COMP9319 Web Data Compression and Search

Lecture 3: BWT and Pattern Matching

A simple example

Input:

#BANANAS

Excerpted from Wikipedia

All rotations

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

Sort the rows

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

Output

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

Exercise: you can try the example

rabcabcababaabacabcabcabcababaa$

aabbbbccacccrcbaaaaaaaaaabbbbba$

Now the inverse…

Input:

`S`

`B`

`N`

`N`

`#`

`A`

`A`

`A`

First add

S
B
N
N
#
A
A
A

Then sort

```
#
A
A
A
B
N
N
S
```

Add again

```
S#
BA
NA
NA
#B
AN
AN
AS
```

Then sort

```
#B
AN
AN
AS
BA
NA
NA
S#
```

Then add

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

Then sort

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

Then add

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

Then sort

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

Then add

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

Then sort

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

Then add

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

Then sort

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

Then add

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

Then sort

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

Then add

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

Then sort (?)

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

Implementation

- Do we need to represent the table in the encoder?
- No, a single pointer for each row is needed.

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)

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)
```

Move to Front (MTF)

- Reduce entropy based on local frequency correlation
- Usually used for BWT before an entropy-encoding step
- Author and detail:
  - Original paper at cs9319/papers
  - http://www.arturocampos.com/ac_mtf.html

Example: abaabacad

Symbol	Code	List
a	0	abcde…..
b	1	bacde…..
a	1	abcde…..
a	0	abcde…..
b	1	bacde…..
a	1	abcde…..
c	2	cabde…..
a	1	acbde…..
d	3	dacbe…..

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

Other ways to reverse BWT

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

- The number of symbols in the substring $V_0 \ldots V_{i-1}$ 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.

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

???????]

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

???????A]

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

??????NA]

