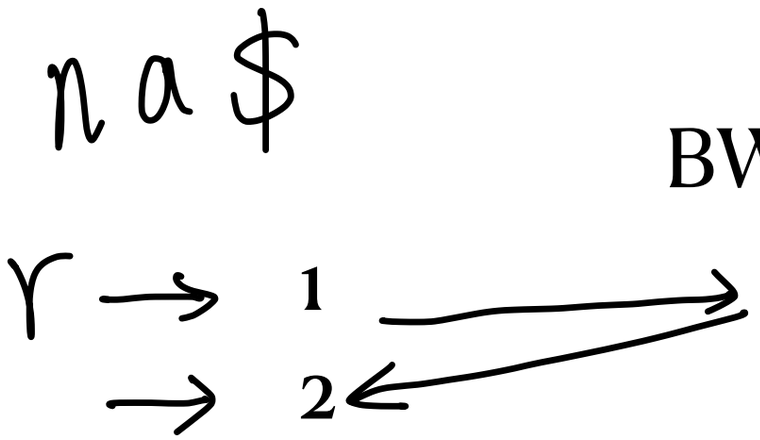
CSE 566 Spring 2023

Query with BWT, FM-index

Instructor: Mingfu Shao

Recover S from BWT, Revisited



	BWT	**rank**
1	a	1
2	n	1
3	n	2
4	b	1
5	$	1
6	a	2
7	a	3

rank[j]: the number of occurrences of letter L[j] in L[1...j]

loc

$	a	b	n
1	2	5	6

loc[c]: the row where c first appears in F

Pseudo-Code of Recovering S

```
r = 1
S = "$"
LOOP
        c <- BWT[r]
        S <- cS
        r <- loc[c] + rank[r] - 1
UNTIL c = $
```

Searching

- Question: if a query q is a substring of S #occurences of q
- Input: BWT of S, q
- Desired running time: O(|q|)

Rank Matrix

BWT(S)	**rank**			
	$	**a**	**b**	**n**
a	0	1	0	0
n	0	1	0	1
n	0	1	0	2
b	0	1	1	2
$	1	1	1	2
a	1	2	1	2
a	1	3	1	2

- rank[j, c]: the number of occurrences of c in L[1...j]
- Can fetch rank[j, c] in constant time.
- Time/Space to construct: $O(|\Sigma|n)$

Algorithm for Searching q

c=a

q=bana

	F	BWT (L)	rank $	rank a	rank b	rank n
	1	a	0	1	0	0
$r_1 \rightarrow$	2	n	0	1	0	1
	3	n	0	1	0	2
$r_2 \rightarrow$	4	b	0	1	1	2
	5	$	1	1	1	2
	6	a	1	2	1	2
	7	a	1	3	1	2

loc

$	a	b	n
1	2	5	6

- Input: BWT of S, rank, loc, q
- Process q from right to left
- Maintain range [r1, r2]
- Invariant: q[k...|q|] appears (r2 - r1 + 1) times in S, for every k.

Algorithm for Searching q

c = n

q=bana

	BWT	rank			
		$	a	b	n
1	a	0	1	0	0
$r_1 \rightarrow$ 2	n	0	1	0	1
3	n	0	1	0	2
$r_2 \rightarrow$ 4	b	0	1	1	2
5	$	1	1	1	2
$r_1 \rightarrow$ 6	a	1	2	1	2
$r_2 \rightarrow$ 7	a	1	3	1	2

loc

$	a	b	n
1	2	5	6

```
p1 <- rank[r1 - 1, c] + 1
p2 <- rank[r2, c]
```

```
r1 <- loc[c] + p1 - 1
r2 <- loc[c] + p2 - 1
```

Algorithm for Searching q

c = a

q=bana

	BWT	**rank**			
		$	a	b	n
1	a	0	1	0	0
2	n	0	1	0	1
$\gamma_1 \rightarrow$ 3	n	0	1	0	2
$\gamma_2 \rightarrow$ 4	b	0	1	1	2
5	$	1	1	1	2
$\gamma_1 \rightarrow$ 6	a	1	2	1	2
$\gamma_2 \rightarrow$ 7	a	1	3	1	2

loc

$	a	b	n
1	2	5	6

```
p1 <- rank[r1 - 1, c] + 1
p2 <- rank[r2, c]
```

1 +1 =2

= 3

```
r1 <- loc[c] + p1 - 1
r2 <- loc[c] + p2 - 1
```

2 +2 -1 =3

=4

2 + 3 -1

Algorithm for Searching q

c = b

q=bana

	BWT	rank				
		$	a	b	n	
1	a	0	1	0	0	
2	n	0	1	0	1	
$r_1 \rightarrow$ 3	n	0	1	0	2	
$r_2 \rightarrow$ 4	b	0	1	1	2	
$r_1, r_2 \rightarrow$ 5	$	1	1	1	2	
6	a	1	2	1	2	
7	a	1	3	1	2	

0 + 1 = 1

```
p1 <- rank[r1 - 1, c] + 1
p2 <- rank[r2, c]
```

p2 = 1

5 + 1 - 1 = 5

```
r1 <- loc[c] + p1 - 1
r2 <- loc[c] + p2 - 1
```

5 + 1 - 1 = 5

loc	$	a	b	n
	1	2	5	6

Q: Locations of q in S?

