

## COMP9319 Web Data Compression and Search

BWT, MTF and Pattern Matching

1

### BWT

- Burrows–Wheeler transform (BWT) is an algorithm used to prepare data for use with data compression techniques such as bzip2.
- It was invented by Michael Burrows and David Wheeler in 1994 at DEC SRC, Palo Alto, California.
- It is based on a previously unpublished transformation discovered by Wheeler in 1983.

2

A simple example  
Input:  
#BANANAS

<https://eduassistpro.github.io/>

3

All rotations

#BANANAS

BANANA

#BANAN

S#BANA

BAN

#BA

S#B

BANANAS#

4

Sort the rows

#BANANAS  
ANANAS#B  
ANAS#BAN  
AS#BANAN  
BANANAS#  
NANAS#BA  
NAS#BANA  
S#BANANA

5

Output

#BANANAS  
ANANAS#B  
ANAS#BAN  
AS#BANAN  
BANANAS#  
NANAS#BA  
NAS#BANA  
S#BANANA

6

Now the inverse, for decoding...

Input:

S  
B  
N  
N  
#  
A  
A  
A

7

First add

S  
B  
N  
N  
#  
A  
A  
A

8

Then sort

Assignment Project Exam Help

#

<https://eduassistpro.github.io/>

Add WeChat [edu\\_assist\\_pro](https://edu_assist_pro)

9

Add again

S#  
BA  
NA  
NN  
#B  
AN  
AN  
AS

10

Then sort

#B  
AN  
AN  
AS  
BA  
NA  
NA  
S#

11

Then add

S#B  
BAN  
NAN  
NAS  
#BA  
ANA  
ANA  
AS#

12

Then sort

#BA  
ANA  
ANA  
AS#  
BAN  
NAN  
NAS  
S#B

13

Then add

S#BA  
BANA  
NANA  
NAS#  
#BAN  
ANAN  
ANAS  
AS#B

14

Then sort

#BAN

<https://eduassistpro.github.io/>  
Add WeChat **edu\_assist\_pro**

15

Then add

S#BAN  
BANAN  
NANAS  
NAS#  
#BANA  
ANANA  
ANAS#  
AS#BA

16

Then sort

#BANA  
ANANA  
ANAS#  
AS#BA  
BANAN  
NANAS  
NAS#B  
S#BAN

17

Then add

S#BANA  
BANANA  
NANAS#  
NAS#BA  
#BANAN  
ANANAS  
ANAS#B  
AS#BAN

18

17

18

Then sort

#BANAN  
ANANAS  
ANAS#B  
AS#BAN  
BANANA  
NANAS#  
NAS#BA  
S#BANA

19

Then add

S#BANAN  
BANANAS  
NANAS#B  
NAS#BAN  
#BANANA  
ANANAS#  
ANAS#BA  
AS#BANA

20

Then sort

#BANANA  
  
BANANAS  
NANAS#B  
NAS#BAN  
S#BANAN

21

Then add

S#BANANA  
BANANAS#  
NANAS#BA  
NAS#BANK  
#BANANAS  
ANANAS#B  
ANAS#BAN  
AS#BANAN

22

Then sort (???)

#BANANAS  
ANANAS#B  
ANAS#BAN  
AS#BANAN  
BANANAS#  
NANAS#BA  
NAS#BANA  
S#BANANA

23

Implementation

Do we need to represent the table in the encoder?

No, a single pointer for each row is needed.

24

## BWT(S)

```
function BWT (string s)
create a table, rows are all possible
rotations of s
sort rows alphabetically
return (last column of the table)
```

25

## InverseBWT(S)

```
function inverseBWT (string s)
create empty table

repeat length(s) times
insert s as a column of table before first
column of the table // first insert creates
first column
sort rows of the table alphabetically
return (row that ends with the 'EOF' character)
```

26

25

26

## Move to Front (MTF)

Reduce entropy based on local frequency correlation

Usually used for BWT encoding step

Author and detail:

Original paper at cs9319/papers  
<http://www.arturocampos.com/cs9319/mtf.html>

27

27

## Example: abaabacab

Symbol	Code	List
		abcde.....
		bacde.....
		a bacde.....
		abcde.....
		de.....
		de.....
		de.....
d	3	dacbe.....

