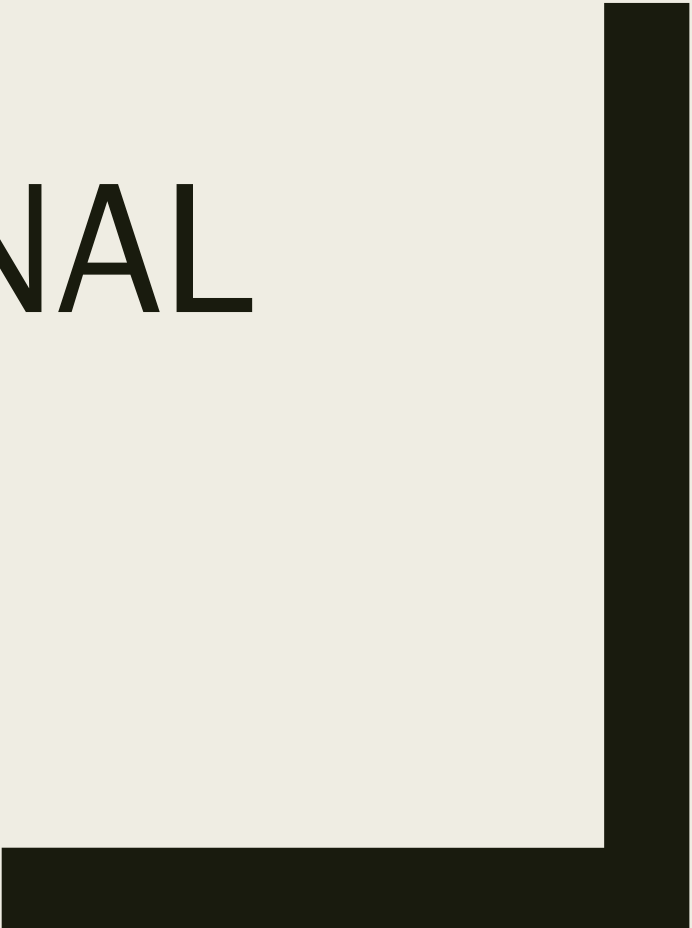


CS 364  
COMPUTATIONAL  
BIOLOGY

Sara Mathieson  
Haverford College



# Outline

- Recap goals of the BWT
- FM-Index data structure
- Using the BWT for read mapping

BWT so far...

# Read mapping

Long reference genome



**...AGTATCTGTCTTTGATTCCTGCCTCATCCTATTATTTATCGCACCTACGTTCAATATT...**

CTTTGATTCCTGCC

TTATTTATCGCACCTAC

CTGCCTCATCCTA



Many short reads to align

Reference doesn't change so lets process it once (slow) but then hopefully it will be fast to map each read.

# Bowtie and BWA (posted reading)

- First practical read aligners to use the **Burrows-Wheeler Transform**

## Fast and accurate short read alignment with Burrows–Wheeler transform

Heng Li, Richard Durbin  [Author Notes](#)

*Bioinformatics*, Volume 25, Issue 14, **15 July 2009**, Pages 1754–1760,

<https://doi.org/10.1093/bioinformatics/btp324>

**Published:** 18 May 2009 **Article history** ▼

BWA



Bowtie



## Ultrafast and memory-efficient alignment of short DNA sequences to the human genome

[Ben Langmead](#) , [Cole Trapnell](#), [Mihai Pop](#) and [Steven L Salzberg](#)

*Genome Biology* 2009 10:R25

<https://doi.org/10.1186/gb-2009-10-3-r25> | © Langmead et al.; licensee BioMed Central Ltd. 2009