Summary of BWT

- Tend to group identical letters together -> compression
- Can be constructed in linear-time.
- Can be recovered in linear-time.
- Substring can be queried in linear-time (w/o recovering)
- Critical applications in bioinformatics (such as BWA, Bowtie, etc) together with FM-index [P. Ferragina and G. Manzini, 2005].

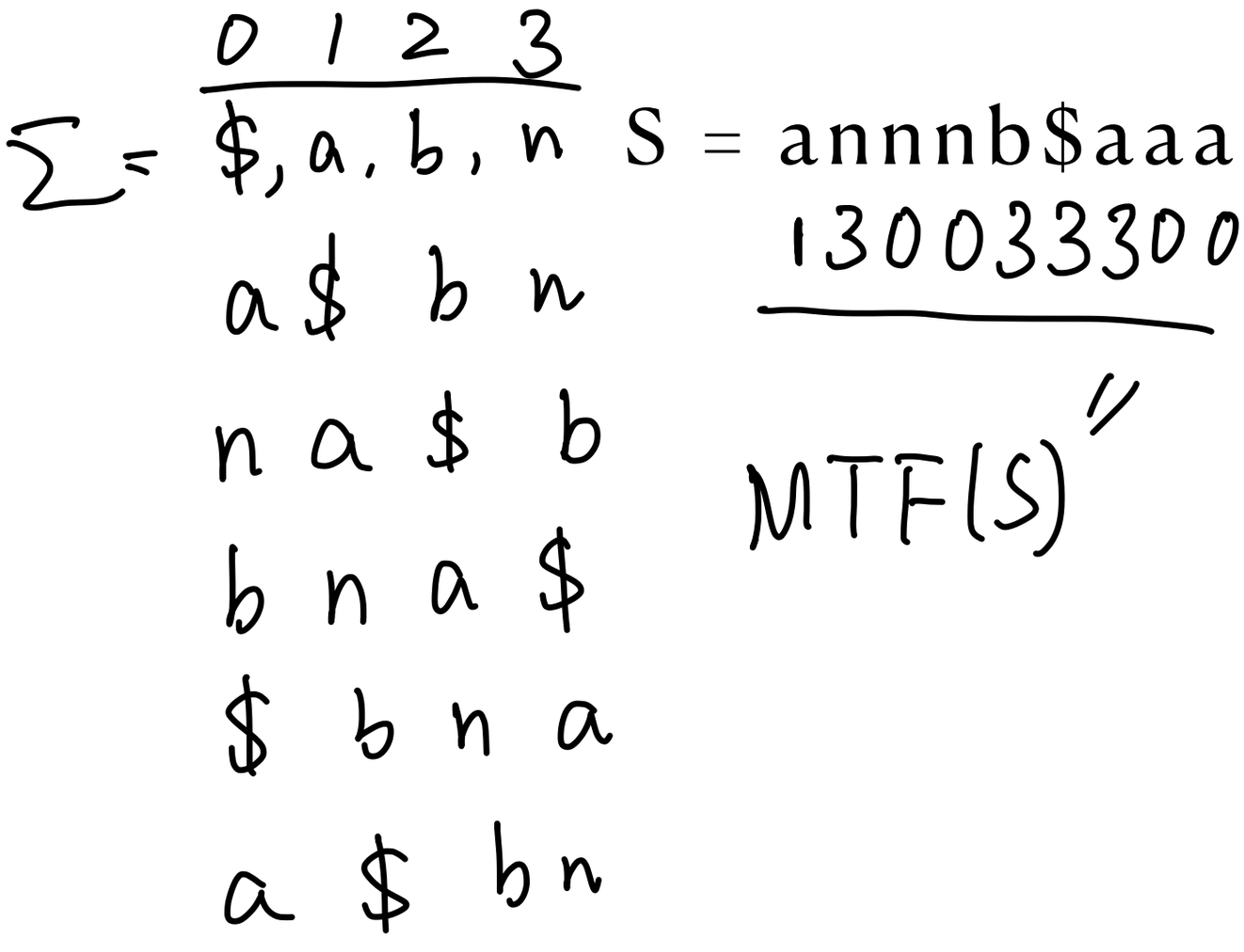
FM-index

- A compressed index for a (long) string (to save space)
- Still allows for fast substring-query with compressed index

FM-index(S) = **PrefixCode(RLE(MTF(BWT(S))))**

- MTF (move-to-front): an adaptive encoding scheme
- RLE (run-length-encoding): replace runs of zeros with count
- PrefixCode: Huffman encoding, for example
- Wavelet Tree: data structure that supports rank[j, c]
- RRR [Raman, Raman, Rao]: compressed bit vector data structure

Move-To-Front Encoding



- Runs of the same letter will lead to runs of zeros
- Self-adaptive: common letters get small numbers; rare numbers get big number