# 5. Sub String Search Algorithms

DATA STRUCTRES AND ALGORITHMS [19ECSC201]

PRAKASH HEGADE

SCHOOL OF CSE | KLE Tech

### The Idea:

We have a text of huge size. It can be a file with Tera Bytes of data. Also, we have got a pattern. We need to search for the pattern, if at all it is present in the text. Is there an occurrence at all? Or are there multiple occurrences?

Searching for a pattern in the text is not an easy job. The simple way is to compare and shift to right by one position if no match is found. That exactly is a brute force technique.

## Brute force String Search

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

This algorithm involves constructing two tables. One is Bad symbol shift table and other one is good Suffix Shift table.

**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 occurrence, otherwise.

The size of the shift is computed by T(c) – k, where k is the number of matched characters. If this result turns out to be less than or equal to 0, then we shift by one position to right.

**Good Suffix Shift Table:**

The steps to be followed are:
- Check if there is another occurrence of matched pattern not preceded by same character as in its last occurrence. D2 is the distance between such rightmost occurrence of pattern and its rightmost occurrence.
- If not, find the longest prefix of size l, l<k, where k is matched pattern length, that matches the suffix of the same size l. If such a prefix exists, shift d2 is computed as the distance between the prefix and the corresponding suffix
- Otherwise d2 is set to pattern length m

The final distance D is computed by the formula:

$$D = \begin{cases} D1 & \text{if } k = 0 \\ \max\{D1, D2\} & \text{if } k > 0 \end{cases}$$

where D1 = max{T(c)-k, 1}

Let us check out some examples.

1. Construct bad symbol shift table for the pattern:

BAD

| c | B | A | D |
|---|---|---|---|
| T(c) | 2 | 1 | 3 |

How did we arrive at this table? Here is the logic. We are only supposed to see the first m-1 characters provided that the length of the pattern is m.

Then, here is how we set the length for each of the character:
- As D is the mth character, set it with length of the pattern which is 3
- Letter A is 1 character away from last character
- Letter B is 2 characters away from the last character

2. Construct a bad symbol shift table for the pattern:
GOOD

| c | G | O | O | D |
|---|---|---|---|---|
| T(c) | 3 |  | 1 | 4 |

How did we arrive at those numbers?

- D, is the mth character and we put the length of the pattern
- O is one character away from D
- We have already covered O. Also meaning that we are only going to look for right most occurrence if there is a repetition.
- G is three characters away from D.

3. Construct a bad symbol shift table for the pattern BARBER

| c | B | A | R | B | E | R |
|---|---|---|---|---|---|---|
| T(c) |  | 4 | 3 | 2 | 1 |  |

4. Construct a good suffix shift table for the ABCBAB

For this table like explained in the beginning we are going to fill up the three part way.

| k | pattern | D2 |
|---|---|---|
| 1 | ABCBAB | 2 |
| 2 | ABCBAB | 4 |
| 3 | ABCBAB | 4 |
| 4 | ABCBAB | 4 |
| 5 | ABCBAB | 4 |

5. Apply Boyer-Moore algorithm on the given text and pattern.
Text: BESS_KNEW_ABOUT_BAOBABS
Pattern: BAOBAB

The length of the pattern string is 6.

Let us first create a bad symbol table:

| c | A | B | C | D | … | O | … | Z | _ |
|---|---|---|---|---|---|---|---|---|---|
| T(c) | 1 | 2 | 6 | 6 | 6 | 3 | 6 | 6 | 6 |

We basically cover everything that is going to come up in the text. _ is basically used for space for better representation. A … represents everything between the mentioned neighbours.

Let us now create the good suffix table.

## 5. Sub String Search Algorithms

| k | pattern | D2 |
|---|---|---|
| 1 | BAOBAB | 2 |
| 2 | BAOBAB | 5 |
| 3 | BAOBAB | 5 |
| 4 | BAOBAB | 5 |
| 5 | BAOBAB | 5 |

Using both the tables, now let us apply the Boyer-Moore algorithm.

| 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 |
|  |  |  |  |  | **x** |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| B | A | O | B | A | **B** |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| D1 = T(K) - k<br>= 6 – 0 = 6<br>Shift by 6 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  | **x** | √ | √ |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  | B | A | O | **B** | **A** | **B** |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  | D1 = T(_) - k = 6-2 = 4<br>D2 = 5<br>D = max{ 4, 5 } = 5<br>Shift by 5 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | **x** | √ |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  | B | A | O | B | **A** | **B** |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  | D1 = T(_) - k = 6-1 = 5<br>D2 = 2<br>D = max{ 5, 2 } = 5<br>Shift by 5 |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | √ | √ | √ | √ | √ | √ |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | **B** | **A** | **O** | **B** | **A** | **B** |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | Match found at position 16 |  |  |  |  |  |  |

6. Construct a bad symbol table for the pattern CONSISTING

| c | C | O | N | S | I | S | T | I | N | G |
|---|---|---|---|---|---|---|---|---|---|---|
| T(c) | 9 | 8 |  |  |  | 4 | 3 | 2 | 1 | 10 |

7. Construct a bad symbol table for the pattern DISGUSTING

| c | D | I | S | G | U | S | T | I | N | G |
|---|---|---|---|---|---|---|---|---|---|---|
| T(c) | 9 |  |  | 6 | 5 | 4 | 3 | 2 | 1 |  |

As the examples very much dictate, now the creation of both the tables must be simple and easy. Let us now do examples to create both the tables for few more examples.