To transform a general file, the list has 256 ASCII symbols.

28

28

## BWT compressor vs ZIP

ZIP (i.e., LZW based)	BWT+RLE+MTF+AC				
File Name	Raw Size	PKZIP Size	PKZIP Bits/Byte	BWT Size	BWT Bits/Byte
bib	111,261	35,821	2.58	29,567	2.13
book1	768,771	315,999	3.29	275,831	2.87
book2	610,856	209,061	2.74	186,592	2.44
geo	102,400	68,917	5.38	62,120	4.85
news	377,109	146,010	3.10	134,174	2.85
obj1	21,504	10,311	3.84	10,857	4.04
obj2	246,814	81,846	2.65	81,948	2.66

29

From <http://marknelson.us/1996/09/01/bwt/>

## Other ways to reverse BWT

Consider  $L = BWT(S)$  is composed of the symbols  $V_0 \dots V_{N-1}$ , the transformed string may be parsed to obtain:

The number of symbols in the substring  $V_0 \dots V_{i-1}$  that are identical to  $V_i$ .

For each unique symbol,  $V_i$ , in  $L$ , the number of symbols that are lexicographically less than that symbol.

30

30

## Example

Position	Symbol	# Matching
0	B	0
1	N	0
2	N	1
3	[	0
4	A	0
5	A	1
6	]	0
7	A	2

Symbol	# LessThan
A	0
B	3
N	4
[	6
]	7

31

## ???????

Position	Symbol	# Matching
0	B	0
1	N	0
2	N	1
3	[	0
4	A	0
5	A	1
6	]	0
7	A	2

Symbol	# LessThan
A	0
B	3
N	4
[	6
]	7

32

??????A]

Assignment Project Exam Help

Position	Symbol	# Matching
0	B	0
1	N	0
2	N	1
3	[	0
4	A	0
5	A	1
6	]	0
7	A	2

Symbol	# LessThan

33

?????NA]

Assignment Project Exam Help

Position	Symbol	# Matching
0	B	0
1	N	0
2	N	1
3	[	0
4	A	0
5	A	1
6	]	0
7	A	2

Symbol	# LessThan

34

????ANA]

Position	Symbol	# Matching
0	B	0
1	N	0
2	N	1
3	[	0
4	A	0
5	A	1
6	]	0
7	A	2

Symbol	# LessThan
A	0
B	3
N	4
[	6
]	7

35

??NANA]

Assignment Project Exam Help

Position	Symbol	# Matching
0	B	0
1	N	0
2	N	1
3	[	0
4	A	0
5	A	1
6	]	0
7	A	2

Symbol	# LessThan
A	0
B	3
N	4
[	6
]	7

36

??ANANA]

Position	Symbol	# Matching
0	B	0
1	N	0
2	N	1
3	[	0
4	A	0
5	A	1
6	]	0
7	A	2

Symbol	# LessThan
A	0
B	3
N	4
[	6
]	7

37

?BANANA]

Position	Symbol	# Matching
0	B	0
1	N	0
2	N	1
3	[	0
4	A	0
5	A	1
6	]	0
7	A	2

Symbol	# LessThan
A	0
B	3
N	4
[	6
]	7

38

[BANANA]

Position	Symbol	# Matching
0	B	0
1	N	0
2	N	1
3	[	0
4	A	0
5	A	1
6	]	0
7	A	2

Symbol	# LessThan

39

An illustration

A                      B  
A                      N  
A                      N  
B                      [  
N                      A  
N                      A  
[                      ]  
]  
First                  Last

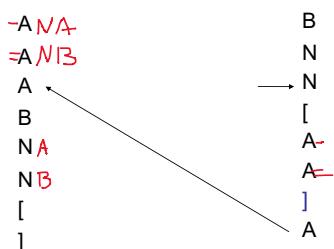
41

A]

A                      B  
A                      N  
A                      N  
B                      [  
N                      A  
N                      A  
[                      ]  
]  
Last                  First

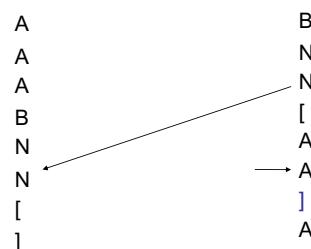
42

NA]