Received: 21 October 2008 | Accepted: 4 March 2009 | Published: **4 March 2009**

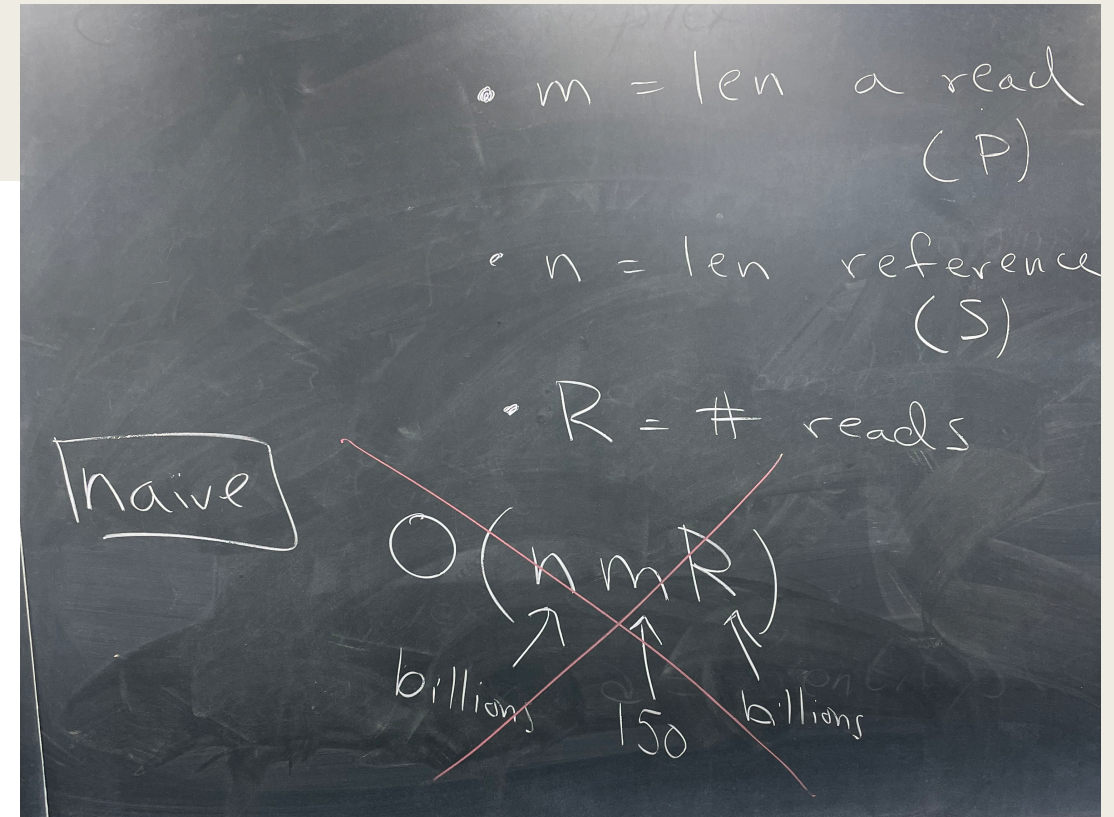
# Comparison of Bowtie and BWA

**Table 2.**

Evaluation on real data

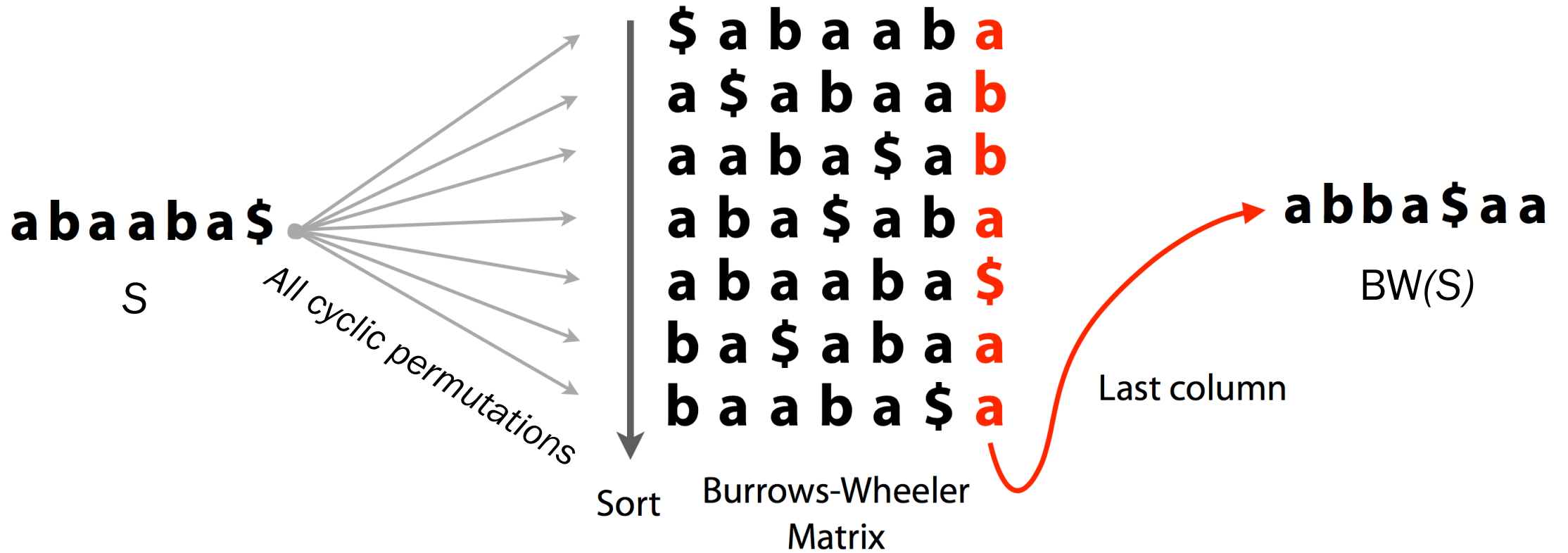
Program	Time (h)	Conf (%)	Paired (%)
Bowtie	5.2	84.4	96.3
BWA	4.0	88.9	98.8
MAQ	94.9	86.1	98.7
SOAP2	3.4	88.3	97.5

The 12.2 million read pairs were mapped to the human genome. CPU time in hours on a single core of a 2.5 GHz Xeon E5420 processor (Time), percent confidently mapped reads (Conf) and percent confident mappings with the mates mapped in the correct orientation and within 300 bp (Paired), are shown in the table.



# BWT

Reversible permutation of the characters of a string, used originally for compression



# Example from the original BWT paper

final char ( $L$ )	sorted rotations
a	n to decompress. It achieves compression
o	n to perform only comparisons to a depth
o	n transformation} This section describes
o	n transformation} We use the example and
o	n treats the right-hand side as the most
a	n tree for each 16 kbyte input block, enc
a	n tree in the output stream, then encodes
i	n turn, set $L[i]$ to be the
i	n turn, set $R[i]$ to the
o	n unusual data. Like the algorithm of Man
a	n use a single set of probabilities table
e	n using the positions of the suffixes in
i	n value at a given point in the vector $R$
e	n we present modifications that improve t
e	n when the block size is quite large. Ho
i	n which codes that have not been seen in
i	n with $sch$ appear in the {\em same order
i	n with $sch$ . In our exam
o	n with Huffman or arithmetic coding. Bri
o	n with figures given by Bell~\cite{bell}.

Figure 1: Example of sorted rotations. Twenty consecutive rotations from the sorted list of rotations of a version of this paper are shown, together with the final character of each rotation.

Burrows M, Wheeler DJ: "A block sorting lossless data compression algorithm." *Digital Equipment Corporation, Palo Alto, CA 1994, Technical Report 124; 1994*



# Compression with BWT

**Text:** I stuffed a shirt or two into my old carpet-bag, tucked it under my arm, and started for Cape Horn and the Pacific. Quitting the good city of old Manhatto, I duly arrived in New Bedford. It was a Saturday night in December. Much was I disappointed upon learning that the little packet for Nantucket had already sailed, and that no way of reaching that place would offer, till the following Monday. As most young candidates for the pains and penalties of whaling stop at this same New Bedford, thence to embark on their voyage, it may as well be related that I, for one, had no idea of so doing. For my mind was made up to sail in no other than a Nantucket craft, because there was a fine, boisterous something about everything connected with that famous old island, which amazingly pleased me. Besides though New Bedford has of late been gradually monopolising the business of whaling, and though in this matter poor old Nantucket is now much behind her, yet Nantucket was her great original—the Tyre of this Carthage

