Faculty of Information Technology, Monash University

FIT2004: Algorithms and Data Structures

Week 7: Burrows-Wheeler Transform

Lecturer: Reza Haffari

These slides are prepared by M. A. Cheema and are based on the material developed by Arun Konagurthu and Lloyd Allison.

Outline

1. Compression
2. Burrows-Wheeler Transform (BWT)
3. Why BWT is effective for compression
4. Decompressing BWT
   A. Naïve Approach
   B. Efficient Approach
5. Substring search using BWT

Compression problem

Suppose you have a large sequence of characters (e.g., English text or DNA sequence). How can you compress the data?

Idea:

Original Text: this is mississippi's history. is this mississippi's history?

Modified: (rearrange such that we get many "runs" of the same characters)

hhhhiiiiooiiiiiiiiiittttmmsssssssssssrrppppyysssss (text length: 50)

Compressed: 4h4i2o10i4t2m11s2r4p2y5s (compressed length: 24)

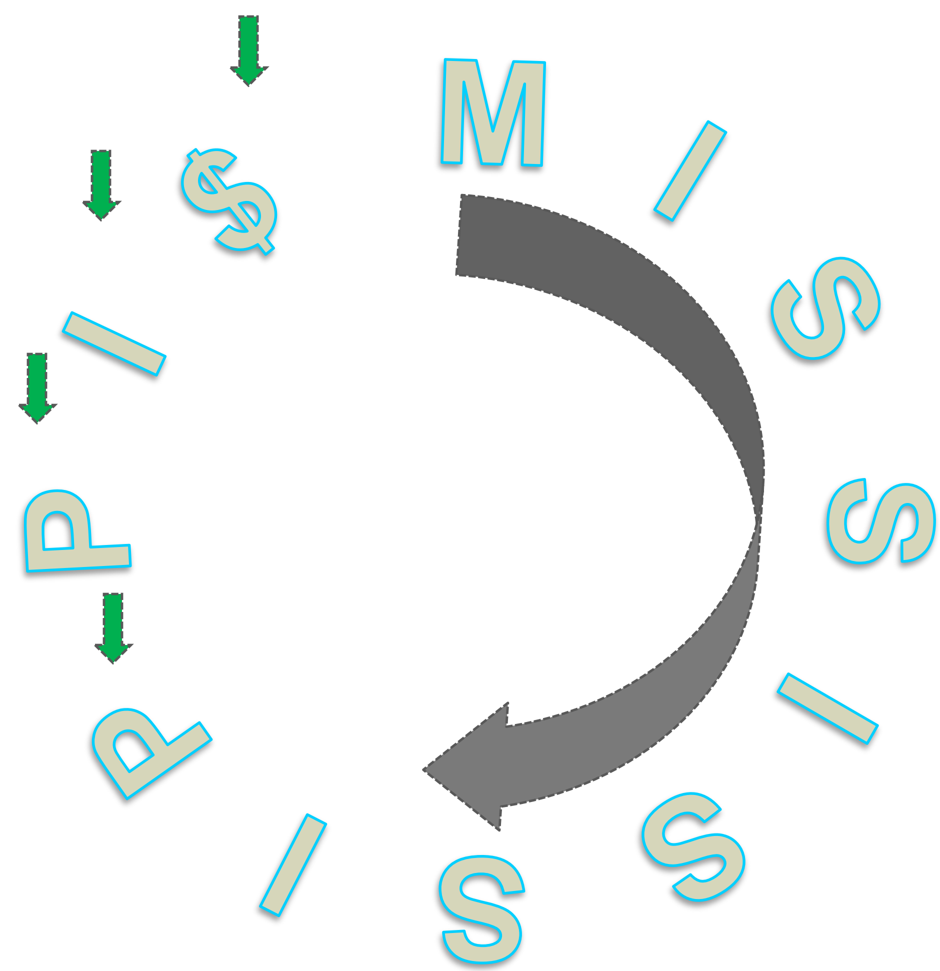
- Sorting the text provides "runs" of maximal lengths.
  - hhhhiiiiiiiiiiiiiimmooppppprrsssssssssssssssstttyy (text length: 50)
  - 4h14i2m2o4p2r16s4t2y (Compressed length: 20)
- However, sorting is not an acceptable solution! We must be able to recover the original text from the compressed data, i.e., decompression.
- So, the question is how to modify the original text such that there are many "runs" of the characters (to effectively compress the data) and the original text can be recovered from the decompressed data.
- Burrows-Wheeler Transform! Used in bzip2.