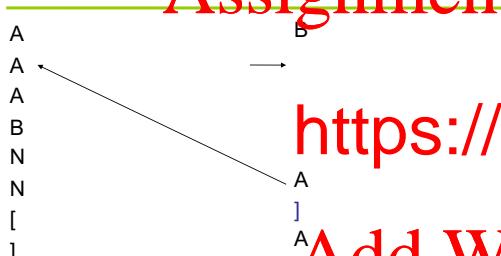
43

ANA]



44

NANA]



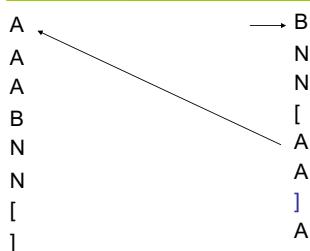
45

ANANA]



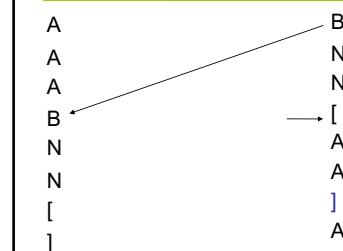
46

BANANA]



47

[BANANA]



48

## Dynamic BWT ?

Instead of reconstructing BWT, local reordering from the original BWT.

Details:

Salson M, Lecroq T, Léonard M and Mouchard L (2009). "A Four-Stage Algorithm for Updating a Burrows–Wheeler Transform". *Theoretical Computer Science* 410 (43): 4350.

49

Search

50

What is Pattern Matching?

**Assignment Project Exam Help**

- Definition:
  - given a text string  
find the pattern insi  
• T: “the rain in spain stays mainly on the pl  
• P: “n th”

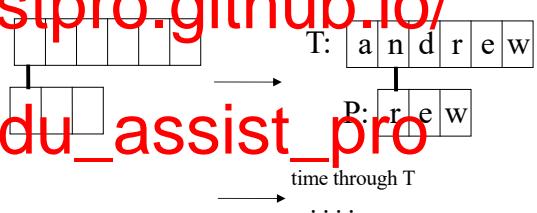
51

<https://eduassistpro.github.io/>

Add WeChat **edu\_assist\_pro**

The Brute Force Algorithm

- Check each position in the text T to see starts in that position



52

## Analysis

- Brute force pattern matching runs in time  $O(mn)$  in the worst case.
- But most searches of ordinary text take  $O(m+n)$ , which is very quick.

53

*continued*

- The brute force algorithm is fast when the alphabet of the text is large
  - e.g. A..Z, a..z, 1..9, etc.
- It is slower when the alphabet is small
  - e.g. 0, 1 (as in binary files, image files, etc.)

54

*continued*

- Example of a worst case:
  - T: "aaaaaaaaaaaaaaaaaaaaah"
  - P: "aaah"
- Example of a more average case:
  - T: "a string searching example is standard"
  - P: "store"

55

## The KMP Algorithm

- The Knuth-Morris-Pratt (KMP) algorithm looks for the pattern in the text in a *left-to-right* order (like the brute force algorithm).
- But it shifts the pattern more intelligently than the brute force algorithm.

56

continued

55

56

- Summary**
- If a mismatch occurs between the text and pattern P at  $P[j]$ , what shift the pattern to a comparisons?

<https://eduassistpro.github.io/>

57

- Summary**
- If a mismatch occurs between the text and , what is the *most* we can to avoid wasteful

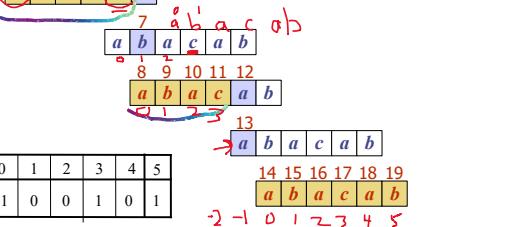
refix of  $P[0 \dots j-1]$  that

Add WeChat [edu\\_assist\\_pro](https://edu_assist_pro)

58

58

## Example

T:	
P:	
	