**BWT:**  
e.www.rsn.es,t.dg.ardthene.aenssdgdhs,,n,yyyp1,eh,egdgtIoIottaedt,senrt,ds  
dotdlhotd,dfnesstdyyswrroydntssyyaessdfryskrttoeedtyrfeoyafsdgIrdhdggdglgt  
rn,efntsd,ep,rtedrhtetdosff,de,tomdgteIeedrgdrtyegdrc  
e lePphmrrerbhyssphhu nn sfh lCMNNNCSb ec tCwwwww ehhehh  
lldmhScmdwdm-m m a i enaeiuuaa  
uaauuu ellealoerneleealnenneneernrirnainieeen a  
agpmhthhhhrlhbcdrcnngmdrllrbDnftvkstttlBBBbbhrw  
cephhdthfhtvtidBnykkkkkpm NNNooooo  
ffuoi dddannnnnnnnnanaauuin gccgtc twwt tttttntttttttt t  
twettcstttttgfhdstcnaat ghmfswhhlnstlozao ehhd  
o wulcrrccccclilps eoooo piaao eioalaaulgar aeea  
o a iiareoioieieiaaaaaiaouaioniiiiiiiiiiiia ro o  
iiiaaaastnwtntnto dpb pfsp M cmgptnffFoff fff Hm hhwymrbnlvou a  
r pou aoeeeeeeeeoieeegcooouye lg oraaooeaaiaaeiaaaneaaiAiaesuiuea iua  
euii eo i aeuaeeheiaraaaaaiIesfesaranctsaair - o ey l tt n  
tsaia nnnnsa idmMtttttttooQod o tooaboie eeee o t  
mlamlatamldao Tral

# Compression with BWT

- We computed the BWT, and showed that we can reconstruct  $S$  from  $BWT(S)$
- $BWT(S)$  is the same length as  $S$ , but in practice we can compress it to make it smaller e.g:

$S =$  "Tomorrow\_and\_tomorrow\_and\_tomorrow\$"

$BWT(S) =$  "w\$wwdd\_\_nnoooaattTmmrrrrrrrooo\_\_ooo"

$Comp =$  "w\$w2d2\_2n2o3a2t2Tm3r6o3\_2o3"

- In fact, BWT was originally developed as a compression algorithm.

# Reconstructing S from BWT: LF mapping

S=ACAACGT\$

	F		L
8 :	\$	ACAACGT	T
3 :	A	ACGT\$AC	C
1 :	A	CAACGT\$	\$
4 :	A	CGT\$ACA	A
2 :	C	AACGT\$A	A
5 :	C	GT\$ACAA	C
6 :	G	T\$ACAAC	C
7 :	T	\$ACAACG	G

If we have L, i.e. BWT(S), we can get F just by sorting it →

# Reconstructing S from BWT: LF mapping

S=ACAACGT\$

F	L
\$	T
A	C
A	\$
A	A
C	A
C	A
G	C
T	G

Key property 1: If  $F[i]=X$  and  $L[i]=Y$ , then the string  $YX$  must appear in  $S$

Key property 2: The order of each  $A$  (also  $C, G, T$ ) in  $F$  is the same as in  $L$

# Reconstructing S from BWT: LF mapping

S=ACAACGT\$

F	L
\$	T
A	C
A	\$
A	A
C	A
C	A
G	C
T	G

Key property 1: If  $F[i]=X$  and  $L[i]=Y$ , then the string  $YX$  must appear in  $S$

Key property 2: The order of each  $A$  (also  $C, G, T$ ) in  $F$  is the same as in  $L$

# Reconstructing S from BWT: LF mapping

Key property 2: The order of each C (also each A, G, T) in F is the same as in L

S=ACAACGT\$

F	L
\$ACAACGT	
AACGT\$A	C
ACAACGT\$	
ACGT\$ACA	
C AACGT\$A	
C GT\$ACAA	
GT\$ACAA	C
T\$ACAACG	

The Cs in the F column are sorted according to the parts of the string that follow them

The Cs in the L column are also sorted according to the parts of the string that follow them

# Reconstructing S from BWT: LF mapping

Key property 2: The order of each A (also C, G, T) in F is the same as in L

S=ACACGT\$

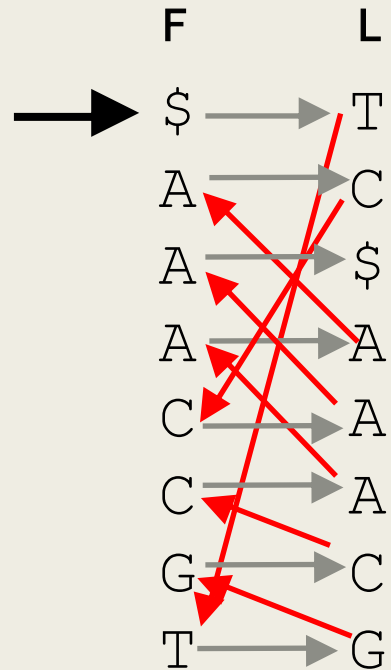
F	L
\$ACAACGT	
<b>A</b> ACGT\$AC	
<b>A</b> CAACGT\$	
<b>A</b> CGT\$AC <b>A</b>	
CAACGT\$ <b>A</b>	
CGT\$ACA <b>A</b>	
GT\$ACAAC	
T\$ACAACG	

The As in the F column are sorted according to the parts of the string that follow them

The As in the L column are also sorted according to the parts of the string that follow them

# Reconstructing S from BWT: LF mapping

\$ must be the last character, start here



Key property 1: If  $F[i]=X$  and  $L[i]=Y$ , then the string  $YX$  must appear in  $S$

Key property 2: The order of each  $A$  (also  $C, G, T$ ) in  $F$  is the same as in  $L$