Outline

1. Compression
2. Burrows-Wheeler Transform (BWT)
3. Why BWT is effective for compression
4. Decompressing BWT
   1. Naïve Approach
   2. Efficient Approach
5. Substring search using BWT

Burrows-Wheeler Transform

M	I	S	S	I	S	S	I	P	P	I	$
$	M	I	S	S	I	S	S	I	P	P	I
I	$	M	I	S	S	I	S	S	I	P	P
P	I	$	M	I	S	S	I	S	S	I	P
P	P	I	$	M	I	S	S	I	S	S	I
I	P	P	I	$	M	I	S	S	I	S	S
S	I	P	P	I	$	M	I	S	S	I	S
S	S	I	P	P	I	$	M	I	S	S	I
I	S	S	I	P	P	I	$	M	I	S	S
S	I	S	S	I	P	P	I	$	M	I	S
S	S	I	S	S	I	P	P	I	$	M	I
I	S	S	I	S	S	I	P	P	I	$	M



All cyclic rotations of the text

M	I	S	S	I	S	S	I	P	P	I	$
$	M	I	S	S	I	S	S	I	P	P	I
I	$	M	I	S	S	I	S	S	I	P	P
P	I	$	M	I	S	S	I	S	S	I	P
P	P	I	$	M	I	S	S	I	S	S	I
I	P	P	I	$	M	I	S	S	I	S	S
S	I	P	P	I	$	M	I	S	S	I	S
S	S	I	P	P	I	$	M	I	S	S	I
I	S	S	I	P	P	I	$	M	I	S	S
S	I	S	S	I	P	P	I	$	M	I	S
S	S	I	S	S	I	P	P	I	$	M	I
I	S	S	I	S	S	I	P	P	I	$	M

All cyclic rotations of the text

$	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

Sort the strings in alphabetical order assuming $ is the smallest

M	I	S	S	I	S	S	I	P	P	I	$
$	M	I	S	S	I	S	S	I	P	P	I
I	$	M	I	S	S	I	S	S	I	P	P
P	I	$	M	I	S	S	I	S	S	I	P
P	P	I	$	M	I	S	S	I	S	S	I
I	P	P	I	$	M	I	S	S	I	S	S
S	I	P	P	I	$	M	I	S	S	I	S
S	S	I	P	P	I	$	M	I	S	S	I
I	S	S	I	P	P	I	$	M	I	S	S
S	I	S	S	I	P	P	I	$	M	I	S
S	S	I	S	S	I	P	P	I	$	M	I
I	S	S	I	S	S	I	P	P	I	$	M

All cyclic rotations of the text

$	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**

The last column of the sorted matrix is Burrows-Wheeler Transform

Note similarity with suffix array which corresponds to IDs of these suffixes/cyclic rotations

M	I	S	S	I	S	S	I	P	P	I	$
$	M	I	S	S	I	S	S	I	P	P	I
I	$	M	I	S	S	I	S	S	I	P	P
P	I	$	M	I	S	S	I	S	S	I	P
P	P	I	$	M	I	S	S	I	S	S	I
I	P	P	I	$	M	I	S	S	I	S	S
S	I	P	P	I	$	M	I	S	S	I	S
S	S	I	P	P	I	$	M	I	S	S	I
I	S	S	I	P	P	I	$	M	I	S	S
S	I	S	S	I	P	P	I	$	M	I	S
S	S	I	S	S	I	P	P	I	$	M	I
I	S	S	I	S	S	I	P	P	I	$	M

All cyclic rotations of the text

$	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

The last column of the sorted matrix is Burrows-Wheeler Transform

Once you get BWT, you can use run-length encoding to compress it (if the goal is compression).

M I S S I S S I P P I $

$ M I S S I S S I P P I

I $ M I S S I S S I P P

P I $ M I S S I S S I P

P P I $ M I S S I S S I

I P P I $ M I S S I S S

S I P P I $ M I S S I S

S S I P P I $ M I S S I

I S S I P P I $ M I S S

S I S S I P P I $ M I S

S S I S S I P P I $ M I

I S S I S S I P P I $ M

All cyclic rotations of the text

$ 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**

The last column of the sorted matrix is Burrows-Wheeler Transform

Exercise

What is the Burrows-Wheeler Transform of BIRD?

A. $BIRD
B. BI$RD
C. D$RBI
D. IRBD$
E. RDI$B
F. None of the above

Outline

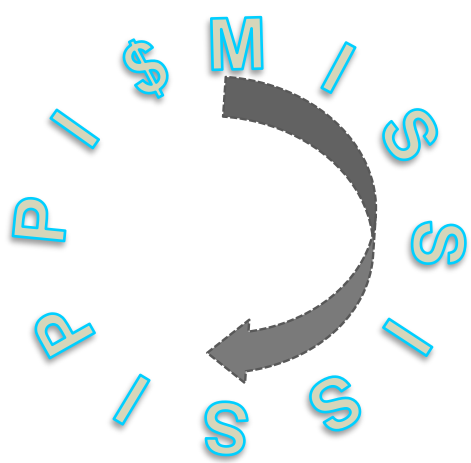
1. Compression
2. Burrows-Wheeler Transform (BWT)
3. Why BWT is effective for compression
4. Decompressing BWT
   A. Naïve Approach
   B. Efficient Approach
5. Substring search using BWT

Why is BWT effective for compression?

