CS481/CS583: Bioinformatics Algorithms

Can Alkan
EA509
calkan@cs.bilkent.edu.tr

http://www.cs.bilkent.edu.tr/~calkan/teaching/cs481/

Burrows-Wheeler Transformation

- Originally developed for data compression
- Reordering text -> Better locality = better compression
  - Used in bzip2
- Additional data structures for sequence search
  - Ferragina-Manzini index
  - “Summarized” suffix array

Burrows-Wheeler Transformation

1. Append to the input string a special char, $, smaller than all alphabet.

mississippi$

Burrows-Wheeler Transformation (cnt'd)

2. Generate all rotations.

m	i	s	s	i	s	s	i	p	p	i	$
i	s	s	i	s	s	i	p	p	i	$	m
s	s	i	s	s	i	p	p	i	$	m	i
s	i	s	s	i	p	p	i	$	m	i	s
i	s	s	i	p	p	i	$	m	i	s	s
s	s	i	p	p	i	$	m	i	s	s	i
s	i	p	p	i	$	m	i	s	s	i	s
i	p	p	i	$	m	i	s	s	i	s	s
p	p	i	$	m	i	s	s	i	s	s	i
p	i	$	m	i	s	s	i	s	s	i	p
i	$	m	i	s	s	i	s	s	i	p	p
$	m	i	s	s	i	s	s	i	p	p	i

Burrows-Wheeler Transformation (cnt'd)

3. Sort rotations according to the alphabetical order.

$	m	i	s	s	i	s	s	i	p	p	i
i	$	m	i	s	s	i	s	s	i	p	p
i	p	p	i	$	m	i	s	s	i	s	s
i	s	s	i	p	p	i	$	m	i	s	s
i	s	s	i	s	s	i	p	p	i	$	m
m	i	s	s	i	s	s	i	p	p	i	$
p	i	$	m	i	s	s	i	s	s	i	p
p	p	i	$	m	i	s	s	i	s	s	i
s	i	p	p	i	$	m	i	s	s	i	s
s	i	s	s	i	p	p	i	$	m	i	s
s	s	i	p	p	i	$	m	i	s	s	i
s	s	i	s	s	i	p	p	i	$	m	i

Burrows-Wheeler Transformation (cnt'd)

4. Output the last column.

$	m	i	s	s	i	s	s	i	p	p	i
i	$	m	i	s	s	i	s	s	i	p	p
i	p	p	i	$	m	i	s	s	i	s	s
i	s	s	i	p	p	i	$	m	i	s	s
i	s	s	i	s	s	i	p	p	i	$	m
m	i	s	s	i	s	s	i	p	p	i	$
p	i	$	m	i	s	s	i	s	s	i	p
p	p	i	$	m	i	s	s	i	s	s	i
s	i	p	p	i	$	m	i	s	s	i	s
s	i	s	s	i	p	p	i	$	m	i	s
s	s	i	p	p	i	$	m	i	s	s	i
s	s	i	s	s	i	p	p	i	$	m	i

Burrows-Wheeler Transformation (cnt'd)

mississippi$

ipssm$pissii

BWT – alternative construction

T = a b a a b a

BWT	Suffix Array	
$ a b a a b a	6	$
a $ a b a a b	5	a $
a a b a $ a b	2	a a b a $
a b a $ a b a	3	a b a $
a b a a b a $	0	a b a a b a $
b a $ a b a a	4	b a $
b a a b a $ a	1	b a a b a $

$$BWT[i] = \begin{cases} T[SA[i] - 1], & \text{if } SA[i]>0 \\ \$, & \text{if } SA[i]=0 \end{cases}$$