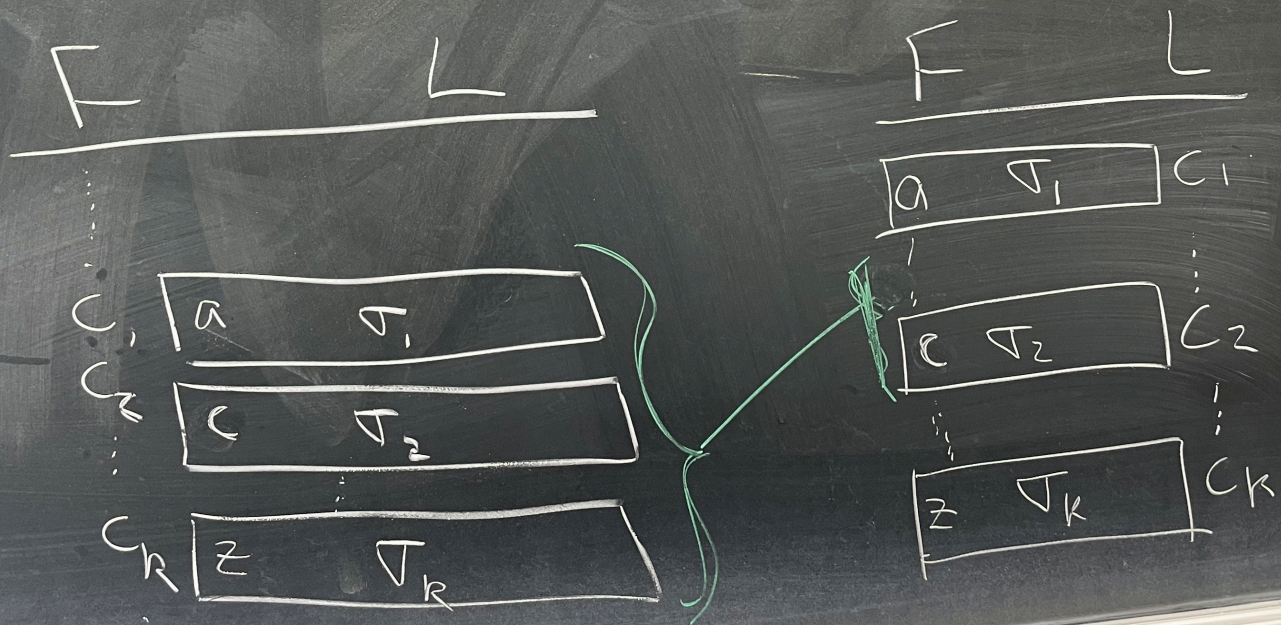
$S = \dots T \$$   
 $S = \dots GT \$$   
 $S = \dots CGT \$$   
 $S = \dots ACGT \$$   
 $S = \dots AACGT \$$   
 $S = \dots CAACGT \$$   
 $S = \dots ACAACGT \$$



claim: if  $F$  has  $k$  copies  
of char  $c: c_1, c_2, \dots, c_k$ ,  
their order is preserved in  $L$

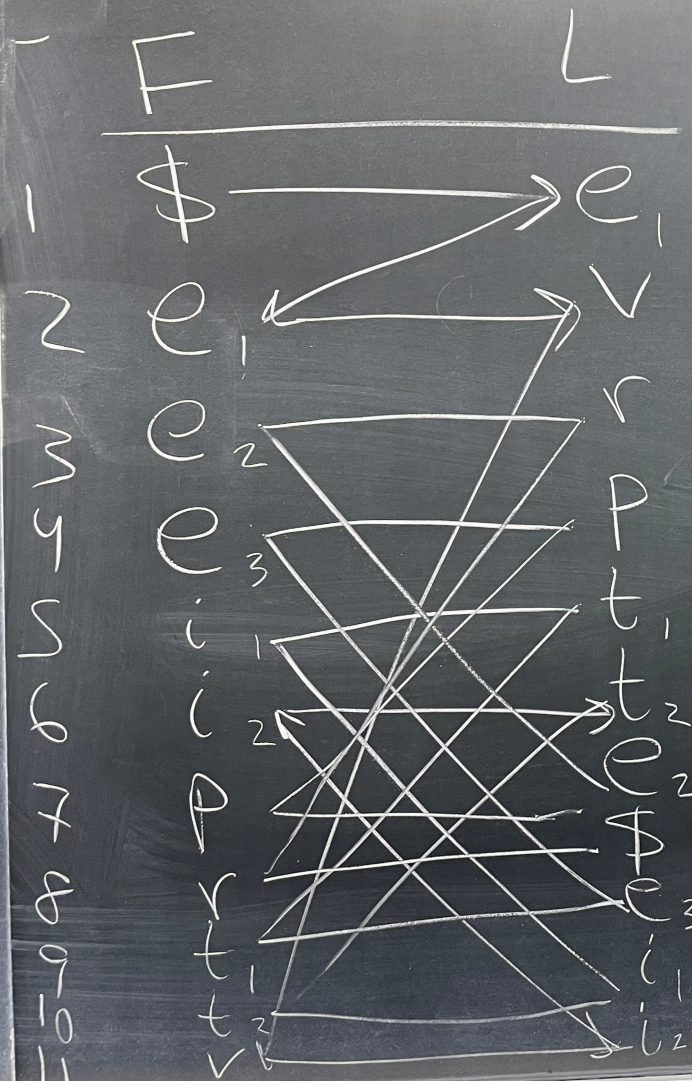
$\sigma_i = \text{substring}$

$$\sigma_1 < \sigma_2 < \dots < \sigma_k \iff \sigma_1 c_1 < \sigma_2 c_2 < \dots < \sigma_k c_k$$



Finish Handout 2

Handout 2, #2



repetitive\$

X stop

# FM-Index data structure

# FM-Index: data structure for pattern matching

- Set of auxiliary data structures computed from the BWT of a string  $S$
- The FM-Index consists of 3 parts:

# FM-Index: data structure for pattern matching

- Set of auxiliary data structures computed from the BWT of a string  $S$
- The FM-Index consists of 3 parts:
  - (a) The BWT of  $S$ , i.e. the  $L$  column of  $\pi^{\text{sorted}}(S)$
  - (b)  $M[c]$ , the first index of  $c$  in  $F$  (note that  $F$  is actually not part of the FM-Index)
  - (c)  $\text{occ}(c, i)$ , the number of times  $c$  occurs in  $L[1 \dots i]$ , inclusive


# FM-Index: data structure for pattern matching

- Set of auxiliary data structures computed from the BWT of a string  $S$
- The FM-Index consists of 3 parts:
  - (a) The BWT of  $S$ , i.e. the  $L$  column of  $\pi^{\text{sorted}}(S)$
  - (b)  $M[c]$ , the first index of  $c$  in  $F$  (note that  $F$  is actually not part of the FM-Index)
  - (c)  $\text{occ}(c, i)$ , the number of times  $c$  occurs in  $L[1 \dots i]$ , inclusive
- The suffix array  $A$  is not technically part of the FM-Index, but we will need it for the last step of finding out where pattern  $P$  occurs in the original string  $S$
- $A[i]$  is the index of  $F[i]$  in the original string

