Design and Analysis of Algorithms
Unit -4

Bharathi R

Department of Computer Science & Engineering

DESIGN AND ANALYSIS OF ALGORITHMS

Unit 4: Space and Time Tradeoffs

Input Enhancement in String Matching- The Boyer-Moore Algorithm

Bharathi R

Department of Computer Science & Engineering

The Boyer-Moore algorithm

Based on the same two ideas:

comparing pattern characters to text from right to left

precomputing shift sizes in **two** tables

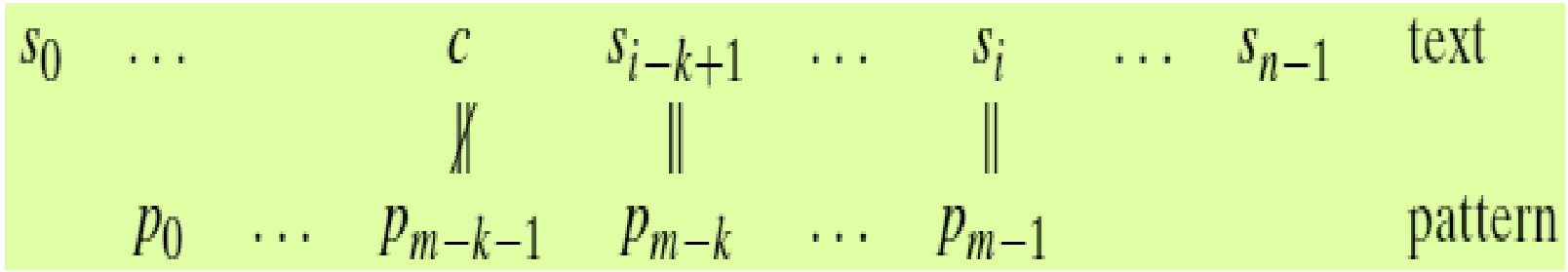
1. ***bad-symbol table*** indicates how much to shift based on text's character causing a mismatch

2. ***good-suffix table*** indicates how much to shift based on matched part (suffix) of the pattern (taking advantage of the periodic structure of the pattern)

Design and Analysis of Algorithms

Bad-symbol shift in Boyer-Moore algorithm

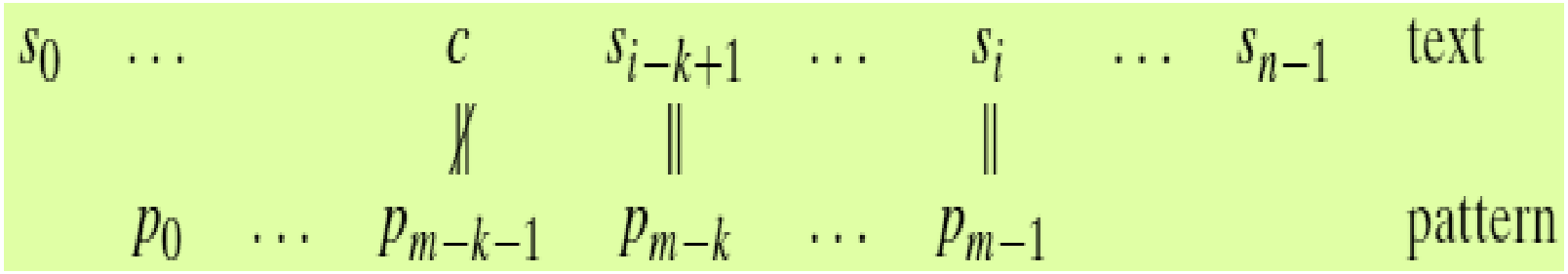
1. If the rightmost character of the pattern doesn't match, BM algorithm acts as Horspool's
2. If the rightmost character of the pattern does match, BM compares preceding characters right to left until either all pattern's characters match or a mismatch on text's character c is encountered after $k > 0$ matches