Last-First Property:

The last character of a row comes before the first character of the row in the input string.

- because each string in the matrix is a cyclic rotation of the text



$	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**

Why is BWT effective for compression?

- Consider a large English text. **IS** is a very common word. Thus, **I** appears before **S** in the text much more frequently compared to some other letters, e.g., **IS** is more frequent than **CABS**, **BOSS** etc.
- When the cyclic rotation matrix is sorted, all the occurrences of **S** in the first column appear together. The last column which is BWT will contain a lot of occurrences of **I**s because **I** appears before **S** much more frequently than the other letters.
- E.g., **this-is-a-historical-story** **(space replaced with – for clarity)**

```
.................................
s-a-historical-story$this-i
s-is-a-historical-story$thi
storical-story$this-is-a-hi
story$this-is-a-historical-
.................................
```

- **Effective for compression when text is large and has such biases in it (i.e., some letters appear before some others much more frequently).**

Outline

1. Compression
2. Burrows-Wheeler Transform (BWT)
3. Why BWT is effective for compression
4. Decompressing BWT
   A. Naïve Approach
   B. Efficient Approach
5. Substring search using BWT

Decompressing (Inverting) BWT

- We saw that BWT produces "runs" of characters which is effective in compression.
- But how do we invert BWT, i.e., how do we decompress the data to recover original text.

Inverting BWT

$ M I S S I S S I P P **I**		$
I $ M I S S I S S I P **P**		I
I P P I $ M I S S I S **S**		I
I S S I P P I $ M I S **S**		I
I S S I S S I P P I $ **M**		I
M I S S I S S I P P I **$**	→ sort	M
P I $ M I S S I S S I **P**		P
P P I $ M I S S I S S **I**		P
S I P P I $ M I S S I **S**		S
S I S S I P P I $ M I **S**		S
S S I P P I $ M I S S **I**		S
S S I S S I P P I $ M **I**		S

Is it true that if we sort the last column (i.e., BWT), we will get the first column of the Matrix?

Matrix Properties

$ 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**

Property 1:

Each column of the Matrix is a permutation of the string.

M I S S I S S I P P I $
$ M I S S I S S I P P I
I $ M I S S I S S I P P
P I $ M I S S I S S I P
P P I $ M I S S I S S I
I P P I $ M I S S I S S
S I P P I $ M I S S I S
S S I P P I $ M I S S I
I S S I P P I $ M I S S
S I S S I P P I $ M I S
S S I S S I P P I $ M I
I S S I S S I P P I $ M

All rotations before sorting

Is it true that each column in the Matrix is a permutation of the string $MISSISSIPPI?

Inverting BWT

$ 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

Concatenate Last and First columns

I $
P I
S I
S I
M I
$ M
P P
I P
S S
S S
I S
I S

Is it true that if we concatenate Last (i.e., BWT) and First (i.e., sorted BWT) columns, each row is a substring of size 2 of $MISSISSIPPI (considering cycles), i.e., I$ is considered a substring in cyclic rotation?

k-mers

k-mers of a string refers to its all possible substrings of size k (considering cyclic rotation).

- 2-mers of $MISSISSIPPI are $M, MI, IS, SS, SI, IS, SS, SI, IP, PP, PI, I$.
- 3-mers of $MISSISSIPPI are $MI, MIS, ISS, SSI, SIS, ISS, SSI, SIP, IPP, PPI, PI$, I$M.

Which of the following represents 2-mers of $BIRD.

A. D$, RI, BI, RD, $B
B. IR, D$, BI, $B, RD
C. B, DR, BI, IR, D
D. D, DR, RI, IB, B
E. None of the above

Inverting BWT

$ M I S S I S S I P P **I**	I $
I $ M I S S I S S I P **P**	P I
I P P I $ M I S S I S **S**	S I
I S S I P P I $ M I S **S**	S I
I S S I S S I P P I $ **M**	M I
M I S S I S S I P P I **$**	$ M
P I $ M I S S I S S I **P**	P P
P P I $ M I S S I S S **I**	I P
S I P P I $ M I S S I **S**	S S
S I S S I P P I $ M I **S**	S S
S S I P P I $ M I S S **I**	I S
S S I S S I P P I $ M **I**	I S

Concatenate Last and First columns

Is it true that concatenating last and first columns gives us 2-mers of $MISSISSIPPI?

Inverting BWT

$	**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**

Concatenate Last and First columns

I	$
P	I
S	I
S	I
M	I
$	M
P	P
I	P
S	S
S	S
I	S
I	S

Sort

$	M
I	$
I	P
I	S
I	S
M	I
P	I
P	P
S	I
S	I
S	S
S	S

Is it true that sorting the 2-mers gives us the first two columns of the Matrix?

Yes! Note that we have obtained the first two columns of the matrix using BWT.