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

????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

???NANA]

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

??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

?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

[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

[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

Occ / Rank

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

C [ ]

An illustration

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

A]

A

A

A

B

N

N

[

]

B

N

N

[

A

A

]

A

NA]

A

A

A

B

N

N

[

]

B

N

N

[

A

A

]

A

ANA]

A	B
A	N
A	N
B	[
N	A
N	A
[	]
]	A

NANA]

A	B
A	N
A	N
B	[
N	A
N	A
[	]
]	A

ANANA]

A	B
A	N
A	N
B	[
N	A
N	A
[	]
]	A

BANANA]

A	B
A	N
A	N
B	[
N	A
N	A
[	]
]	A

[BANANA]

A
A
A
B
N
N
[
]

B
N
N
[
A
A
]
A

Dynamic BWT ?

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

Details:

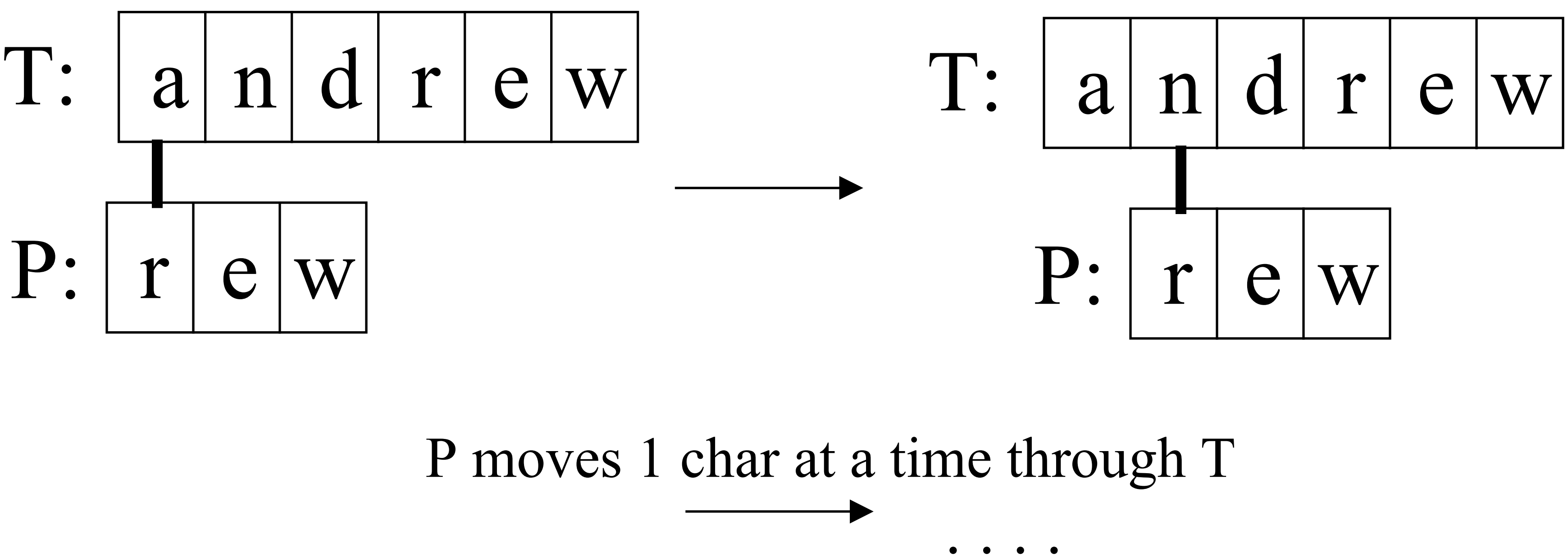
Salson M, Lecroq T, Léonard M and Mouchard L (2009). "A Four-Stage Algorithm for Updating a Burrows–Wheeler Transform". Theoretical Computer Science 410 (43): 4350.

What is Pattern Matching?

- Definition:
  - given a text string T and a pattern string P, find the pattern inside the text
    - T: “the rain in spain stays mainly on the plain”
    - P: “n th”

The Brute Force Algorithm

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



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.

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.)

continued

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

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.

continued

Summary

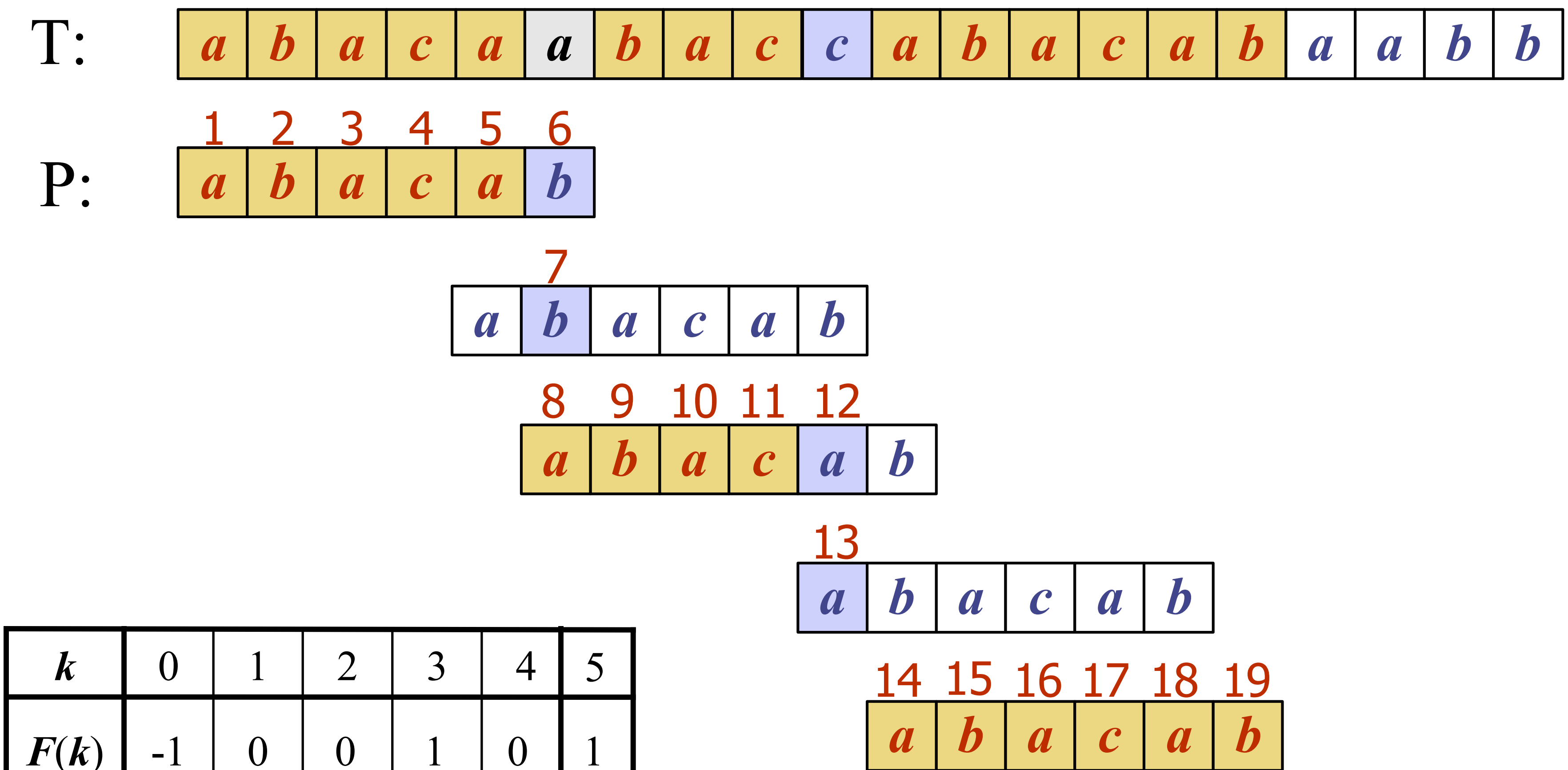