BWT = characters just to the left of characters in SA

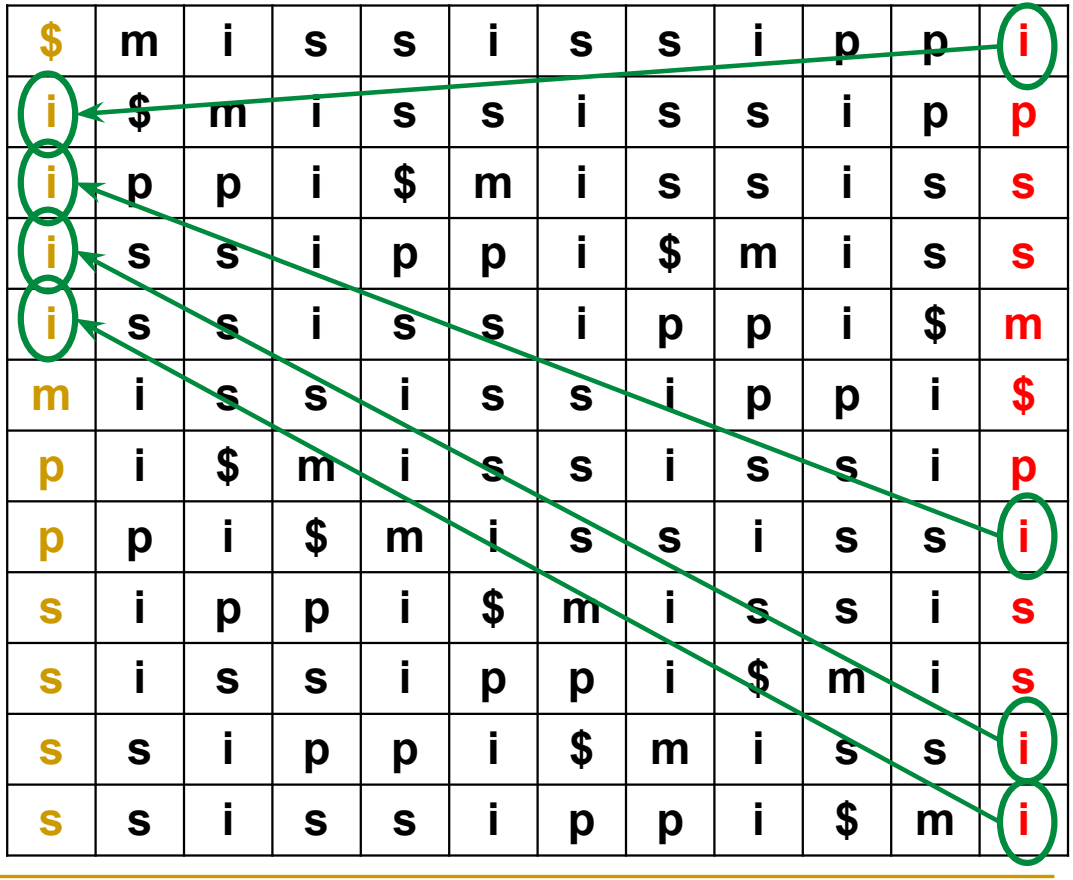
L to F map

First column: F

Last column: L

Let's make an L to F map.

Observation: The n^{th} i in L is the n^{th} i in F.



L to F map

Store/compute a two dimensional Occ(*j*,'c') table of the number of occurrences of char 'c' up to position *j* (inclusive).

and one dimensional Cnt('c') and Rank('c') tables

	$	i	m	p	s
i	0	1	0	0	0
p	0	1	0	1	0
s	0	1	0	1	1
s	0	1	0	1	2
m	0	1	1	1	2
$	1	1	1	1	2
p	1	1	1	2	2
i	1	2	1	2	2
s	1	2	1	2	3
s	1	2	1	2	4
i	1	3	1	2	4
i	1	4	1	2	4

Occ(*j*,'c')

Cnt('c')

$	i	m	p	s
1	4	1	2	4

Rank('c')

$	i	m	p	s
12	2	1	9	3

L to F map

[Cnt('$') +
Cnt('i') +
Cnt('m') +
Cnt('p') = 8]
+ [Occ(9, 's')= 3]
= 11

before 's'