Example:  $S = \text{abaaba}\$, P = \text{aba}$   
 1 2 3 4 5 6 7

i	F	L	A	occ(\$)	occ(a)	occ(b)
1	\$ <sub>1</sub> abaab	a <sub>1</sub>				
2	a <sub>1</sub> \$abaa	b <sub>1</sub>				
3	a <sub>2</sub> aba\$a	b <sub>2</sub>				
4	a <sub>3</sub> ba\$ab	a <sub>2</sub>				
5	a <sub>4</sub> baaba	\$ <sub>1</sub>				
6	b <sub>1</sub> a\$aba	a <sub>3</sub>				
7	b <sub>2</sub> aaba\$	a <sub>4</sub>				

Step 1: compute the BWT of S





Example:  $S = \text{abaaba}\$, P = \text{aba}$   
 1 2 3 4 5 6 7

i	F	L	A	occ(\$)	occ(a)	occ(b)
1	\$ <sub>1</sub> abaab	a <sub>1</sub>	7			
2	a <sub>1</sub> \$abaa	b <sub>1</sub>	6			
3	a <sub>2</sub> aba\$a	b <sub>2</sub>	3			
4	a <sub>3</sub> ba\$ab	a <sub>2</sub>	4			
5	a <sub>4</sub> baaba	\$ <sub>1</sub>	1			
6	b <sub>1</sub> a\$aba	a <sub>3</sub>	5			
7	b <sub>2</sub> aaba\$	a <sub>4</sub>	2			

Step 2: compute the suffix array, where  $A[i] = \text{index of } F[i] \text{ in the original sequence}$

Example:  $S = \text{abaaba}\$, P = \text{aba}$   
 1 2 3 4 5 6 7

i	F	L	A	occ(\$)	occ(a)	occ(b)
1	\$ <sub>1</sub>	abaab	a <sub>1</sub>	7	0	
2	a <sub>1</sub>	\$abaa	b <sub>1</sub>	6	0	
3	a <sub>2</sub>	aba\$a	b <sub>2</sub>	3	0	
4	a <sub>3</sub>	ba\$ab	a <sub>2</sub>	4	0	
5	a <sub>4</sub>	baaba	\$ <sub>1</sub>	1	1	
6	b <sub>1</sub>	a\$aba	a <sub>3</sub>	5	1	
7	b <sub>2</sub>	aaba\$	a <sub>4</sub>	2	1	

Step 3: compute the occurrence table for each character c (# times c in L[1...i])

Example:  $S = \text{abaaba}\$, P = \text{aba}$   
 1 2 3 4 5 6 7

i	F	L	A	occ(\$)	occ(a)	occ(b)
1	\$ <sub>1</sub>	abaab	a <sub>1</sub>	7	0	1
2	a <sub>1</sub>	\$abaa	b <sub>1</sub>	6	0	1
3	a <sub>2</sub>	aba\$a	b <sub>2</sub>	3	0	1
4	a <sub>3</sub>	ba\$ab	a <sub>2</sub>	4	0	2
5	a <sub>4</sub>	baaba	\$ <sub>1</sub>	1	1	2
6	b <sub>1</sub>	a\$aba	a <sub>3</sub>	5	1	3
7	b <sub>2</sub>	aaba\$	a <sub>4</sub>	2	1	4

Step 3: compute the occurrence table for each character c (# times c in L[1...i])

Example:  $S = \text{abaaba}\$, P = \text{aba}$   
 1 2 3 4 5 6 7

i	F	L	A	occ(\$)	occ(a)	occ(b)	
1	\$ <sub>1</sub>	abaab	a <sub>1</sub>	7	0	1	0
2	a <sub>1</sub>	\$abaa	b <sub>1</sub>	6	0	1	1
3	a <sub>2</sub>	aba\$a	b <sub>2</sub>	3	0	1	2
4	a <sub>3</sub>	ba\$ab	a <sub>2</sub>	4	0	2	2
5	a <sub>4</sub>	baaba	\$ <sub>1</sub>	1	1	2	2
6	b <sub>1</sub>	a\$aba	a <sub>3</sub>	5	1	3	2
7	b <sub>2</sub>	aaba\$	a <sub>4</sub>	2	1	4	2

Step 3: compute the occurrence table for each character c (# times c in L[1...i])

Example:  $S = \text{abaaba}\$, P = \text{aba}$

1 2 3 4 5 6 7



$i$	F	L	A	occ(\$)	occ(a)	occ(b)
1	$\$$ <sub>1</sub> abaab	a <sub>1</sub>	7	0	1	0
sp(a) → 2	a <sub>1</sub> \$abaa	b <sub>1</sub>	6	0	1	1
3	a <sub>2</sub> aba\$a	b <sub>2</sub>	3	0	1	2
4	a <sub>3</sub> ba\$ab	a <sub>2</sub>	4	0	2	2
ep(a) → 5	a <sub>4</sub> baaba	$\$$ <sub>1</sub>	1	1	2	2
6	b <sub>1</sub> a\$aba	a <sub>3</sub>	5	1	3	2
7	b <sub>2</sub> aaba\$	a <sub>4</sub>	2	1	4	2

Step 4: for pattern  $P$ , start with its last char and compute the start and end points

Example:  $S = \text{abaaba}\$, P = \text{aba}$

1 2 3 4 5 6 7

i	F	L	A	occ(\$)	occ(a)	occ(b)
1	\$ <sub>1</sub> abaab	a <sub>1</sub>	7	0	1	0
2	a <sub>1</sub> \$abaa	b <sub>1</sub>	6	0	1	1
3	a <sub>2</sub> aba\$a	b <sub>2</sub>	3	0	1	2
4	a <sub>3</sub> ba\$ab	a <sub>2</sub>	4	0	2	2
5	a <sub>4</sub> baaba	\$ <sub>1</sub>	1	1	2	2
6	b <sub>1</sub> a\$aba	a <sub>3</sub>	5	1	3	2
7	b <sub>2</sub> aaba\$	a <sub>4</sub>	2	1	4	2

0 -> 2 means we must have seen b<sub>1</sub> and b<sub>2</sub> in the L column

Step 5: for each new character, find the correct number of occurrences in L

Example:  $S = \text{abaaba}\$, P = \text{aba}$

1 2 3 4 5 6 7

i	F	L	A	occ(\$)	occ(a)	occ(b)
1	\$ <sub>1</sub> abaab	a <sub>1</sub>	7	0	1	0
2	a <sub>1</sub> \$abaa	b <sub>1</sub>	6	0	1	1
3	a <sub>2</sub> aba\$a	b <sub>2</sub>	3	0	1	2
4	a <sub>3</sub> ba\$ab	a <sub>2</sub>	4	0	2	2
5	a <sub>4</sub> baaba	\$ <sub>1</sub>	1	1	2	2
6	b <sub>1</sub> a\$aba	a <sub>3</sub>	5	1	3	2
7	b <sub>2</sub> aaba\$	a <sub>4</sub>	2	1	4	2

