorted suffixes
A	$
N	A $
N	A N A $
$	A N A N A $
A	N A $
A	N A N A $

Incremental Step

1. Replace $ with new first character
2. Insert new suffix and preceding character $
   - No change to ordering of other suffixes

NANA$

BWT	sorted suffixes
A	$
N	A$
N	ANA$
A	NA$
$	NANA$

ANANA$

BWT	sorted suffixes
A	$
N	A$
N	ANA$
$	ANANA$
A	NA$
A	NANA$

Finding place of new suffix

BWT	sorted suffixes
A	
N	
N	
$	
A	
A	

Finding place of new suffix

- First suffix column is stable sorting of BWT

BWT	sorted suffixes
A	$
N	A
N	A
$	A
A	N
A	N

Finding place of new suffix

- First suffix column is stable sorting of BWT
- The new suffix starts with the new first character

BWT	sorted suffixes
A	$
N	A
N	A
$	A
A	N
A	N

Finding place of new suffix

- First suffix column is stable sorting of BWT
- The new suffix starts with the new first character
- To determine the position, we need to count

BWT	sorted suffixes
A	$
N	A
N	A
$	A
A	N
A	N

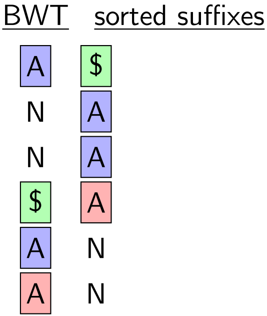
Finding place of new suffix

- First suffix column is stable sorting of BWT
- The new suffix starts with the new first character
- To determine the position, we need to count
  - smaller characters

BWT	sorted suffixes
A	$
N	A
N	A
$	A
A	N
A	N

Finding place of new suffix

- First suffix column is stable sorting of BWT
- The new suffix starts with the new first character
- To determine the position, we need to count
  - smaller characters
  - preceding equal characters (rank)



BWT	sorted suffixes
A	$
N	A
N	A
$	A
A	N
A	N

Analysis

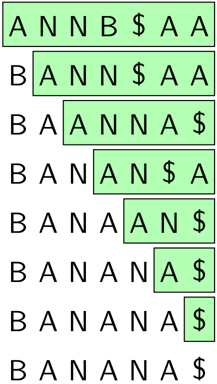
- $\mathcal{O}(n)$ incremental steps
- Step i needs $\mathcal{O}(i)$ time for
  - counting characters
  - inserting by moving $\mathcal{O}(i)$ characters
- Extra space needed for $\mathcal{O}(1)$ pointers, counters and characters
- Characters are only compared and moved

Theorem

The Burrows–Wheeler transform of a text of length n over a general alphabet can be compute in-place in $\mathcal{O}(n^2)$ time using $\mathcal{O}(1)$ words of extra space.

In-Place Inverse BWT

Same steps in reverse order and direction



A N N B $ A A

B A N N $ A A

B A A N N A $

B A N A N $ A

B A N A A N $

B A N A N A $

B A N A N A $

B A N A N A $

Inverse Incremental Step

A N A N A $

BWT	sorted suffixes
A	$
N	A $
N	A N A $
$	A N A N A $
A	N A $
A	N A N A $

N A N A $

BWT	sorted suffixes
A	$
N	A $
N	A N A $
A	N A $
$	N A N A $

Inverse Incremental Step

1. Delete $

A N A N A $

BWT	sorted suffixes
A	$
N	A $
N	A N A $
$	A N A N A $
A	N A $
A	N A N A $

N A N A $

BWT	sorted suffixes
A	$
N	A $
N	A N A $
A	N A $
$	N A N A $

Inverse Incremental Step

1. Delete $
2. Replace removed first text character with $

A N A N A $

BWT	sorted suffixes
A	$
N	A $
N	A N A $
$	A N A N A $
A	N A $
A	N A N A $

N A N A $

BWT	sorted suffixes
A	$
N	A $
N	A N A $
A	N A $
$	N A N A $

Finding first text character

BWT	sorted suffixes
A	
N	
N	
$	
A	
A	

Finding first text character

- First suffix column is stable sorting of BWT

BWT	sorted suffixes
A	$
N	A
N	A
$	A
A	N
A	N

Finding first text character

- First suffix column is stable sorting of BWT
- $ precedes first text character

BWT	sorted suffixes
A	$
N	A
N	A
$	A
A	N
A	N

Finding first text character

- First suffix column is stable sorting of BWT
- $ precedes first text character
- Find which character that is (read-only selection)