's' section

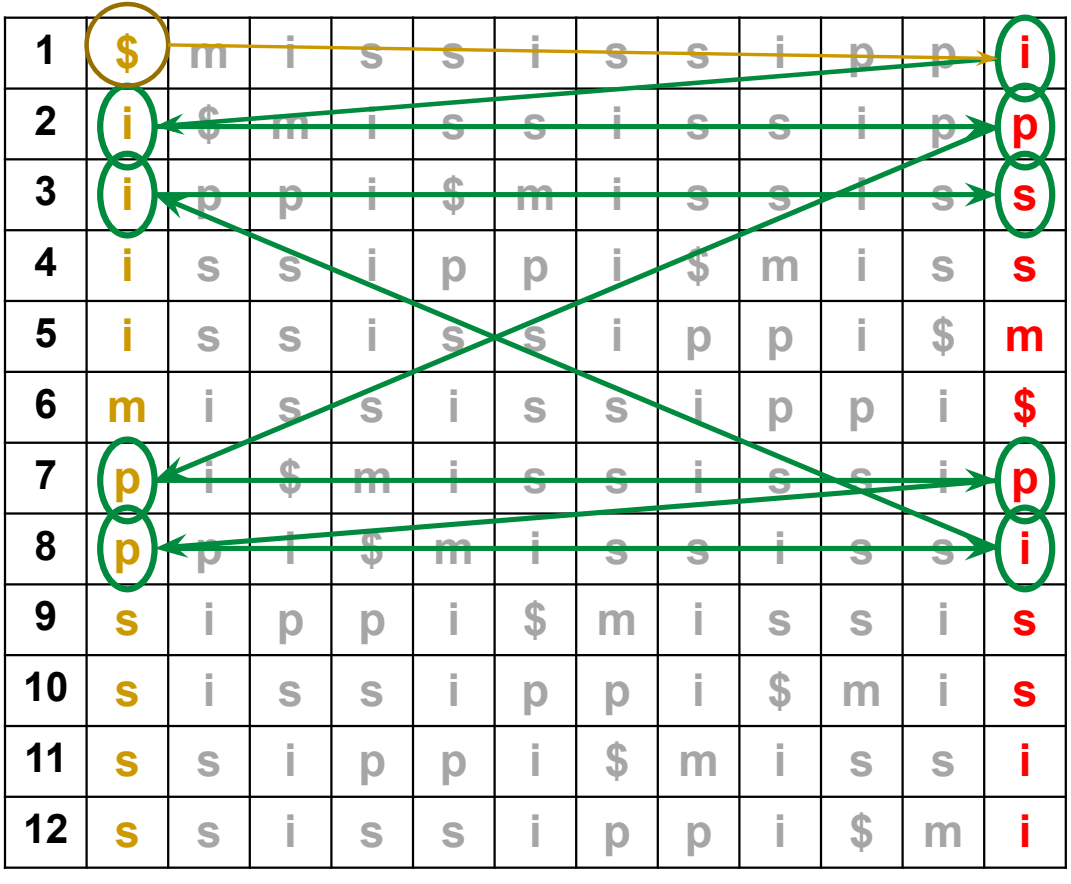
1	$	m	i	s	s	i	s	s	i	p	p	i
2	i	$	m	i	s	s	i	s	s	i	p	p
3	i	p	p	i	$	m	i	s	s	i	s	s
4	i	s	s	i	p	p	i	$	m	i	s	s
5	i	s	s	i	s	s	i	p	p	i	$	m
6	m	i	s	s	i	s	s	i	p	p	i	$
7	p	i	$	m	i	s	s	i	s	s	i	p
8	p	p	i	$	m	i	s	s	i	s	s	i
9	s	i	p	p	i	$	m	i	s	s	i	s
10	s	i	s	s	i	p	p	i	$	m	i	s
11	s	s	i	p	p	i	$	m	i	s	s	i
12	s	s	i	s	s	i	p	p	i	$	m	i

Cnt('c')