Find where  $b_1$  and  $b_2$  are in the F column, and repeat the process

Step 5: for each new character, find the correct number of occurrences in L

Example:  $S = \text{abaaba}\$, P = \text{aba}$   
 1 2 3 4 5 6 7

i	F	L	A	occ(\$)	occ(a)	occ(b)	
1	\$ <sub>1</sub> abaab	a <sub>1</sub>	7	0	1	0	
2	a <sub>1</sub> \$abaa	b <sub>1</sub>	6	0	1	1	
3	a <sub>2</sub> aba\$a	b <sub>2</sub>	3	0	1	2	
4	a <sub>3</sub> ba\$ab	a <sub>2</sub>	4	0	2	2	
5	a <sub>4</sub> baaba	\$ <sub>1</sub>	1	1	2	2	
sp(ba) →	6	b <sub>1</sub> a\$aba	a <sub>3</sub>	5	1	3	2
ep(ba) →	7	b <sub>2</sub> aaba\$	a <sub>4</sub>	2	1	4	2

Step 5: for each new character, find the correct number of occurrences in L



Example:  $S = \text{abaaba}\$, P = \text{aba}$

1 2 3 4 5 6 7

i	F	L	A	occ(\$)	occ(a)	occ(b)
1	\$ <sub>1</sub>	abaab a <sub>1</sub>	7	0	1	0
2	a <sub>1</sub>	\$abaa b <sub>1</sub>	6	0	1	1
3	a <sub>2</sub>	aba\$a b <sub>2</sub>	3	0	1	2
4	a <sub>3</sub>	ba\$ab a <sub>2</sub>	4	0	2	2
5	a <sub>4</sub>	baaba \$ <sub>1</sub>	1	1	2	2
6	b <sub>1</sub>	a\$aba a <sub>3</sub>	5	1	3	2
7	b <sub>2</sub>	aaba\$ a <sub>4</sub>	2	1	4	2

2 -> 4 means we must have seen a<sub>3</sub> and a<sub>4</sub> in the L column

Step 5: for each new character, find the correct number of occurrences in L

Example:  $S = \text{abaaba}\$, P = \text{aba}$

1 2 3 4 5 6 7

i	F	L	A	occ(\$)	occ(a)	occ(b)
1	\$ <sub>1</sub> abaab	a <sub>1</sub>	7	0	1	0
2	a <sub>1</sub> \$abaa	b <sub>1</sub>	6	0	1	1
3	a <sub>2</sub> aba\$a	b <sub>2</sub>	3	0	1	2
4	a <sub>3</sub> ba\$ab	a <sub>2</sub>	4	0	2	2
5	a <sub>4</sub> baaba	\$ <sub>1</sub>	1	1	2	2
6	b <sub>1</sub> a\$aba	a <sub>3</sub>	5	1	3	2
7	b <sub>2</sub> aaba\$	a <sub>4</sub>	2	1	4	2

Find where a<sub>3</sub> and a<sub>4</sub> are in the F column, done since P ended

Step 5: for each new character, find the correct number of occurrences in L

Example:  $S = \text{abaaba}\$, P = \text{aba}$

1 2 3 4 5 6 7

$i$	F	L	A	occ(\$)	occ(a)	occ(b)
1	$\$$ <sub>1</sub> abaab	a <sub>1</sub>	7	0	1	0
2	a <sub>1</sub> \$abaa	b <sub>1</sub>	6	0	1	1
3	a <sub>2</sub> aba\$a	b <sub>2</sub>	3	0	1	2
$sp(aba) \rightarrow$ 4	a <sub>3</sub> ba\$ab	a <sub>2</sub>	4	0	2	2
$ep(aba) \rightarrow$ 5	a <sub>4</sub> baaba	$\$$ <sub>1</sub>	1	1	2	2
6	b <sub>1</sub> a\$aba	a <sub>3</sub>	5	1	3	2
7	b <sub>2</sub> aaba\$	a <sub>4</sub>	2	1	4	2

Note that start and end points are inclusive

Step 6: when we reach the end of  $P$ , we should have the start/end points in F

Example:  $S = \text{abaaba}\$, P = \text{aba}$

1 2 3 4 5 6 7

$i$	F	L	A	occ(\$)	occ(a)	occ(b)
1	$\$$ <sub>1</sub> abaab	a <sub>1</sub>	7	0	1	0
2	a <sub>1</sub> \$abaa	b <sub>1</sub>	6	0	1	1
3	a <sub>2</sub> aba\$a	b <sub>2</sub>	3	0	1	2
$sp(aba) \rightarrow$ 4	a <sub>3</sub> ba\$ab	a <sub>2</sub>	4	0	2	2
$ep(aba) \rightarrow$ 5	a <sub>4</sub> baaba	$\$$ <sub>1</sub>	1	1	2	2
6	b <sub>1</sub> a\$aba	a <sub>3</sub>	5	1	3	2
7	b <sub>2</sub> aaba\$	a <sub>4</sub>	2	1	4	2

Use A (suffix array) to find the original locations of  $P$  in  $S$

Step 7: we are not truly done until we find the locations in the original string!

base case

$$sp(a) = 2$$

$$ep(a) = 6 - 1 = 5$$

$$sp(ba) = 6 + occ(b, sp(a) - 1)$$
$$occ(b, 2 - 1)$$

$$= 6 + 0 = 6$$

$$ep(ba) = 6 + occ(b, ep(a) - 1)$$
$$6 + 1 = 7$$

# BWT pattern matching algorithm

Base case: find the start point (sp) and end point (ep) of the *last* character in  $P$  (inclusive, so we subtract 1 from the end point):

$$\text{sp}(c) = M[c], \quad \text{ep}(c) = M[\text{char alphabetically after } c] - 1$$