8. Construct bad symbol table and good suffix table for the pattern: 00001

Length of the pattern is: 5

Bad symbol table:

| c | 0 | 1 |
|---|---|---|
| T(c) | 1 | 5 |

Good suffix table:

| k | pattern | D2 |
|---|---|---|
| 1 | 00001 | 5 |
| 2 | 00001 | 5 |
| 3 | 00001 | 5 |
| 4 | 00001 | 5 |

9. Construct bad symbol table and good suffix table for the pattern: 10000

Length of the pattern is: 5

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 and Suffixes:**

For a string school:

Prefixes are:

s

sc

sch

scho

schoo

school

and Suffixes are:
l
ol
ool
hool
chool
school

For the string s, sc, sch, scho, schoo, school are all proper prefixes. Similar explanation holds good for a proper suffixes too. A proper prefix or a proper suffix of a string is all the prefix or suffix other than the string itself.

### Knuth-Morris-Pratt Algorithm

The principle behind the algorithm that is to generate the prefix table P is that:
Find the length of the longest proper prefix in the subpattern that matches a proper suffix in the same subpattern.

Let us understand through creation of a Prefix Table P using an example.

Pattern: ABAB

| **Substring** | **Proper Prefixes** | **Proper Suffixes** | **Longest Match** | **Length** |
|---|---|---|---|---|
| A | NULL | NULL | NULL | 0 |
| AB | A | B | NULL | 0 |
| ABA | A, AB | A, BA | A | 1 |
| ABAB | A, AB, ABA | B, AB, BAB | AB | 2 |

Prefix Table P:

| | | | | |
|---|---|---|---|---|
| char | A | B | A | B |
| index | 0 | 1 | 2 | 3 |
| value | 0 | 0 | 1 | 2 |

Now with the help of this generated table, we can apply the algorithm. The formula to compute the shift is given by: k – P[k-1]

For example, if k is the number of matched characters then the shift is computed by:

$$\begin{aligned}\text{Shift} &= 2 - P[2-1]\\ &= 2 - P[1]\\ &= 2 - 0\\ &= 2\end{aligned}$$

Let us see a more detailed example.

2. Search for the pattern ababaca in the text bacbababacaab using KMP algorithm.
Let us first create the prefix table P.

The prefix table will be populated based on the principle stated by the algorithm.

Pattern: abababaca

| Substring | Proper Prefixes | Proper Suffixes | Longest Match | Length |
|---|---|---|---|---|
| a | NULL | NULL | NULL | 0 |
| ab | a | b | NULL | 0 |
| aba | a, ab | a, ba | a | 1 |
| abab | a, ab, aba | b, ab, bab | ab | 2 |
| ababa | a, ab, aba, abab | a, ba, aba, baba | aba | 3 |
| ababac | a, ab, aba, abab, ababa | c,ac, bac, abac, babac | NULL | 0 |
| ababaca | a, ab, aba, abab, ababac | a, ca, aca, baca, abaca, babaca | a | 1 |

Prefix Table P:

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

Now using the generated table we can search for the pattern

Tracing:

Iteration 01:

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| b | a | c | b | a | b | a | b | a | c | a | a | B |
| x | | | | | | | | | | | | |
| a | b | a | b | a | c | a | | | | | | |

Shift by 1

Iteration 02:

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| b | a | c | b | a | b | a | b | a | c | a | a | B |
| | √ | x | | | | | | | | | | |
| | a | b | a | b | a | c | a | | | | | |

k = 1

Shift k – P [k-1] = 1 – P[0] = 1

So, shift by 1.

Iteration 03:

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| b | a | c | b | a | b | a | b | a | c | a | a | B |
| | | x | | | | | | | | | | |
| | | a | b | a | b | a | c | a | | | | |

Shift by 1.

Iteration 04:

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| b | a | c | b | a | b | a | b | a | c | a | a | B |
|  |  |  | x |  |  |  |  |  |  |  |  |  |
|  |  |  | a | b | a | b | a | c | a |  |  |  |

Shift by 1.

Iteration 05:

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| b | a | c | b | a | b | a | b | a | c | a | a | B |
|  |  |  |  | √ | √ | √ | √ | √ | √ | √ |  |  |
|  |  |  |  | a | b | A | b | a | c | a |  |  |

Match found at index 4.

### 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 the original paper, it has worst case running time of O(m+n) only if the pattern does not appear in text. When appears, the worst case in O(mn)

2. For KMP, we need
O(m) to compute prefix function values
O(n) to compute pattern
And the total is O(n+m)

**Please refer class notes and handout for Rabin-Karp Algorithm.**

~*~*~*~*~*~*~