Inverting BWT

$ M I S S I S S I P P **I**	I $ M
I $ M I S S I S S I P **P**	P I $
I P P I $ M I S S I S **S**	S I P
I S S I P P I $ M I S **S**	S I S
I S S I S S I P P I $ **M**	M I S
M I S S I S S I P P I **$**	$ M I
P I $ M I S S I S S I **P**	P P I
P P I $ M I S S I S S **I**	I P P
S I P P I $ M I S S I **S**	S S I
S I S S I P P I $ M I **S**	S S I
S S I P P I $ M I S S **I**	I S S
S S I S S I P P I $ M **I**	I S S

Concatenate Last and First two columns

Concatenating the last and first two columns gives us the 3-mers of $MISSISSIPPI.

Inverting BWT

	Concatenate Last and First two columns	Sort
$ M I S S I S S I P P I	I $ M	$ M I
I $ M I S S I S S I P P	P I $	I $ M
I P P I $ M I S S I S S	S I P	I P P
I S S I P P I $ M I S S	S I S	I S S
I S S I S S I P P I $ M	M I S	I S S
M I S S I S S I P P I $	$ M I	M I S
P I $ M I S S I S S I P	P P I	P I $
P P I $ M I S S I S S I	I P P	P P I
S I P P I $ M I S S I S	S S I	S I P
S I S S I P P I $ M I S	S S I	S I S
S S I P P I $ M I S S I	I S S	S S I
S S I S S I P P I $ M I	I S S	S S I

Sorting the 3-mers gives us the first three columns of the matrix.

Inverting BWT

$ M I S S I S S I P P I	I $ M I	$ M I S
I $ M I S S I S S I P P	P I $ M	I $ M I
I P P I $ M I S S I S S	S I P P	I P P I
I S S I P P I $ M I S S	S I S S	I S S I
I S S I S S I P P I $ M	M I S S	I S S I
M I S S I S S I P P I $	$ M I S	M I S S
P I $ M I S S I S S I P	P P I $	P I $ M
P P I $ M I S S I S S I	I P P I	P P I $
S I P P I $ M I S S I S	S S I P	S I P P
S I S S I P P I $ M I S	S S I S	S I S S
S S I P P I $ M I S S I	I S S I	S S I P
S S I S S I P P I $ M I	I S S I	S S I S

Concatenate Last and First three columns

Sort

- Concatenating the last column with the first three columns gives us 4-mers.
- Sorting the 4-mers gives us the first four columns.

Inverting BWT

Inverting BWT

Create an empty table **M**

Make a column C containing BWT

Repeat len(BWT) times

    Concatenate C with M

    Sort M alphabetically

Return the first row (ignore $).

Let N be the total number of characters in the original string. What is the complexity?

Time complexity:

Requires N calls to sorting

Cost of sorting N rows where each row has N characters: O (N^2) using radix sort

Total cost for sorting: O(N^3) if radix sort is being used

Space complexity:

Size of matrix: O(N^2)

Can we improve?

Yes! It is possible to invert in O(N) time complexity and O(N) space complexity

Outline

1. Compression
2. Burrows-Wheeler Transform (BWT)
3. Why BWT is effective for compression
4. Decompressing BWT
   A. Naïve Approach
   B. Efficient Approach
5. Substring search using BWT

Faster Inversion of BWT

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

$ M I S S I S S I P P I

We have used different colors for different occurrences of S in $MISSISSIPPI.

Which row of the matrix has the **red** S in the last column?

Which row of the matrix has the **red** S in the first column?

Which row of the matrix has the **purple** S in the last column and which row has the **purple** S in the first column?

Which row of the matrix has the **blue** S in the last column and which row has the **blue** S in the first column?

Which row of the matrix has the **black** S in the last column and which row has the **black** S in the first column?

Observation

The relative orders of the **same** characters in the first column and the last column is the same.

E.g., the i-th S in the first column is the i-th S in the last column

Faster Inversion of BWT

$	M	I	S	S	I	S	S	I	P	P	I_1
I_1	$	M	I	S	S	I	S	S	I	P	P_1
I_2	P	P	I	$	M	I	S	S	I	S	S_1
I_3	S	S	I	P	P	I	$	M	I	S	S_2
I_4	S	S	I	S	S	I	P	P	I	$	M_1
M_1	I	S	S	I	S	S	I	P	P	I	**$**
P_1	I	$	M	I	S	S	I	S	S	I	P_2
P_2	P	I	$	M	I	S	S	I	S	S	I_2
S_1	I	P	P	I	$	M	I	S	S	I	S_3
S_2	I	S	S	I	P	P	I	$	M	I	S_4
S_3	S	I	P	P	I	$	M	I	S	S	I_3
S_4	S	I	S	S	I	P	P	I	$	M	I_4

i-th occurrence of a letter in first column and i-th occurrence of the letter in the last column point to the same letter.

