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

Bharathi R
Department of Computer Science & Engineering

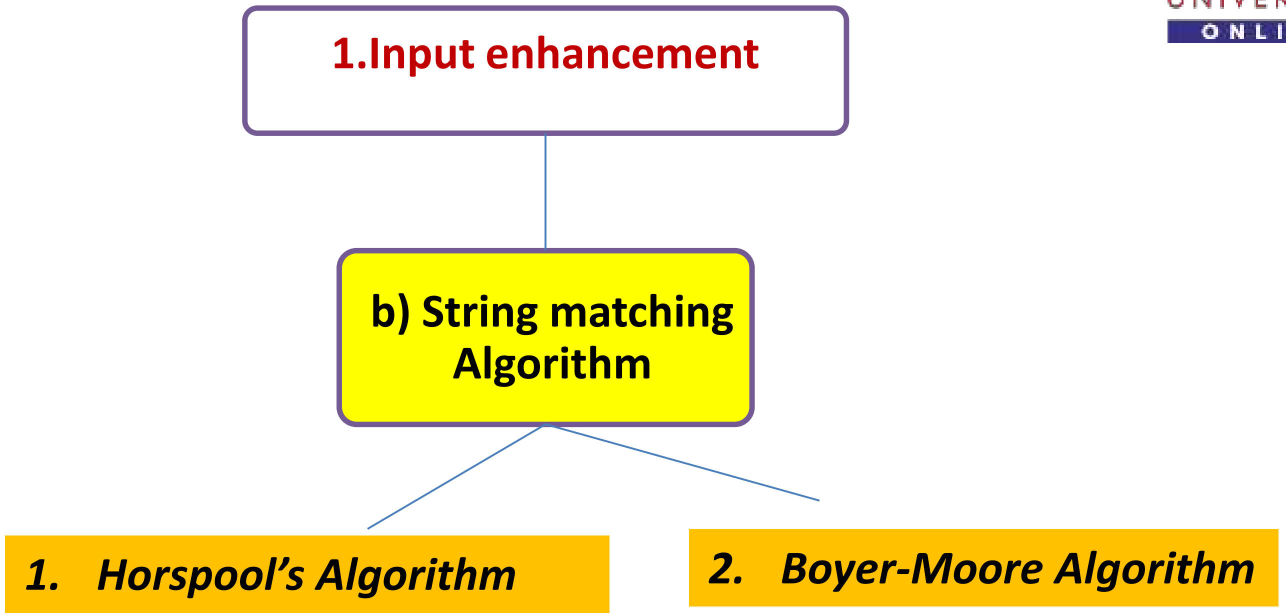
String Matching

String matching requires finding an occurrence of a given string of *m* characters called the ***pattern*** in a longer string of *n* characters called the ***text***.

$$
\begin{array}{cccccccccc}
t_0 & \cdots & t_i & \cdots & t_{i+j} & \cdots & t_{i+m-1} & & \cdots & t_{n-1} \\
 & & \updownarrow & & \updownarrow & & \updownarrow & & & \\
 & & p_0 & \cdots & p_j & \cdots & p_{m-1} & & \text{pattern } P &
\end{array}
$$

Input-enhancement idea:

Preprocess the pattern to get some information about it, store this information in a table, and then use this information during an actual search for the pattern in a given text.



String matching by Brute force

pattern: a string of *m* characters to search for
text: a (long) string of *n* characters to search in

<u>Brute force algorithm</u>

Step 1 Align pattern at beginning of text

Step 2 Moving from left to right, compare each character of pattern to the corresponding character in text until either all characters are found to match (successful search) or a mismatch is detected

Step 3 While a mismatch is detected and the text is not yet exhausted, realign pattern one position to the right and repeat Step 2

Time complexity (worst-case): O(mn)

String matching by preprocessing

Several string searching algorithms are based on the input enhancement idea of preprocessing the **pattern**

1. **Knuth-Morris-Pratt (KMP)** algorithm preprocesses pattern left to right to get useful information for later searching

2. **Boyer -Moore algorithm** preprocesses pattern right to left and store information into two tables

3. **Horspool's algorithm** simplifies the Boyer-Moore algorithm by using just one table

A simplified version of Boyer-Moore algorithm:

Preprocesses pattern to generate a "shift table" that determines how much to shift the pattern when a mismatch occurs

Always makes a shift based on the text's character **c** aligned with the last compared (mismatched) character in the pattern according to the shift table's entry for ***c***

How far to shift?