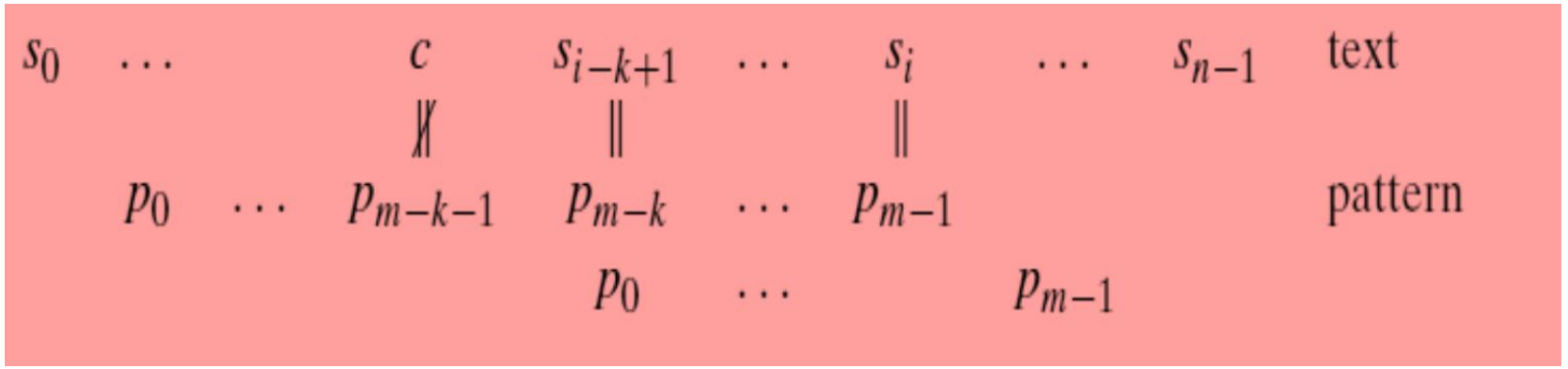
In this situation, the Boyer-Moore algorithm determines the shift size by considering two quantities. The first one is guided by the text's character c that caused a mismatch with its counterpart in the pattern. Accordingly, it is called the ***bad symbol shift***.

Design and Analysis of Algorithms

Bad-symbol shift in Boyer-Moore algorithm



If *c* is not in the pattern, we shift the pattern to just pass this *c* in the text. Conveniently, the size of this shift can be computed by the formula *t*1*(c)* − *k* where *t*1*(c)* is the entry in the precomputed table used by Horspool's algorithm and *k* is the number of matched characters:



Design and Analysis of Algorithms

Bad-symbol shift in Boyer-Moore algorithm- Example

character c	A	B	C	D	E	F	. . .	R	. . .	Z	_
shift $t(c)$	4	2	6	6	1	6	6	3	6	6	6

For example, if we search for the pattern BARBER in some text and match the last two characters before failing on letter S in the text, we can shift the pattern by $t_1(\text{S}) - 2 = 6 - 2 = 4$ positions:

```
s0  . . .          S E R                . . .  sn−1
                   ✗ ‖ ‖
          B A R B E R
                B A R B E R
```

Bad-symbol shift in Boyer-Moore algorithm- Example

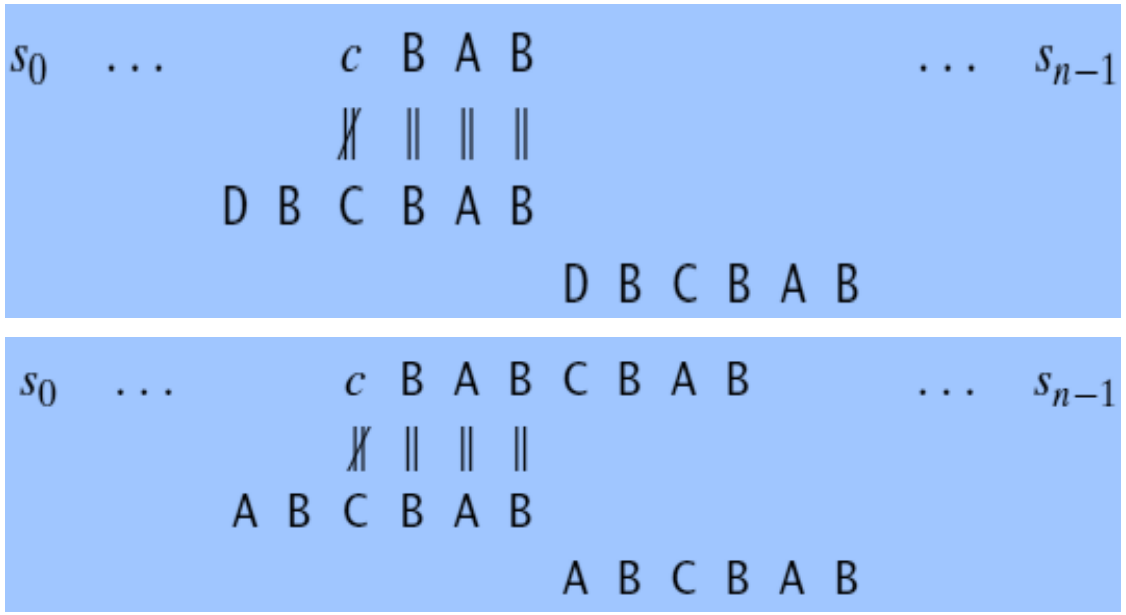
character c	A	B	C	D	E	F	. . .	R	. . .	Z	_
shift $t(c)$	4	2	6	6	1	6	6	3	6	6	6