Faster Inversion of BWT

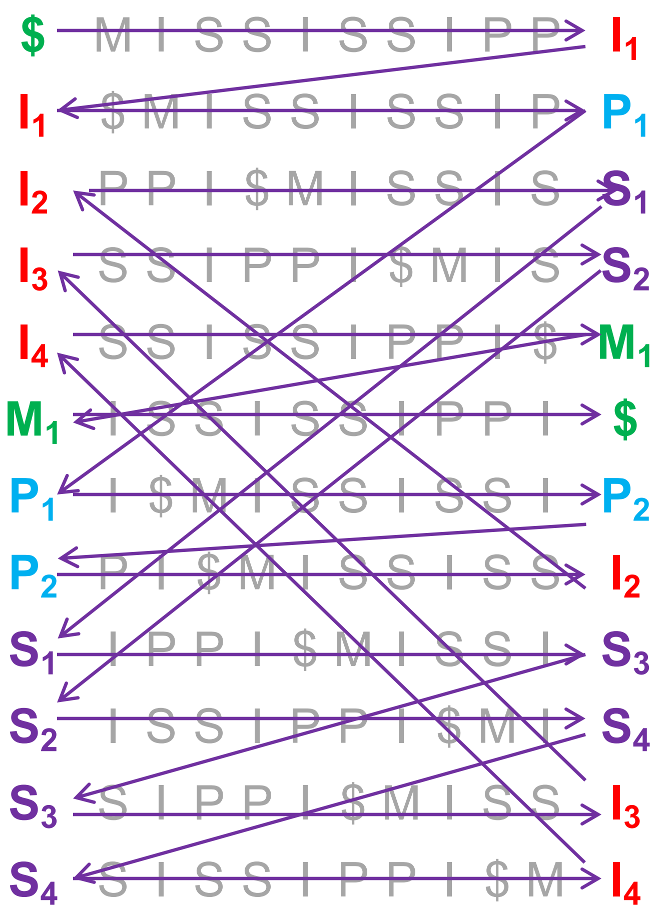
$ M I S S I S S I P P I
I $ M I S S I S S I P P
I P P I $ M I S S I S S
I S S I P P I $ M I S S
I S S I S S I P P I $ M
M I S S I S S I P P I $
P I $ M I S S I S S I P
P P I $ M I S S I S S I
S I P P I $ M I S S I S
S I S S I P P I $ M I S
S S I P P I $ M I S S I
S S I S S I P P I $ M I

Why does this observation hold?

- Rotate each row that ends at S by one character
- First characters of all these are the same (i.e., S)
- This means the sorting is based on the remaining characters, i.e., the sorting order is determined by stripping off S.
- Hence, the row that appeared earlier before rotation must appear earlier after rotation.

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

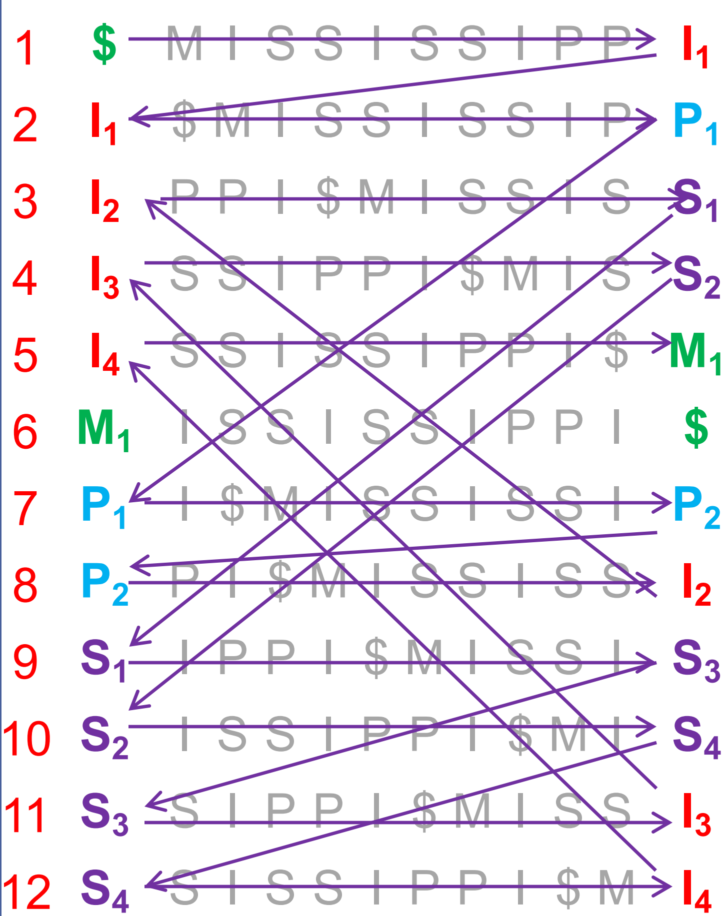
Faster Inversion of BWT



- So, we know which character in the last column corresponds to which character in the first column. The inversion can then be done as follows.
- Start from $ in the first column (F)
- The previous letter in this row **I** is the letter before $ in the original string (Last-First property). Recover this letter.
- Now, find this **I** in the first column
- The previous letter in this row **P** is the letter before this **I** in the original string (Last-First property). Recover this letter
- Now, find this P in the first column.
- The previous letter in this row P is the letter before this P in the original string (Last-First property). Recover it.
- and so on …