Recursion:

$$\text{sp}(c\sigma) = M[c] + \text{occ}(c, \text{sp}(\sigma) - 1)$$

$$\text{ep}(c\sigma) = M[c] + \text{occ}(c, \text{ep}(\sigma)) - 1$$

# BWT Pattern Matching Algorithm

- M table
- occ table
- A column

start  
point  
in F  
→  
end  
point

• base case: let  $c$  be last char in  $P$

$$sp[c] = M[c]$$

$$ep[c] = M[\text{next char alphabetically}] - 1$$

↑  
inclusive

• recursive call: let  $c$  be next char in  $P$

$$sp[c\sigma] = M[c] + occ(c, sp(\sigma) - 1)$$

$$ep[c\sigma] = M[c] + occ(c, ep(\sigma)) - 1$$

# Handout 3



Handout 3 example:  $S = \text{barbar}\$, P = \text{ba}$   
 1 2 3 4 5 6 7 8

i	F	L	A	occ(\$)	occ(a)	occ(b)	occ(r)
1	\$ <sub>1</sub>	barbar	a <sub>1</sub>				
2	a <sub>1</sub>	\$barba	r <sub>1</sub>				
3	a <sub>2</sub>	ra\$bar	b <sub>1</sub>				
4	a <sub>3</sub>	rbara\$	b <sub>2</sub>				
5	b <sub>1</sub>	ara\$ba	r <sub>2</sub>				
6	b <sub>2</sub>	arbara	\$ <sub>1</sub>				
7	r <sub>1</sub>	a\$barb	a <sub>2</sub>				
8	r <sub>2</sub>	bara\$b	a <sub>3</sub>				

Work with a partner!

- 1) Fill in a column for A as well
- 2) Try to come up with a formula for **sp** and **ep** in terms of **M** and **occ**

Handout 3 example:  $S = \text{barbar}\$, P = \text{ba}$   
 1 2 3 4 5 6 7 8

i	F	L	A	occ(\$)	occ(a)	occ(b)	occ(r)
1	\$ <sub>1</sub>	barbar	a <sub>1</sub>	8	0	1	0
2	a <sub>1</sub>	\$barba	r <sub>1</sub>	7	0	1	1
3	a <sub>2</sub>	ra\$bar	b <sub>1</sub>	5	0	1	1
4	a <sub>3</sub>	rbar\$a	b <sub>2</sub>	2	0	1	1
5	b <sub>1</sub>	ara\$ba	r <sub>2</sub>	4	0	1	2
6	b <sub>2</sub>	arbara	\$ <sub>1</sub>	1	1	1	2
7	r <sub>1</sub>	a\$barb	a <sub>2</sub>	6	1	2	2
8	r <sub>2</sub>	bara\$b	a <sub>3</sub>	3	1	3	2

Handout 3 example:  $S = \text{barbarar}\$, P = \text{bar}$   
 1 2 3 4 5 6 7 8

i	F	L	A	occ(\$)	occ(a)	occ(b)	occ(r)
1	\$ <sub>1</sub> barbar	a <sub>1</sub>	8	0	1	0	0
2	a <sub>1</sub> \$barba	r <sub>1</sub>	7	0	1	0	1
3	a <sub>2</sub> ra\$bar	b <sub>1</sub>	5	0	1	1	1
4	a <sub>3</sub> rbarar\$	b <sub>2</sub>	2	0	1	2	1
5	b <sub>1</sub> arar\$ba	r <sub>2</sub>	4	0	1	2	2
6	b <sub>2</sub> arbarar	\$ <sub>1</sub>	1	1	1	2	2
7	r <sub>1</sub> ar\$barb	a <sub>2</sub>	6	1	2	2	2
8	r <sub>2</sub> barar\$b	a <sub>3</sub>	3	1	3	2	2

c	M[c]
\$	1
a	2
b	5
r	7

M[c] is the first index of character c in F  
 (Store instead of F)

Handout 3 example:  $S = \text{barbar}\$, P = \text{ba}$   
 1 2 3 4 5 6 7 8

i	F	L	A	occ(\$)	occ(a)	occ(b)	occ(r)
1	\$ <sub>1</sub> barbar	a <sub>1</sub>	8	0	1	0	0
2	a <sub>1</sub> \$barba	r <sub>1</sub>	7	0	1	0	1
3	a <sub>2</sub> ra\$bar	b <sub>1</sub>	5	0	1	1	1
4	a <sub>3</sub> rbara\$	b <sub>2</sub>	2	0	1	2	1
5	b <sub>1</sub> ara\$ba	r <sub>2</sub>	4	0	1	2	2
6	b <sub>2</sub> arbara	\$ <sub>1</sub>	1	1	1	2	2
7	r <sub>1</sub> a\$barb	a <sub>2</sub>	6	1	2	2	2
8	r <sub>2</sub> bara\$b	a <sub>3</sub>	3	1	3	2	2

c	M[c]
\$	1
a	2
b	5
r	7

M[c] is the first index of character c in F (Store instead of F)

$$\text{sp}(a) = 2$$

$$\text{ep}(a) = 4$$

# Handout 3 example: $S = \text{barbara}\$, P = \text{ba}$

1 2 3 4 5 6 7 8

i	F	L	A	occ(\$)	occ(a)	occ(b)	occ(r)
1	\$ <sub>1</sub> barbar	a <sub>1</sub>	8	0	1	0	0
2	a <sub>1</sub> \$barba	r <sub>1</sub>	7	0	1	0	1
3	a <sub>2</sub> ra\$bar	b <sub>1</sub>	5	0	1	1	1
4	a <sub>3</sub> rbar\$a	b <sub>2</sub>	2	0	1	2	1
5	b <sub>1</sub> ara\$ba	r <sub>2</sub>	4	0	1	2	2
6	b <sub>2</sub> arbara	\$ <sub>1</sub>	1	1	1	2	2
7	r <sub>1</sub> a\$barb	a <sub>2</sub>	6	1	2	2	2
8	r <sub>2</sub> bara\$b	a <sub>3</sub>	3	1	3	2	2

c	M[c]
\$	1
a	2
b	5
r	7

$M[c]$  is the first index of character  $c$  in  $F$   
(Store instead of  $F$ )