- If a mismatch occurs between the text and pattern P at P[j], what is the *most* we can shift the pattern to avoid wasteful comparis ons?

Summary

- If a mismatch occurs between the text and pattern P at P[j], what is the *most* we can shift the pattern to avoid wasteful comparis ons?

- *Answer*: the largest prefix of P[0 .. j-1] that is a suffix of P[1 .. j-1]

Example



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

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 d evices or through a network stream

KMP Disadvantages

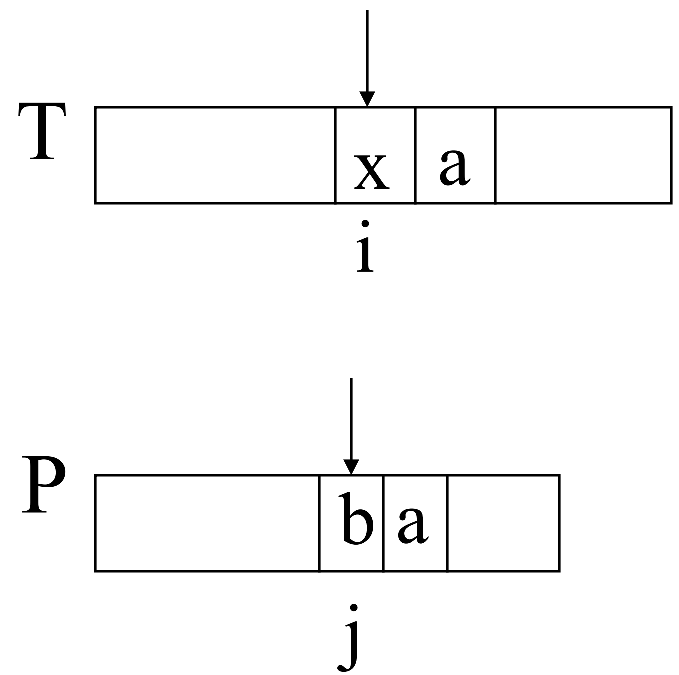
- KMP doesn't work so well as the size of the alphabet increases
  - more chance of a mismatch (more possible mismatches)
  - mismatches tend to occur early in the pattern, but KMP is faster when the mismatches occur later

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