BWT	sorted suffixes
A	$
N	A
N	A
$	A
A	N
A	N

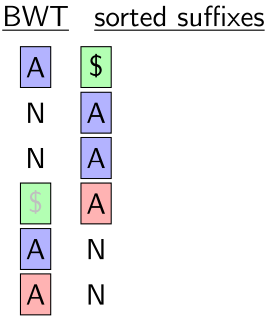
Finding first text character

- First suffix column is stable sorting of BWT
- $ precedes first text character
- Find which character that is (read-only selection)
- Determine rank (number of preceding equal characters) by counting smaller characters

BWT	sorted suffixes
A	$
N	A
N	A
$	A
A	N
A	N

Finding first text character

- First suffix column is stable sorting of BWT
- $ precedes first text character
- Find which character that is (read-only selection)
- Determine rank (number of preceding equal characters) by counting smaller characters
- Find the position in BWT (select)



BWT	sorted suffixes
A	$
N	A
N	A
$	A
A	N
A	N

Read-Only Selection

- Incremental step time is dominated by read-only selection time

Problem

Given unsorted array of size n and integer k, find the kth value in sorted order without modifying array.

Read-Only Selection

- Incremental step time is dominated by read-only selection time

Problem

Given unsorted array of size n and integer k, find the kth value in sorted order without modifying array.

Theorem (Munro & Raman, 1996)

The read-only selection problem can be solved in $\mathcal{O}(n^{1+\epsilon})$ time using $\mathcal{O}(1)$ words of extra space.

Read-Only Selection

- Incremental step time is dominated by read-only selection time

Problem

Given unsorted array of size n and integer k, find the kth value in sorted order without modifying array.

Theorem (Munro & Raman, 1996)

The read-only selection problem can be solved in $\mathcal{O}(n^{1+\epsilon})$ time using $\mathcal{O}(1)$ words of extra space.

Theorem (Chan 2010)

Any algorithm for the read-only selection problem using $\mathcal{O}(1)$ words of extra space requires $\Omega(n \log \log n)$ time.

Analysis

- Let $t_S(n)$ be the read-only selection time using constant extra space.

Theorem

Given the Burrows–Wheeler transform of a text of length n the text can be recovered in-place in $\mathcal{O}(n \cdot t_S(n))$ time using $\mathcal{O}(1)$ words of extra space.

Corollary

Given the Burrows–Wheeler transform of a text of length n over a general alphabet, the text can be recovered in-place in $\mathcal{O}(n^{2+\epsilon})$ time using $\mathcal{O}(1)$ words of extra space.

Analysis

- Let $t_S(n)$ be the read-only selection time using constant extra space.

Theorem

Given the Burrows–Wheeler transform of a text of length n the text can be recovered in-place in $\mathcal{O}(n \cdot t_S(n))$ time using $\mathcal{O}(1)$ words of extra space.

Corollary

Given the Burrows–Wheeler transform of a text of length n over a general alphabet, the text can be recovered in-place in $\mathcal{O}(n^{2+\epsilon})$ time using $\mathcal{O}(1)$ words of extra space.

Corollary

Given the Burrows–Wheeler transform of a text of length n over an alphabet of constant size, the text can be recovered in-place in $\mathcal{O}(n^2)$ time using $\mathcal{O}(1)$ words of extra space.

Open Problems on In-Place BWT

In-place BWT in $\mathcal{O}(n^2)$ time and inverse BWT in $\mathcal{O}(n^{2+\epsilon})$ time.

- Can the time complexities be improved?
  - In the general case?
  - Special cases (such as small alphabet)?
- Lower bounds?
- Time-space tradeoffs: What can we do if given more extra space?

Thank You!