M I S S I S S I P P I $

Faster Inversion of BWT



Pseudocode

- Number each character in the Last column
- Create a Rank array that records the row number of the first occurrence of each character in sorted order
- row = 1
- str = “$”
- Repeat len(BWT) - 1 times:
- c = Last[row]
- str = c + str
- Row = Rank[c] + num(c) - 1

Space Complexity:
O(N)

Rank

2	6	7	9
I	M	P	S

Time Complexity:
O(N) if using counting sort to obtain the first column

str M I S S I S S I P P I $

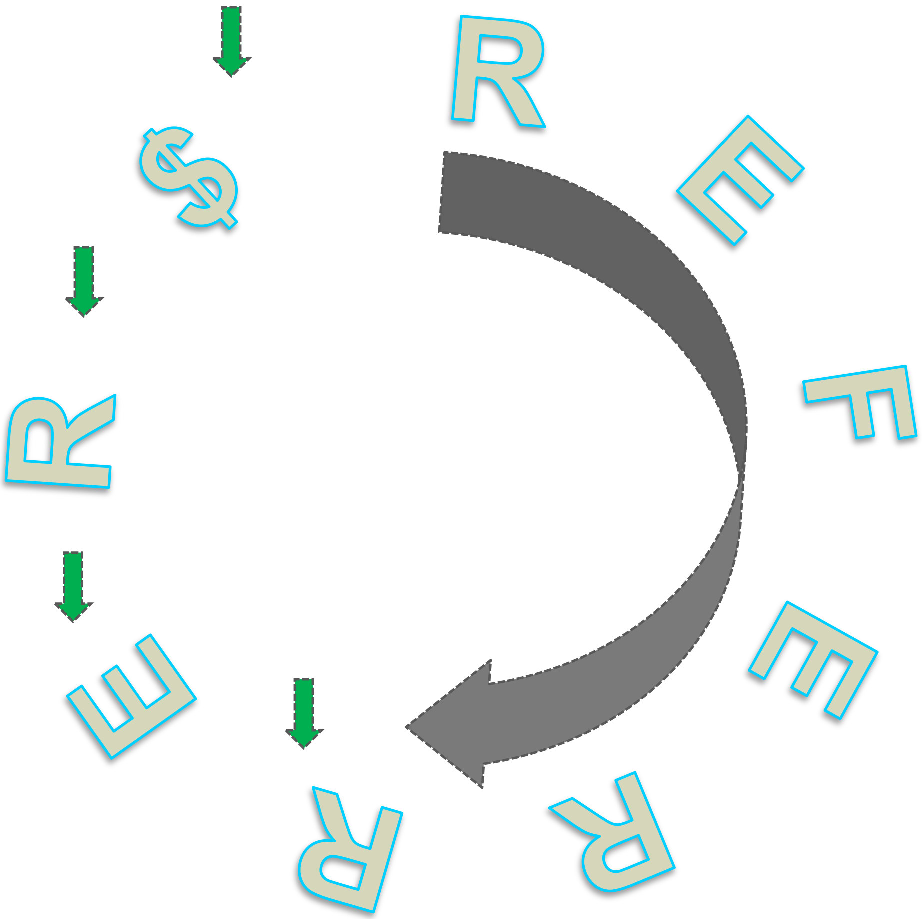
Practice

What is Burrows-Wheeler Transform of REFERRER?

A. RRRFEE$RE
B. $REFERRER
C. RRRFE$ERE
D. RRREFEE$R
E. None of the above

Practice: Burrows-Wheeler Transform

R E F E R R E R $
$ R E F E R R E R
R $ R E F E R R E
E R $ R E F E R R
R E R $ R E F E R
R R E R $ R E F E
E R R E R $ R E F
F E R R E R $ R E
E F E R R E R $ R



All cyclic rotations of the text

Practice: Burrows-Wheeler Transform

All cyclic rotations of the text		Sorted rows
R E F E R R E R $	→	$ R E F E R R E **R**
$ R E F E R R E R		E F E R R E R $ **R**
R $ R E F E R R E		E R $ R E F E R **R**
E R $ R E F E R R		E R R E R $ R E **F**
R E R $ R E F E R		F E R R E R $ R **E**
R R E R $ R E F E		R $ R E F E R R **E**
E R R E R $ R E F		R E F E R R E R **$**
F E R R E R $ R E		R E R $ R E F E **R**
E F E R R E R $ R		R R E R $ R E F **E**

Sort all rows alphabetically

The last colum is BWT.