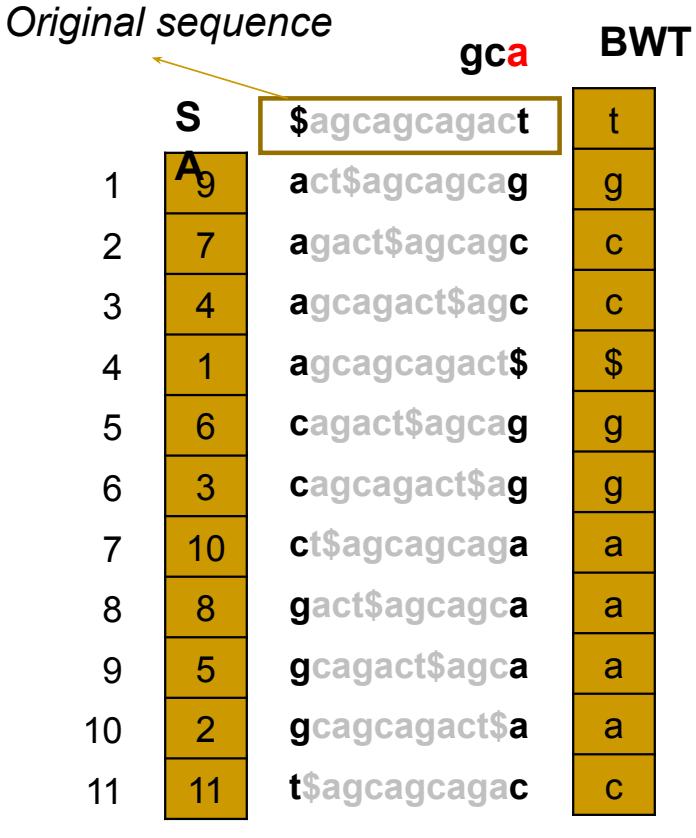
$	i	m	p	s
1	4	1	2	4

L to F map

(1) i
(2) p
(7) p
(8) i
(3) s
(9) s
(11) i
(4) s
(10) s
(12) i
(5) m
(6) $



Search with BWT-FM: L to F map



Search with BWT-FM: FM-index

Auxiliary data structures for efficient pattern matching: how to find the corresponding chars in the first column efficiently, in terms of both time and space.

	a	c	g	t
rank	1	5	8	11

Original sequence

	SA		BWT
		$agcagcagact	t
1	9	act$agcagcag	g
2	7	agact$agcagc	c
3	4	agcagact$agc	c
4	1	agcagcagact$	$
5	6	cagact$agcag	g
6	3	cagcagact$ag	g
7	10	ct$agcagcaga	a
8	8	gact$agcagca	a
9	5	gcagact$agca	a
10	2	gcagcagact$a	a
11	11	t$agcagcagac	c

a	c	g	t
0	0	0	1
0	0	1	1
0	1	1	1
0	2	1	1
0	2	1	1
0	2	2	1
0	2	3	1
1	2	3	1
2	2	3	1
3	2	3	1
4	2	3	1
4	3	3	1

FM indices

Search with BWT-FM: FM-index

Auxiliary data structures for efficient pattern matching: how to find the corresponding chars in the first column efficiently, in terms of both time and space.

	a	c	g	t
rank	1	5	8	11

Original sequence

gca

	SA		BWT
		$agcagcagact	t
1	9	act$agcagcag	g
2	7	agact$agcagc	c
3	4	agcagact$agc	c
4	1	agcagcagact$	$
5	6	cagact$agcag	g
6	3	cagcagact$ag	g
7	10	ct$agcagcaga	a
8	8	gact$agcagca	a
9	5	gcagact$agca	a
10	2	gcagcagact$a	a
11	11	t$agcagcagac	c

a	c	g	t
0	0	0	1
0	0	1	1
0	1	1	1
0	2	1	1
0	2	1	1
0	2	2	1
0	2	3	1
1	2	3	1
2	2	3	1
3	2	3	1
4	2	3	1
4	3	3	1

FM indices

Next block:
From 1 + 0 = 1
to 1 + (4-1) = 4

Search with BWT-FM: FM-index

Auxiliary data structures for efficient pattern matching: how to find the corresponding chars in the first column efficiently, in terms of both time and space.

	a	c	g	T
rank	1	5	8	11

Original sequence

gca

	SA		BWT
1	9	$agcagcagact	t
2	7	act$agcagcag	g
3	4	agact$agcagc	c
4	1	agcagcagact$	$
5	6	cagact$agcag	g
6	3	cagcagact$ag	g
7	10	ct$agcagcaga	a
8	8	gact$agcagca	a
9	5	gcagact$agca	a
10	2	gcagcagact$a	a
11	11	t$agcagcagac	c

a	c	g	t
0	0	0	1
0	0	1	1
0	1	1	1
0	2	1	1
0	2	1	1
0	2	2	1
0	2	3	1
1	2	3	1
2	2	3	1
3	2	3	1
4	2	3	1
4	3	3	1