$k$	0 1 2 3 4 5
$F(k)$	-1 0 0 1 0 1

59

## KMP Advantages

- KMP runs in optimal time:  $O(m+n)$ 
  - very fast
- The algorithm never needs to move backwards in the input text, T
  - this makes the algorithm good for processing very large files that are read in from external devices or through a network stream

60

60

## KMP Disadvantages

- KMP doesn't work so well as the size of the alphabet increases
  - more chance of a mismatch (more possible mismatches)
  - mismatches tend to occur early in the pattern, but KMP is faster when the mismatches occur later

61

61

## The Boyer-Moore Algorithm

- The Boyer-Moore pattern matching algorithm is based on two techniques.
- 1. The *looking-glass* technique
  - find P in T by moving *backwards* through P, starting at its end

62

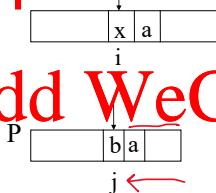
62

## Assignment Project Exam Help

- 2. The *character-jump* technique

- when a mismatch o
  - the character in patt same as  $T[i]$

- There are 3 possible cases.

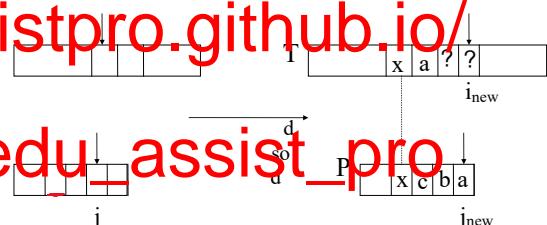


63

Add WeChat [edu\\_assist\\_pro](https://eduassistpro.github.io/)

### Case 1

- If P contains x somewhere, then try to shift P right to align the last occurrence i].

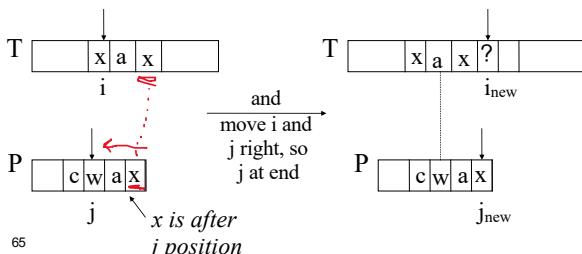


64

64

### Case 2

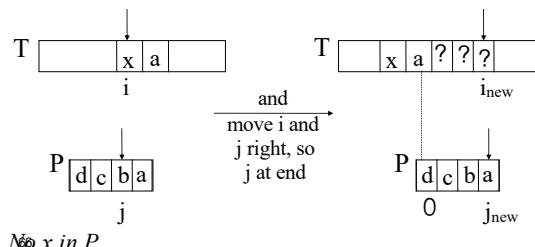
- If P contains x somewhere, but a shift right to the last occurrence is *not* possible, then shift P right by 1 character to  $T[i+1]$ .



65

### Case 3

- If cases 1 and 2 do not apply, then shift P to align  $P[0]$  with  $T[i+1]$ .

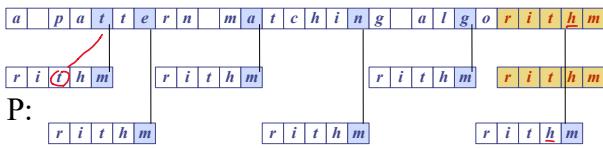


66

65

## Boyer-Moore Example (1)

T:



P:

67

67

## Last Occurrence Function

- Boyer-Moore's algorithm preprocesses the pattern P and the alphabet A to build a last occurrence function L()
  - L() maps all the letters in A to integers
- L(x) is defined as: // x is a letter in A
  - the largest index i such that P[i] == x, or
  - -1 if no such index exists

68

68

## L() Example

- A = {a, b, c, d}
- P: "abacab"

P: a b a c a b

x	a	b	c	d
L(x)	4	5	3	-1

L() stores indexes into P[]