All cyclic rotations of the text

Practice: Efficient Inversion of BWT

1	$ R E F E R R E R
2	E F E R R E R $ R
3	E R $ R E F E R R
4	E R R E R $ R E F
5	F E R R E R $ R E
6	R $ R E F E R R E
7	R E F E R R E R $
8	R E R $ R E F E R
9	R R E R $ R E F E

Psuedocde:

Number each character in the Last column

Create a Rank array that records the row number of the first occurrence of each character in sorted order

```
row = 1
str = "$"
Repeat len(BWT) - 1 times:
        c = Last[row]
        str = c + str
        Row = Rank[c] + num(c) - 1
```

What are the values of the Rank array?

What is the Row of this character?

A. 2, 6, 9
B. 4, 5, 9
C. 2, 5, 6
D. None of the above

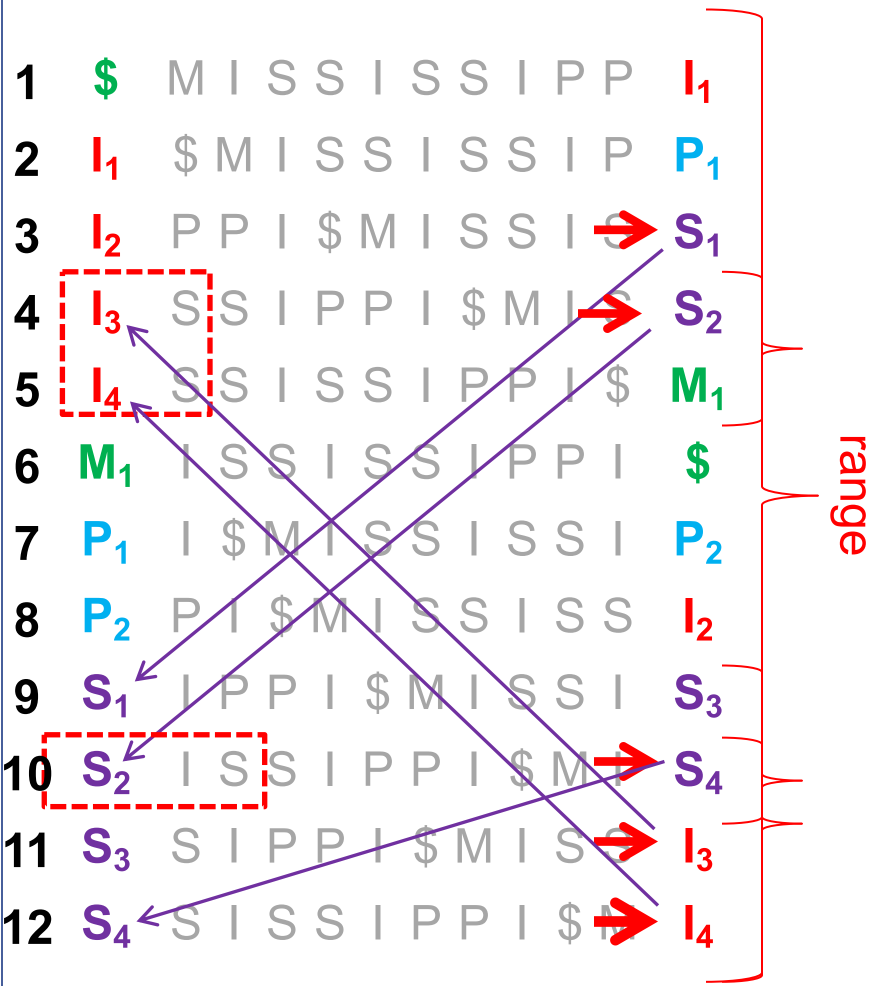
Rank

E	**F**	**R**

Outline

1. Compression
2. Burrows-Wheeler Transform (BWT)
3. Why BWT is effective for compression
4. Decompressing BWT
   A. Naïve Approach
   B. Efficient Approach
5. Substring search using BWT

