CHAPTER 06 – PART II
STRING SEARCH ALGORITHMS

Data Structures and Algorithms (17ECSC204)

Prakash Hegade
School of CSE, KLE TECH

2017-18

Brute Force String Search Algorithm:

A brute force string matching algorithm is quite obvious. Align the pattern against the first **m** characters of the text and start matching the corresponding pairs of characters from left to right until either all **m** pairs of characters match or a mismatching pair is encountered. In the latter case, shift the pattern one position to the right and resume character comparisons, starting again with the first character of the pattern and its counterpart in the text.

Example:

Text: N O B O D Y_S A W _ M E

Pattern: S A W

```
N O B O D Y_S A W _ M E
S A W
  S A W
    S A W
      S A W
        S A W
          S A W
            S A W
              S A W
```

ALGORITHM BruteForceStringMatch(T[0... n-1], P[0... m-1])

// Implements brute force string match

// Input: An array T[0... n-1] of n characters representing a text and an array P[0... m-1] of

// m characters representing a pattern

// Output: The index of the first character in the text that starts a matching substring or

// -1 if the search is unsuccessful

```
for i← 0 to n-m do
    j ← 0
    while j < m and P[ j ] = T [ i + j ] do
        j ← j + 1
    if j = m
        return i
return -1
```

The worst case would be that the algorithm would have to make all the m comparisons before shifting the pattern and this can happen for n-m+1 tries.

Thus, in the worst case, the algorithm efficiency is in **O(nm)**.

Boyer-Moore Algorithm:

Algorithm involves constructing two tables:

Bad symbol shift table:

Given a character c, T(c) is computed as:

→ the pattern length m, if c is not among the first m-1 characters of the pattern

→ the distance from the rightmost c among the first m-1 characters of the pattern to its last character, otherwise.

The size of shift is computed by: $T(c) - k$

where k is number of matched characters.

If $T(c) - k$ is $<= 0$, then shift by one position to right.

Good Suffix shift Table:

→ check if there is another occurrence of matched pattern not preceded by same character as in its last occurrence. d2 is distance between such second rightmost occurrence of pattern & its rightmost occurrence.

→ If not, find the longest prefix of size t, $t < k$ (k = matched pattern length), that matches the suffix of same size t. If such a prefix exists, shift d2 is computed as the distance between

this prefix & the corresponding suffix.

→ otherwise d2 is set to pattern length m.

The final shift distance d is computed by:

$$d = \begin{cases} d1 & \text{if } k = 0 \\ \max\{d1, d2\} & \text{if } k > 0 \end{cases}$$

where $d1 = \max\{T(c) - k, 1\}$

Examples:

1. Construct bad symbol shift table for the pattern BARBER

c	A	B	C	D	E	F	...	R	...	2	-
T(c)	4	2	6	6	1	6	6	3	6	6	6

2. Construct a good suffix shift table for:

ABCBAB

k	pattern	d2
1	ABCBA$\underline{B}$	2
2	$\overline{A}$BCB$\underline{AB}$	4
3	$\overline{AB}$CB$\underline{AB}$	4
4	$\overline{AB}$C$\underline{BAB}$	4
5	$\overline{AB}$C$\underline{BAB}$	4

3. Apply Boyer-Moore algorithm on the given Text & pattern.

Text: BESS-KNEW-ABOUT-BAOBABS"

Pattern: BAOBABS

Bad Symbol table:

c	A	B	C	D	...	O	...	Z	-
T(c)	1	2	6	6	6	3	6	6	6

Good suffix table:

k	pattern	d_2
1	BAOB̄AB	2
2	B̄AOBAB	5
3	B̄AOBAB	5
4	B̄AOBAB	5
5	B̄AOBAB	5

Iterations:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
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																	
									X	✓	✓											
	$d_1 = 6-0=6$					B	A	O	B	A	B											
						$d_1 = 6-2=4$									X	✓						
						$d_2 = 5$																
						$d = \max\{4,5\} = 5$																
											B	A	O	B	A	B						
											$d_1 = 6-1=5$											
											$d_2 = 2$											
											$d = \max\{5,2\} = 5$											
																✓	✓	✓	✓	✓	✓	✓
																B	A	O	B	A	B	-
																match.						