FM indices

Next block: From 5 + 0 = 5 to 5 + (2-1) = 6

Search with BWT-FM: FM-index

Auxiliary data structures for efficient pattern matching: how to find the corresponding chars in the first column efficiently, in terms of both time and space.

	a	c	g	T
rank	1	5	8	11

Original sequence

gca

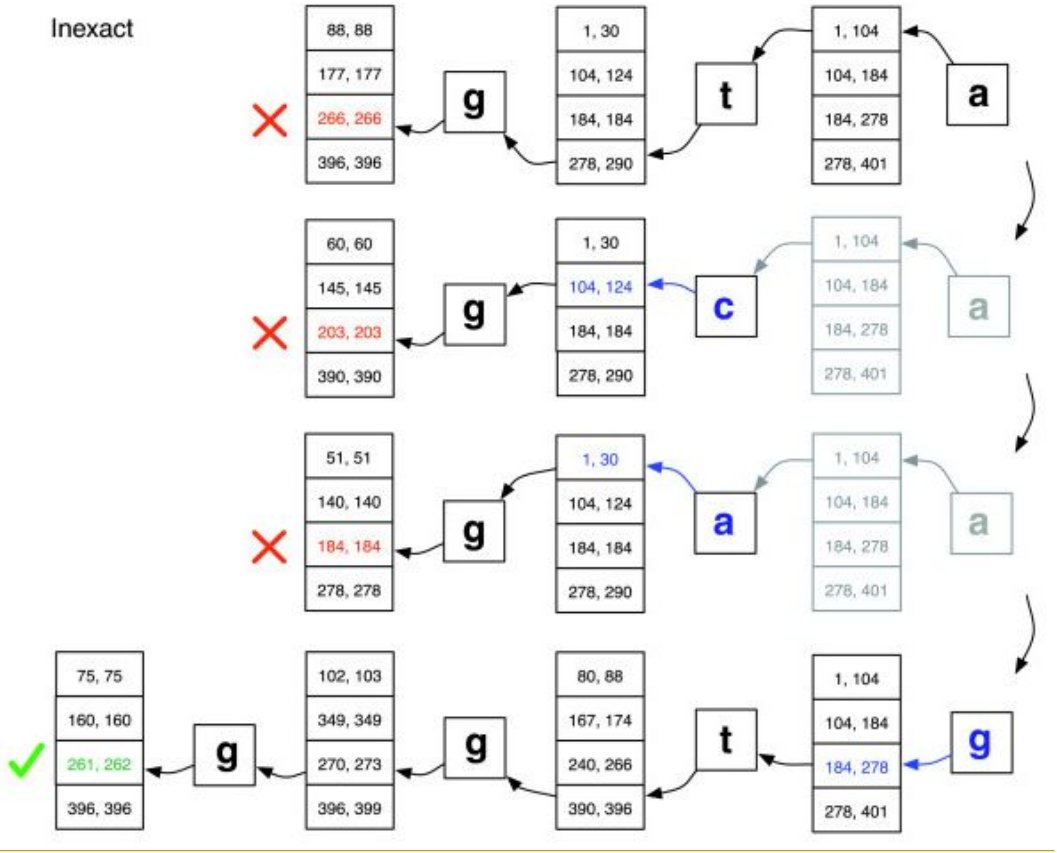
	SA		BWT
1	9	$agcagcagact	t
2	7	act$agcagcag	g
3	4	agact$agcagc	c
4	1	agcagact$agc	c
5	6	agcagcagact$	$
6	3	cagact$agcag	g
7	10	cagcagact$ag	g
8	8	ct$agcagcaga	a
9	5	gact$agcagca	a
10	2	gcagact$agca	a
11	11	gcagcagact$a	a
		t$agcagcagac	c

a	c	g	t
0	0	0	1
0	0	1	1
0	1	1	1
0	2	1	1
0	2	1	1
0	2	2	1
0	2	3	1
1	2	3	1
2	2	3	1
3	2	3	1
4	2	3	1
4	3	3	1

FM indices

Next block:
From 8 - 1 = 9
to 8 + (3-1) = 10

Inexact match



Videos

- BWT
  - https://www.youtube.com/watch?v=4n7NPk5lwbI
- FM-index
  - https://www.youtube.com/watch?v=kvVGj5V65io