69

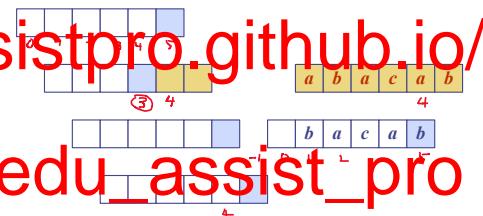
<https://eduassistpro.github.io/>

Add WeChat

edu\_assist\_pro

## Boyer-Moore Example (2)

T: a b a c a a b a d c a b a c a b a a b b



70

x	a	b	c	d
L(x)	4	5	3	-1

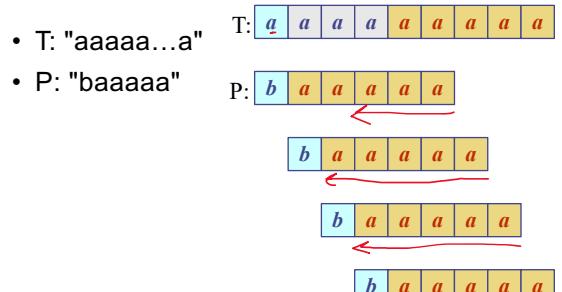
70

## Analysis

- Boyer-Moore worst case running time is  $O(nm + A)$
- But, Boyer-Moore is fast when the alphabet (A) is large, slow when the alphabet is small.
  - e.g. good for English text, poor for binary
- Boyer-Moore is *significantly faster than brute force* for searching English text.

71

## Worst Case Example

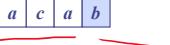


72

72

## Boyer-Moore Example (2)

T: 

P: 

a	b	a	c	a	b
a	b	a	c	a	b
a	b	a	c	a	b
a	b	a	c	a	b
a	b	a	c	a	b

73

x	a	b	c	d
L(x)	4	5	3	-1

73

## Boyer-Moore: Good suffix rule

If  $t$  is the longest suffix of  $P$  that matches  $T$  in the current position, then  $P$  can be shifted so that the previous occurrence of  $t$  in  $P$  matches  $T$ . In fact, it can be required that the character before the previous occurrence of  $t$  be different from the character before the occurrence of  $t$  as a suffix. If no such previous occurrence of  $t$  exists, then the following cases apply:

- Find the smallest shift that matches a prefix of the pattern to a suffix of  $t$  in the text
- If there's no such match, shift the pattern by  $n$  (the length of  $P$ )

74

## Boyer-Moore: Good suffix rule

# Assignment Project Exam Help

- Consider the example in the paper:

•  $P = \underline{\text{ABCXX}}$      $T = \text{YXABCXXXXAB}C$   
     01234 →

• -6 -5 -4 -3 2 0 0 3 2 7

75

## Boyer-Moore: Good suffix rule

- Consider the example in the paper:

•  $P = \underline{\text{ABCXX}}$      $T = \text{A}EYX\overline{\text{ABC}}\text{XXXXAB}C$   
     01234 ←

76

Add WeChat **edu\_assist\_pro**

0007

## Boyer-Moore: Good suffix rule

- Consider the examples in the paper:
- ABCXXXABC
- ABYXCDEYYX
- -6 -5 -4 -3 -2 -1 -3 -2 7
- -9 -8 -7 -6 -5 -4 1 -2 7

77

## Boyer-Moore: Good suffix rule

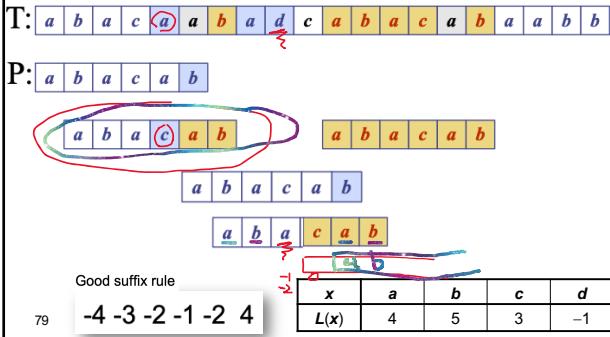
- Another example:

•  $P = \underline{\text{abacab}}$      $T = \text{X}C\text{ab}\overline{\text{ab}}\text{a}=\text{ab}$   
     012345

• -4 -3 -2 -1 -2 4

78

### Boyer-Moore Example (3)



79

### KMP & BM

- Please refer to the original papers (available at WebCMS) for the details of the algorithms
- Most text processors use BM for “find” (& “replace”) due to its good performance for general text documents

80

# Assignment Project Exam Help

<https://eduassistpro.github.io/>

Add WeChat edu\_assist\_pro