4. Construct Bad Symbol Table for the pattern:

CONSISTING

c	A	B	C	...	G	H	I	...	N	O	...	S	T	...	Z	-
T(c)	10	10	9	10	10	10	2	10	1	8	10	4	3	10	10	10

A Better Understable table:

C	O	N	S	I	S	T	I	N	G
9	8				4	3	2	1	10

5. Construct bad symbol table for the pattern:

DISGUSTING

D	I	S	G	U	S	T	I	N	G
9			6	5	4	3	2	1	

6. Construct bad symbol table and good-suffix table for: 00001

c	0	1
T(c)	1	5

k	pattern	d2
1	0000<u>1</u>	5
2	000<u>01</u>	5
3	00<u>001</u>	5
4	0<u>0001</u>	5

76. Compute the two shift tables for the pattern string: 10000

Bad-Symbol table

c	0	1
T(c)	1	4

Good Suffix table

k	pattern	d2
1	10000	3
2	10000	2
3	10000	1
4	10000	5

Prefixes & Suffixes:

Text: School

Prefixes: S
Sc
Sch
Scho
Schoo
School

Suffixes: School
chool
hool
ool
ol
l

For the string "school": S, Sc, Sch, Scho, Schoo are proper prefixes. Similar explanation holds good for proper suffixes too.

i.e; A proper prefix or proper suffix of a string is not equal to the string itself.

Knuth- Morris- Pratt Algorithm:

Principle to Generate the prefix table π:

"Find the length of the longest proper prefix in the Subpattern that matches a proper suffix in the same Subpattern."

Example:

Pattern : ABAB

Char	A	B	A	B
index	0	1	2	3
value	0	0	1	2

a) Substring : A
proper prefix : NULL
proper suffix : NULL

b) Substring : AB
proper prefix = A
proper suffix = B

c) Substring : ABA
proper prefix : A, AB
proper Suffix : BA, A

d) Substring : ABAB
proper prefix : A, AB, ABA
proper Suffix : BAB, AB, B

Shift is computed by the formula:

$k - \pi[k-1]$ where k is number of matched chars.

Eg:- If $k = 2$ then shift $= 2 - \pi[2-1]$

$= 2 - \pi[1]$

$= 2 - 0$

$= 2.$

Example:

1. Seach for pattern ababaca in the text bacbababababacaab using KMP algorithm.

Filling the prefix table π, for the pattern:

char	a	b	a	b	a	c	a
index	0	1	2	3	4	5	6
value	0	0	1	2	3	0	1

Substring	Proper Prefix	Proper Suffix	π-value
a	NULL	NULL	0
ab	a	b	0
aba	a, ab	a, ba	1
abab	a, ab, aba	b, bab, bab	2
ababa	a, ab, aba, abab	a, ba, aba, baba	3
ababac	a, ab, aba, abab, ababa	c, ac, bac, abac, babac	0
ababaca	a, ab, aba, abab, ababa, ababac	a, ca, aca, baca, abaca, babaca	1

Tracing:

i> b a c b a b a b a c a a b
   ≠
   a b a b a c a

shift by 1

ii> b a c b a b a b a c a a b
      || ≠
      a b a b a c a

k = 1

shift = 1 - π[0]

= 1 - 0 = 1

iii> b a c b a b a b a c a a b
        ≠
        a b a b a c a

shift by 1

iv> b a c b a b a b a c a a b
          ≠
          a b a b a c a

shift by 1

v> b a c b a b a b a b a c a a b
          || || || ≠
          a b a b a c a

k = 5

shift = 5 - π[4] = 5 - 3 = 2

vi> b a c b a b a b a c a a b
              || || || ||
              a b a b a c a

- match found.

2. Apply KMP for pattern AABAAB in the text AABBABAABBAAABA.

Efficiency Analysis:

1. When searching for the first occurrence of the pattern in Boyer-Moore algorithm, the worst case efficiency is known to be linear.

If implemented as presented in original paper has worst case running time of $O(n+m)$ only if the patter does not appear in text. When appears, the worst case is $O(nm)$.

2. KMP Analysis

$O(m)$ to Compute prefix function values

$O(n)$ to compare patter to text

Total = $O(n+m)$

Space & Time tradeoff Technique:

A way of solving a problem in less time by using more storage or by solving a problem in very little space by spending a long time.

Examples:

- Boyre Moore algorithm
- Hashing.

The most common situction is an algorithm involving a lookup table.

* —— * —— * —— * —— *