The same formula can also be used when the mismatching character c of the text occurs in the pattern, provided $t_1(c) - k > 0$. For example, if we search for the pattern BARBER in some text and match the last two characters before failing on letter A, we can shift the pattern by $t_1(\text{A}) - 2 = 4 - 2 = 2$ positions:

```
s0   . . .          A  E  R          . . .   sn−1
                    ✗  ‖  ‖
          B  A  R   B  E  R
                B  A  R  B  E  R
```

$$d_1 = \max\{t_1(c) - k, 1\}$$

Good-suffix shift in Boyer-Moore algorithm and when to use?



Erroneous shift will happen.
To avoid that a shift based on matched suffix of the pattern.
Called "Good Suffix shift". It is denoted by d_2.

Good-suffix shift in Boyer-Moore algorithm

$d_2(k)$ = Distance between matched suffix of size k and its rightmost occurrence in the pattern that is not preceded by the same character as the suffix

k	pattern	d_2
1	ABC$\overline{B}$AB	2
2	$\overline{AB}$CBAB	4
3	$\overline{AB}$CBAB	4
4	$\overline{AB}$CBAB	4
5	$\overline{AB}$CBAB	4

$$d = \begin{cases} d_1 & \text{if } k = 0, \\ \max\{d_1, d_2\} & \text{if } k > 0, \end{cases}$$

where $d_1 = \max\{t_1(c) - k, 1\}$

Good-suffix shift in Boyer-Moore algorithm

- Good-suffix shift d_2 is applied after $0 < k < m$ last characters were matched

- $d_2(k)$ = the distance between (the last letter of) the matched suffix of size k and (the last letter of) its rightmost occurrence in the pattern that is not preceded by the same character preceding the suffix

  Example: `CABABA` $d_2(1) = 4$

- If there is no such occurrence, match the longest part (tail) of the k-character suffix with corresponding prefix;
  if there are no such suffix-prefix matches, $d_2(k) = m$

  Example: `WOWWOW` $d_2(2) = 5$, $d_2(3) = 3$, $d_2(4) = 3$, $d_2(5) = 3$

Boyer-Moore algorithm

Step 1 Fill in the bad-symbol shift table
Step 2 Fill in the good-suffix shift table
Step 3 Align the pattern against the beginning of the text
Step 4 Repeat until a matching substring is found or text ends:
Compare the corresponding characters right to left.
If no characters match, retrieve entry $t_1(c)$ from the bad-symbol table for the text's character c causing the mismatch and shift the pattern to the right by $t_1(c)$.
If $0 < k < m$ characters are matched, retrieve entry $t_1(c)$ from the bad-symbol table for the text's character c causing the mismatch and entry $d_2(k)$ from the good-suffix table and shift the pattern to the right by

$$d = \max \{d_1, d_2\}$$

where $d_1 = \max\{t_1(c) - k, 1\}$.

Design and Analysis of Algorithms

Example of Boyer-Moore alg. application

BAOBAB

c	A	B	C	D	. . .	O	. . .	Z	_
$t_1(c)$	1	2	6	6	6	3	6	6	6

k	pattern	d_2
1	BAO$\overline{\text{B}}$A$\underline{\text{B}}$	2
2	$\overline{\text{B}}$AOB$\underline{\text{AB}}$	5
3	$\overline{\text{B}}$AO$\underline{\text{BAB}}$	5
4	$\overline{\text{B}}$A$\underline{\text{OBAB}}$	5
5	$\overline{\text{B}}$$\underline{\text{AOBAB}}$	5

B E S S _ K N E W _ A B O U T _ B A O B A B S

B A O B A B

$d_1 = t_1(K) - 0 = 6$

B A O B A B

$d_1 = t_1(\_) - 2 = 4$

$d_2 = 5$

$d = \max\{4, 5\} = 5$

B A O B A B

$d_1 = t_1(\_) - 1 = 5$

$d_2 = 2$

$d = \max\{5, 2\} = 5$

B A O B A B

Analysis

- The worst-case efficiency of the Boyer-Moore algorithm is known to be linear.
- Though this algorithm runs very fast, especially on large alphabets (relative to the length of the pattern),many people prefer its simplified versions, such as Horspool's algorithm, when dealing with natural-language–like strings.

References

Chapter 7 ,Introduction to The Design and Analysis of Algorithms by Anany Levitin

THANK YOU

Bharathi R

Department of Computer Science & Engineering

rbharathi@pes.edu