Substring search using BWT



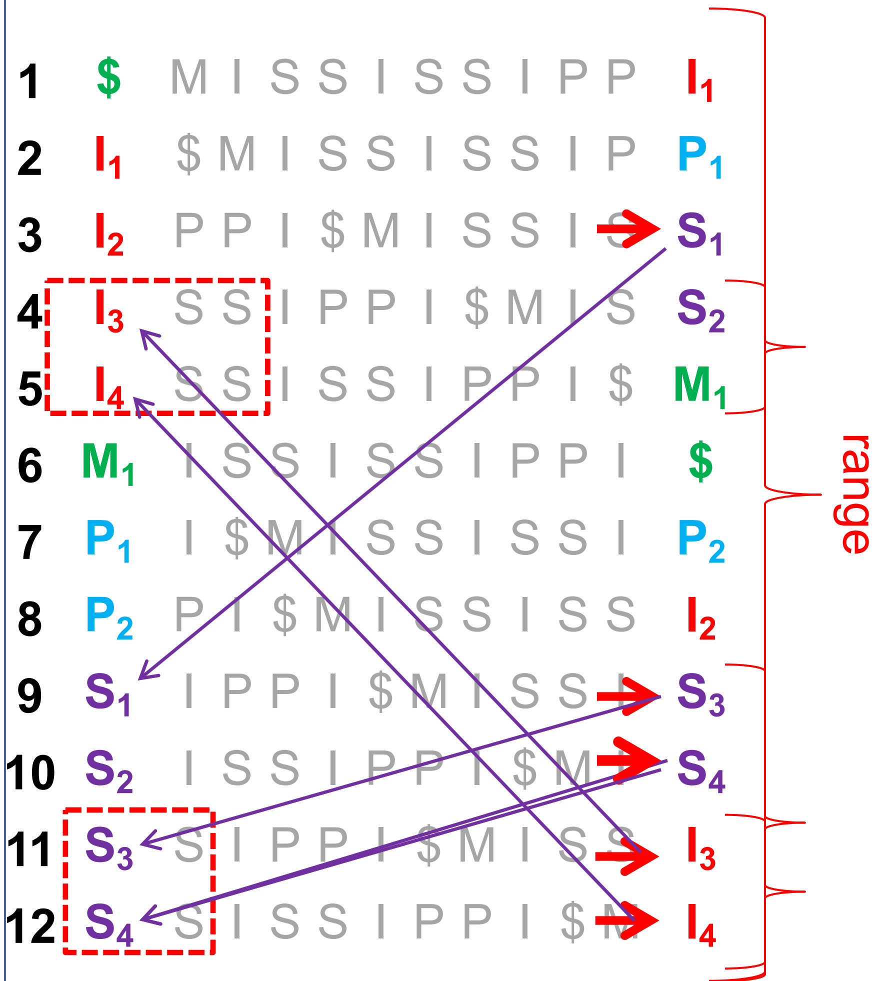
Suppose we want to search **SIS** in the string.

- Initially the range contains all rows of BWT
- Start from the last character **S** of SIS.
- Find first **S** in the range and the last **S** in the range in the Last column
- Find the corresponding **S**s in the first column and update the range
- Now, find the first **I** in the range and the last **I** in the range in the Last column
- Find the corresponding **I**s in the first column and update the range.
- Now, find the first **S** in the range and the last S in the range
- Find the corresponding **S**s in first column and update the range

At any stage, if the character is not found in the range then the substring is not present and false can be returned.

S I S

Substring search using BWT



Another example:

Suppose we want to search **ISS** in the string.

- Initially the range contains all rows of BWT
- Start from the last character **S** of SIS.
- Find first **S** in the range and the last **S** in the range in the Last column
- Find the corresponding **S**s in the first column and update the range
- Now, find the first **S** in the range and the last S in the range in the Last column
- Find the corresponding **S**s in the first column and update the range.
- Now, find the first **I** in the range and the last **I** in the range
- Find the corresponding **I**s in first column and update the range

I S S

Substring search using BWT

1	\$	M I S S I S S I P P	I_1
2	I_1	\$ M I S S I S S I P	P_1
3	I_2	P P I \$ M I S S I S	S_1
4	I_3	S S I P P I \$ M I S	S_2
5	I_4	S S I S S I P P I \$	M_1
6	M_1	I S S I S S I P P I	\$
7	P_1	I \$ M I S S I S S I	P_2
8	P_2	P I \$ M I S S I S S	I_2
9	S_1	I P P I \$ M I S S I	S_3
10	S_2	I S S I P P I \$ M I	S_4
11	S_3	S I P P I \$ M I S S	I_3
12	S_4	S I S S I P P I \$ M	I_4

How to efficiently compute first and last occurrence of a character c in the range.

- **For each character, create a sorted array of their positions in the last column – this can be done in linear time**

To search a character c in range(i,j), use binary search.

- **to search the first S in the range (5,11), binary search for the smallest position equal to or larger than 5 in the array of S**
- **to search the last S in the range (5,11), binary search for the largest position smaller than or equal to 11**

I	1, 8, 11, 12
M	5
P	2, 7
S	3, 4, 9, 10

Time Complexity:

O(M log N) where M is length of substring.

Could be improved to O(M) by maintaining, at each row, the next and previous occurrence of each character

Practice: Substring matching

1 $ R E F E R R E R
2 E F E R R E R $ R
3 E R $ R E F E R R
4 E R R E R $ R E F
5 F E R R E R $ R E
6 R $ R E F E R R E
7 R E F E R R E R $
8 R E R $ R E F E R
9 R R E R $ R E F E

- Search ER
- Search RE
- Search FEF

Summary

Take home message

- Burrows-Wheeler Transform is an elegant algorithm that allows efficient and effective compression and substring matching

Things to do (this list is not exhaustive)

- Read more about Burrows-Wheeler Transform and understand how and why it works
- Implement it in Python

Coming Up Next

- Introduction to Graphs and Path problems on Graphs