$sp(ba) = M[b] + \# \text{ b's we saw right before the first a}$

$ep(ba) = M[b] + \# \text{ b's we saw up until the last a}$

# Handout 3 example: $S = \text{barbara}\$, P = \text{ba}$

1 2 3 4 5 6 7 8

i	F	L	A	occ(\$)	occ(a)	occ(b)	occ(r)
1	\$ <sub>1</sub> barbar	a <sub>1</sub>	8	0	1	0	0
2	a <sub>1</sub> \$barba	r <sub>1</sub>	7	0	1	0	1
3	a <sub>2</sub> ra\$bar	b <sub>1</sub>	5	0	1	1	1
4	a <sub>3</sub> rbar\$a	b <sub>2</sub>	2	0	1	2	1
5	b <sub>1</sub> ara\$ba	r <sub>2</sub>	4	0	1	2	2
6	b <sub>2</sub> arbara	\$ <sub>1</sub>	1	1	1	2	2
7	r <sub>1</sub> a\$barb	a <sub>2</sub>	6	1	2	2	2
8	r <sub>2</sub> bara\$b	a <sub>3</sub>	3	1	3	2	2

c	M[c]
\$	1
a	2
b	5
r	7

M[c] is the first index of character c in F (Store instead of F)

$$\text{sp}(\text{ba}) = 5 + 0$$

$$\text{ep}(\text{ba}) = 5 + 2 - 1 \text{ (subtract 1 since we are being inclusive)}$$

Handout 3 example:  $S = \text{barbar}\$, P = \text{ba}$   
 1 2 3 4 5 6 7 8

i	F	L	A	occ(\$)	occ(a)	occ(b)	occ(r)
1	\$ <sub>1</sub> barbar	a <sub>1</sub>	8	0	1	0	0
2	a <sub>1</sub> \$barba	r <sub>1</sub>	7	0	1	0	1
3	a <sub>2</sub> ra\$bar	b <sub>1</sub>	5	0	1	1	1
4	a <sub>3</sub> rbara\$	b <sub>2</sub>	2	0	1	2	1
5	b <sub>1</sub> ara\$ba	r <sub>2</sub>	4	0	1	2	2
6	b <sub>2</sub> arbara	\$ <sub>1</sub>	1	1	1	2	2
7	r <sub>1</sub> a\$barb	a <sub>2</sub>	6	1	2	2	2
8	r <sub>2</sub> bara\$b	a <sub>3</sub>	3	1	3	2	2

c	M[c]
\$	1
a	2
b	5
r	7

M[c] is the first index of character c in F (Store instead of F)

$$\text{sp}(\text{ba}) = 5$$

$$\text{ep}(\text{ba}) = 6$$

Handout 3 example:  $S = \text{barbarar}\$, P = \text{ba}$   
 1 2 3 4 5 6 7 8

i	F	L	A	occ(\$)	occ(a)	occ(b)	occ(r)
1	\$ <sub>1</sub>	barbar	a <sub>1</sub>	8	0	1	0
2	a <sub>1</sub>	\$barba	r <sub>1</sub>	7	0	1	0
3	a <sub>2</sub>	ra\$bar	b <sub>1</sub>	5	0	1	1
4	a <sub>3</sub>	rbarar\$	b <sub>2</sub>	2	0	1	2
5	b <sub>1</sub>	ara\$ba	r <sub>2</sub>	4	0	1	2
6	b <sub>2</sub>	arbarar	\$ <sub>1</sub>	1	1	1	2
7	r <sub>1</sub>	a\$barb	a <sub>2</sub>	6	1	2	2
8	r <sub>2</sub>	barar\$b	a <sub>3</sub>	3	1	3	2

c	M[c]
\$	1
a	2
b	5
r	7

$sp(ba) = 5$   
 $ep(ba) = 6$

Use A to find locations  
 in original string



$$sp(ba) = M[b] + \text{occ}(b, \overset{2-1}{\underbrace{sp(a)-1}})$$

$$= 5 + 0 = 5$$

$$ep(ba) = M[b] + \text{occ}(b, \underbrace{ep(a)}_{-1})$$

$$= 5 + 1 = 6$$

runtime  $\Rightarrow O(m)$

F L  $P=ba$   
occ(b)

a	z	3
a	b <sub>4</sub>	4
a	m	4
a	b <sub>5</sub>	5
a	a	5

- 11 b<sub>1</sub>
- 12 b<sub>2</sub>
- 13 b<sub>3</sub>
- 14 b<sub>4</sub>
- 15 b<sub>5</sub>
- 16 b<sub>6</sub>

# Pattern matching with BWT

- Setup time  $O(N)$
- Search time  $O(M)$
- Storage space  $O(N)$ 
  - $O(1)$  to store  $F$  (i.e.  $M$ )
  - $O(N)$  to store  $L$  (i.e.  $BWT(S)$ )
  - $O(N)$  to store  $A$
  - $O(N|\Sigma|)$  to store  $OCC$  ("checkpointing" extension allows you to store only part of  $OCC$ , without increasing complexity).
- Inexact matching can be implemented in a similar way to inexact matching with little extra cost (as long as few mismatches)

# Summary

Algorithm	Setup time	Lookup time	Storage space
Boyer-Moore	$O(M)$	$O(N)$	$O(M)$
k-mer hash table	$O(N)$	$O(M)$	$O(N)$
BWT/FM-index	$O(N)$	$O(M)$	$O(N)$

But, in practice, for the read mapping problem, BWT approaches have turned out to be the most efficient. Almost all sequence data is processed with a program called *bwa* which uses BWT to map.

# Brief history of BWT and read mapping application

- 1994, BWT introduced (as a compression algorithm)
  - Burrows, M. and Wheeler, D.J. (1994) A block-sorting lossless data compression algorithm. *Technical report 124*, Palo Alto, CA, Digital Equipment Corporation.
- 2000, FM-index for fast searching
  - Ferragina, P. and Manzini, G. (2000) Opportunistic data structures with applications. In *Proceedings of the 41st Symposium on Foundations of Computer Science (FOCS 2000)*, IEEE Computer Society, pp. 390–398.
- 2008, *BWT-SW* for sequence alignment
  - Tam, C. K. Wong, S. M. Yiu (2008) Compressed indexing and local alignment of DNA, *Bioinformatics* 24
- 2009, *Bowtie* for short read alignment (~19,000 citations to date)
  - Langmead, B. Trapnell, C. Pop, M. Salzberg, S. Ultrafast and memory-efficient alignment of short DNA sequences to the human genome. *Genome Biology* 10:R25
- 2009, *bwa* (~46,000 citations in 2024)
  - Li, H. and Durbin, R. Fast and accurate short read alignment with Burrows–Wheeler transform *Bioinformatics* 25: 1754–1760