Look at first (rightmost) character in text that was compared:

1. The character is not in the pattern

 ....c...................... (c not in pattern)

 ≠

 BAOBAB

2. The character is in the pattern (but not the rightmost)

 .....O...................... (O occurs once in pattern)

 BAOBAB

 .....A...................... (A occurs twice in pattern)

 BAOBAB

3. The rightmost characters do match

 .....B......................

 BAOBAB

Shift Table

Shift sizes can be precomputed by the formula

$$t(c) = \begin{cases} \text{the pattern's length } m, \\ \text{if } c \text{ is not among the first } m-1 \text{ characters of the pattern;} \\ \\ \text{the distance from the rightmost } c \text{ among the first } m-1 \text{ characters} \\ \text{of the pattern to its last character, otherwise.} \end{cases}$$

By scanning pattern before search begins and stored in a table called ***shift table***. After the shift, the right end of pattern is *t(c)* positions to the right of the last compared character in text.

Shift table is indexed by text and pattern alphabet Eg: for **BAOBAB**:

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
1	2	6	6	6	6	6	6	6	6	6	6	6	6	3	6	6	6	6	6	6	6	6	6	6	6

Shift Table

$$t(c) = \begin{cases} \text{the pattern's length } m, \\ \text{if } c \text{ is not among the first } m-1 \text{ characters of the pattern;} \\ \\ \text{the distance from the rightmost } c \text{ among the first } m-1 \text{ characters} \\ \text{of the pattern to its last character, otherwise.} \end{cases}$$

ALGORITHM *ShiftTable*$(P[0..m-1])$

//Fills the shift table used by Horspool's and Boyer-Moore algorithms
//Input: Pattern $P[0..m-1]$ and an alphabet of possible characters
//Output: $Table[0..size-1]$ indexed by the alphabet's characters and
// filled with shift sizes computed by formula (7.1)
for $i \leftarrow 0$ **to** $size-1$ **do** $Table[i] \leftarrow m$
for $j \leftarrow 0$ **to** $m-2$ **do** $Table[P[j]] \leftarrow m-1-j$
return *Table*

Horspool Matching

ALGORITHM *HorspoolMatching*$(P[0..m-1], T[0..n-1])$

```
//Implements Horspool's algorithm for string matching
//Input: Pattern P[0..m − 1] and text T[0..n − 1]
//Output: The index of the left end of the first matching substring
//          or −1 if there are no matches
ShiftTable(P[0..m − 1])        //generate Table of shifts
i ← m − 1                      //position of the pattern's right end
while i ≤ n − 1 do
    k ← 0                      //number of matched characters
    while k ≤ m − 1 and P[m − 1 − k] = T[i − k] do
        k ← k + 1
    if k = m
        return i − m + 1
    else  i ← i + Table[T[i]]
return −1
```

Horspool Matching -Example

$s_0 \quad \ldots \qquad\qquad\qquad c \quad \ldots \quad s_{n-1}$

```
        B A R B E R
```

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

```
J I M _ S A W _ M E _ I N _ A _ B A R B E R S H O P
B A R B E R                 B A R B E R
        B A R B E R             B A R B E R
          B A R B E R                 B A R B E R
```

Example of Horspool's algorithm

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	_
1	2	6	6	6	6	6	6	6	6	6	6	6	6	3	6	6	6	6	6	6	6	6	6	6	6	6

```
BARD LOVED BANANAS
BAOBAB
      BAOBAB
           BAOBAB
                  BAOBAB  (unsuccessful search)
```

References

"Introduction to the Design and Analysis of Algorithms", Anany Levitin, Pearson Education, Delhi (Indian Version), 3rd edition, 2012. Chapter- 7

THANK YOU

Bharathi R

Department of Computer Science & Engineering

rbharathi@pes.edu