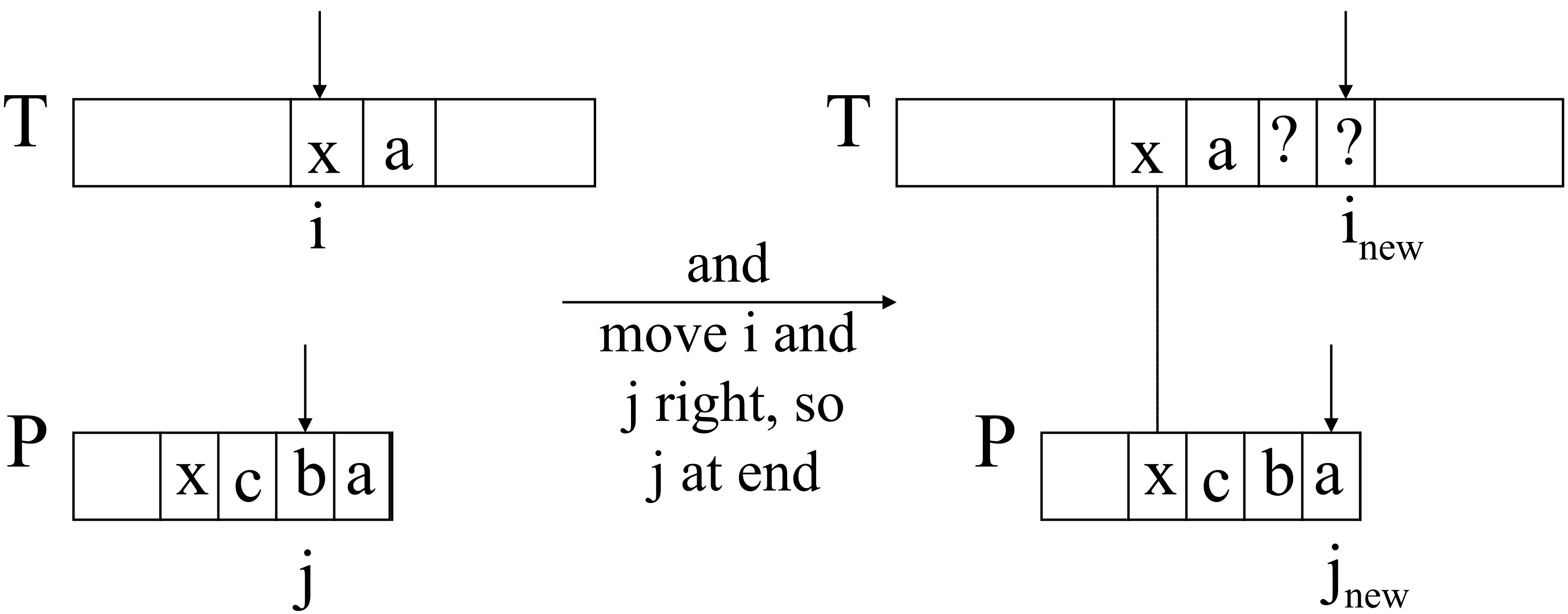
- 2. The *character-jump* technique
  - when a mismatch occurs at T[i] == x
  - the character in pattern P[j] is not the same as T[i]



- There are 3 possible cases.

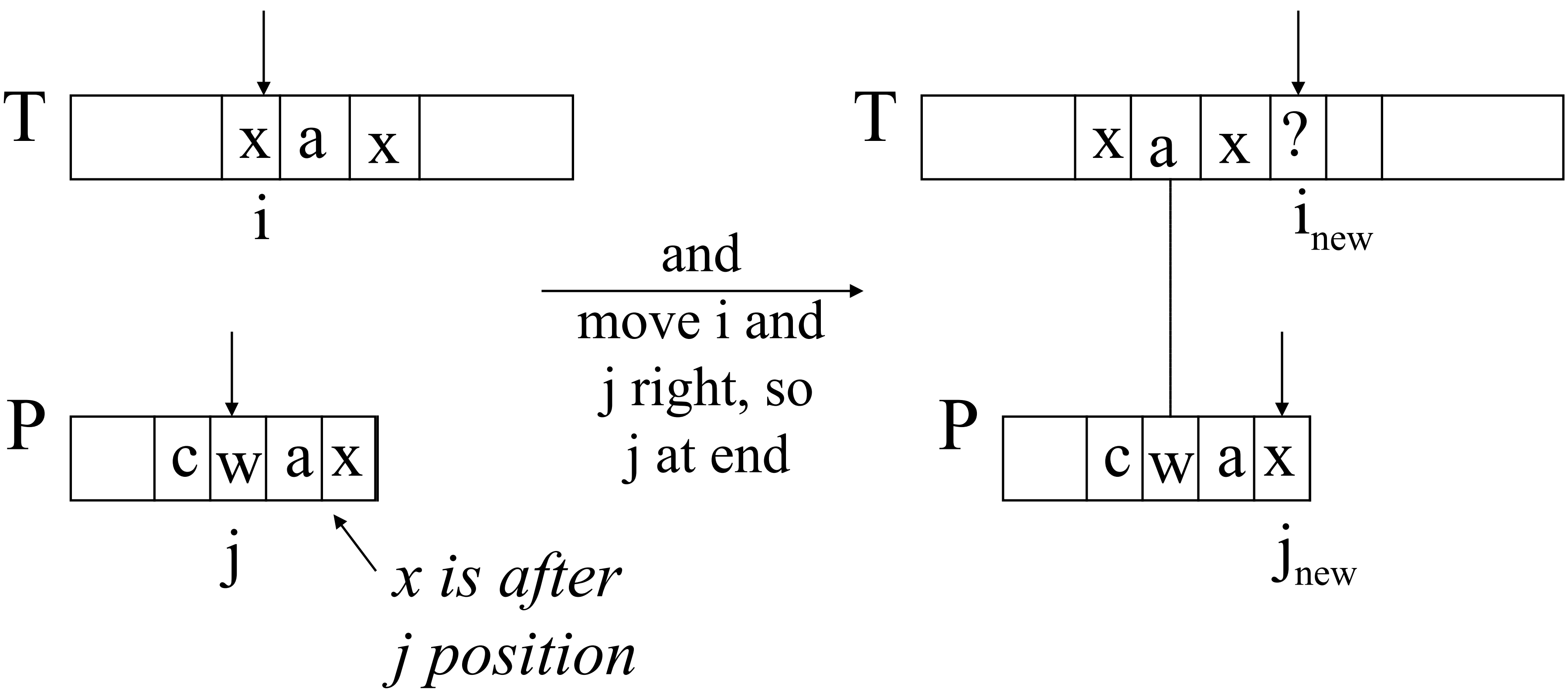
Case 1

- If P contains x somewhere, then try to *shift P* right to align the last occurrence of x in P with T[i].



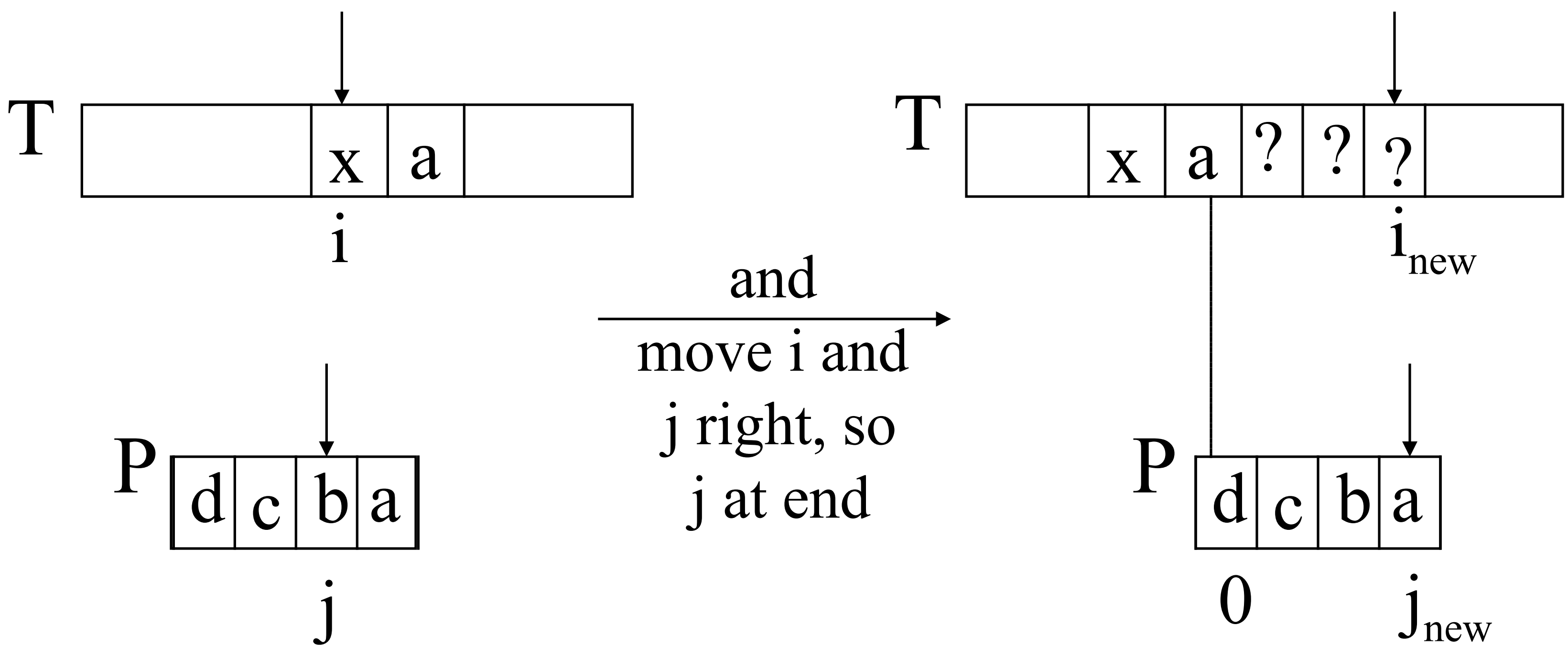
Case 2

- 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].



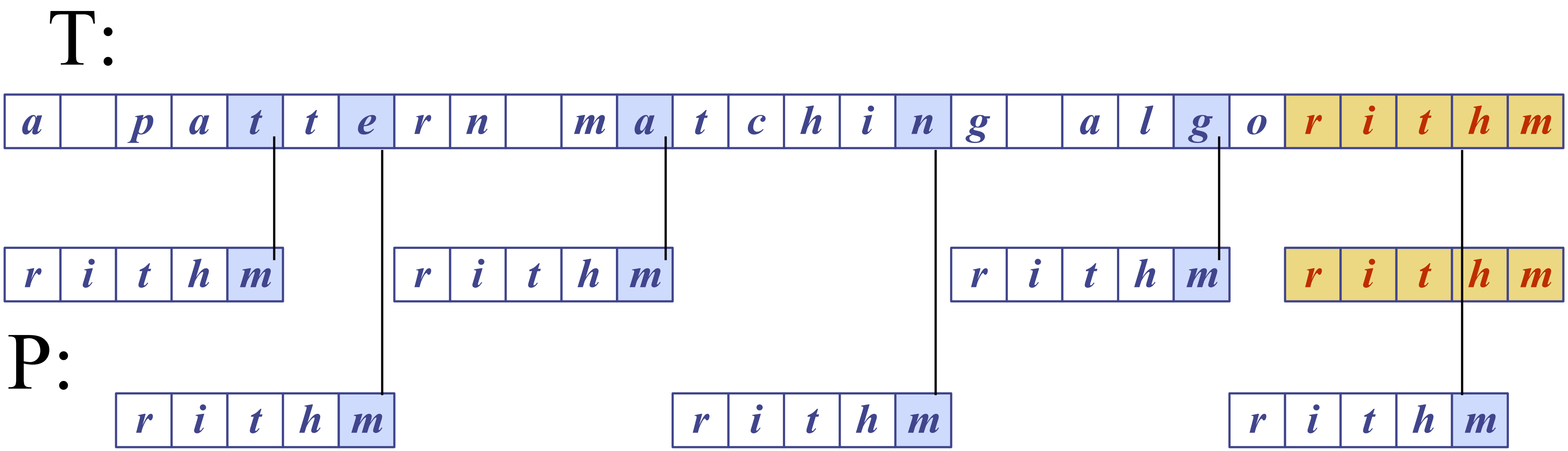
Case 3

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



No x in P

Boyer-Moore Example (1)



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

L() Example

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

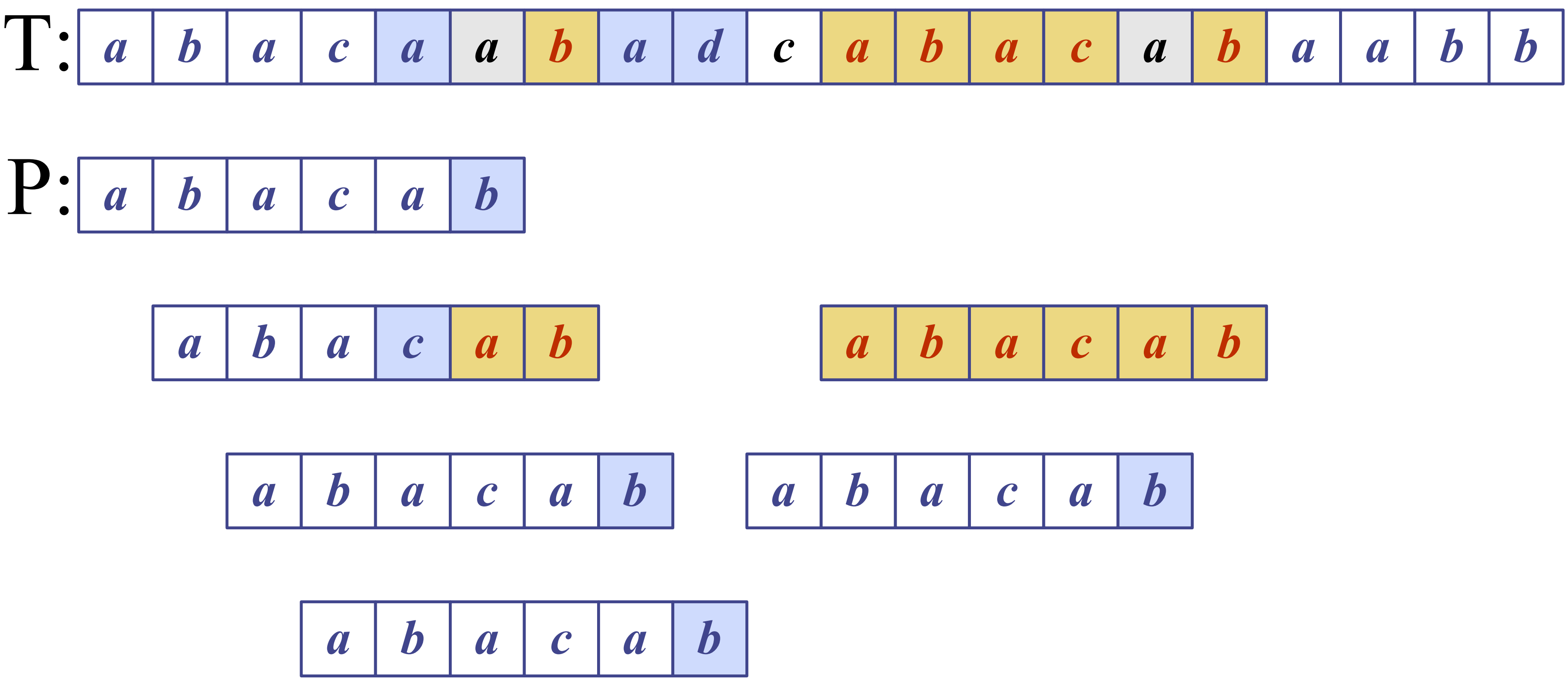
P

a	b	a	c	a	b
0	1	2	3	4	5

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

L() stores indexes into P[]

Boyer-Moore Example (2)



x	*a*	*b*	*c*	*d*
L(*x*)	4	5	3	21

Analysis

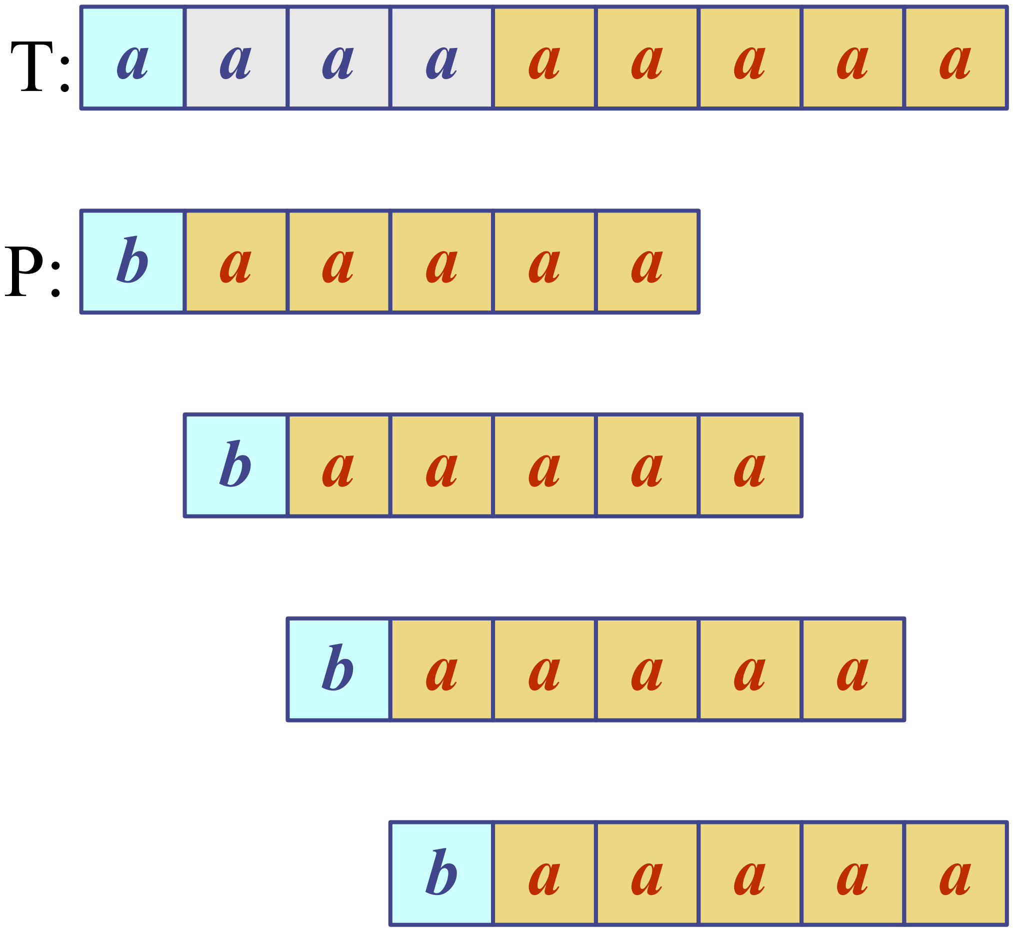
- Boyer-Moore worst case running time is O(nm + A)

- 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

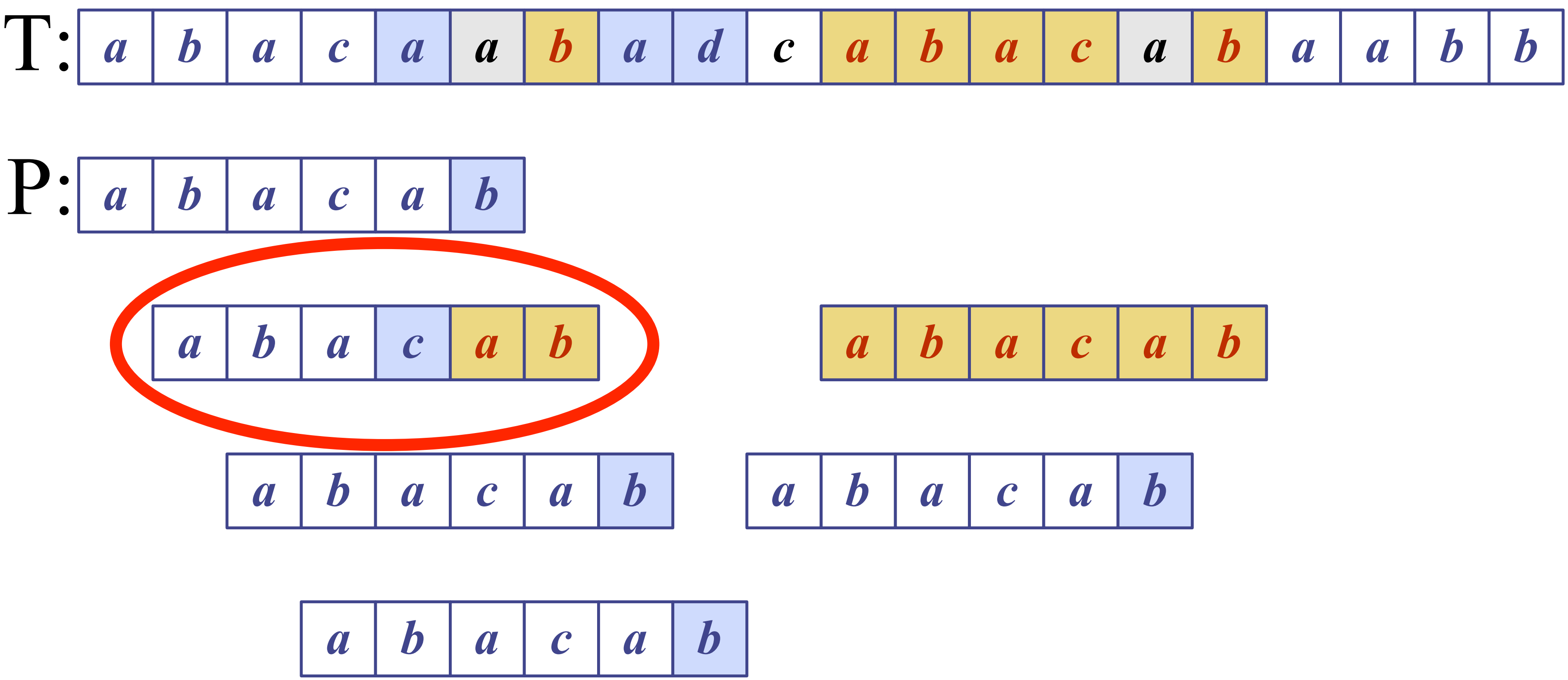
- Boyer-Moore is *significantly faster than brute force* for searching English text.

Worst Case Example

- T: "aaaaa…a"
- P: "baaaaa"



Boyer-Moore Example (2)



x	***a***	***b***	***c***	***d***
***L*(*x*)**	4	5	3	21

Boyer-Moore: Good suffix rule

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

KMP & BM

- Please refer to the original papers (available in the cs9319 website) for the details of the algorithms
- Most text processors use BM for “find” (& “replace”) due to its good performance for general text documents