MULTIMEDIA UNIVERSITY®

TCP2101 Algorithm Design & Analysis

Assignment

Group: TT06

Question 2: Perform a comparative analysis between naïve, Boyer-Moore and Horspool implementation in string matching

Prepared by:

Choe Chooon Ho - 1132702963

Wong Tiong Kiat – 1132702943

Tutor:

William Ban

Introduction

We implemented 3 different string matching algorithms which are:

1. Naïve method (brute force)
2. Boyer-Moore (bad character rule)
3. Boyer-Moore-Horspool (bad character table)

Boyer-Moore horspool is just another variance/extension of boyer-moore which uses a bad character table as a reference to how many skips for each character mismatch.

Description of Algorithm

Naive

We implemented naïve method of string matching, which is just double, first loop is to loop (i) through the text, second loop(j) searches the pattern text to the matching text from index of i until the end of pattern text, the i loops end at the matching text.

Boyer-Moore

Boyer-Moore learn from character comparisons to skip pointless alignments.

Boyer-Moore follow a **bad character rule** which this rule started that upon mismatch skip alignments until one of the two condition hold trues, where the conditions state:

- The pattern is found in the matching string
- P moves past the mismatched characters (shifting to skip all the unnecessary loops)

The number is skip done by looping from the right to the left. Let's say the current character being matched is index i, and your pattern is currently matched until j character, where match.at(i) != pattern.at(j). Therefore you loop j until you find match.at(i), where the number of skips will be position at i – position at j.

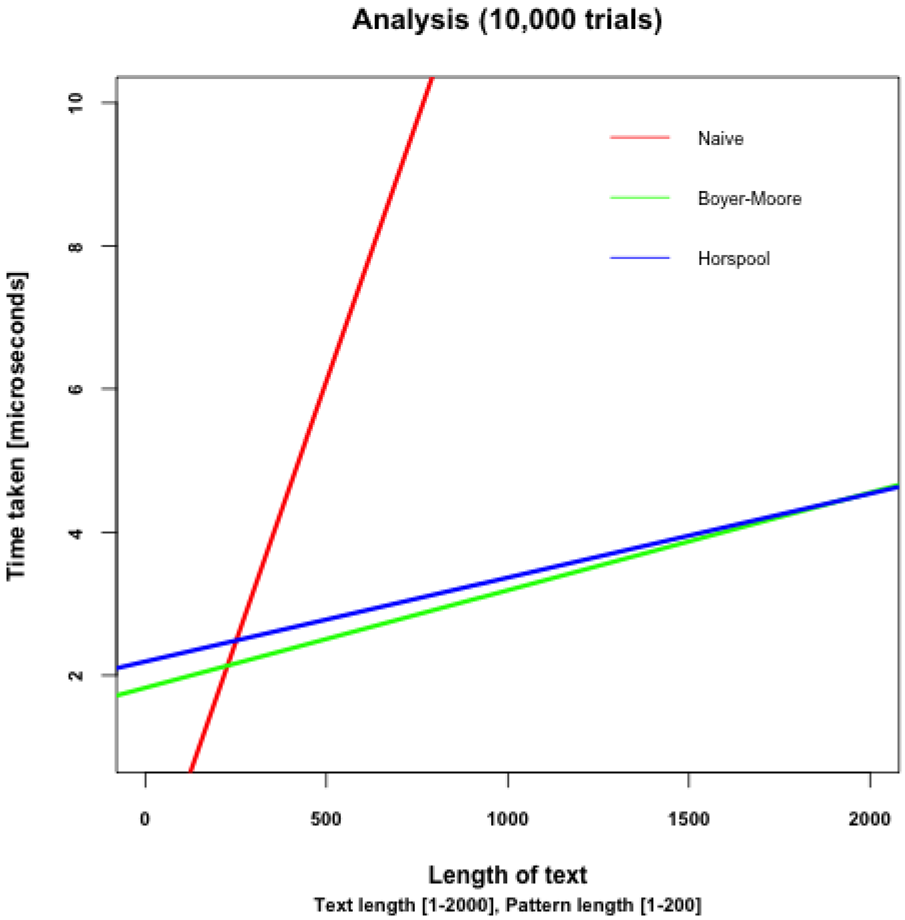
Boyer-Moore Horspool

Simplified implementation of boyer-moore.

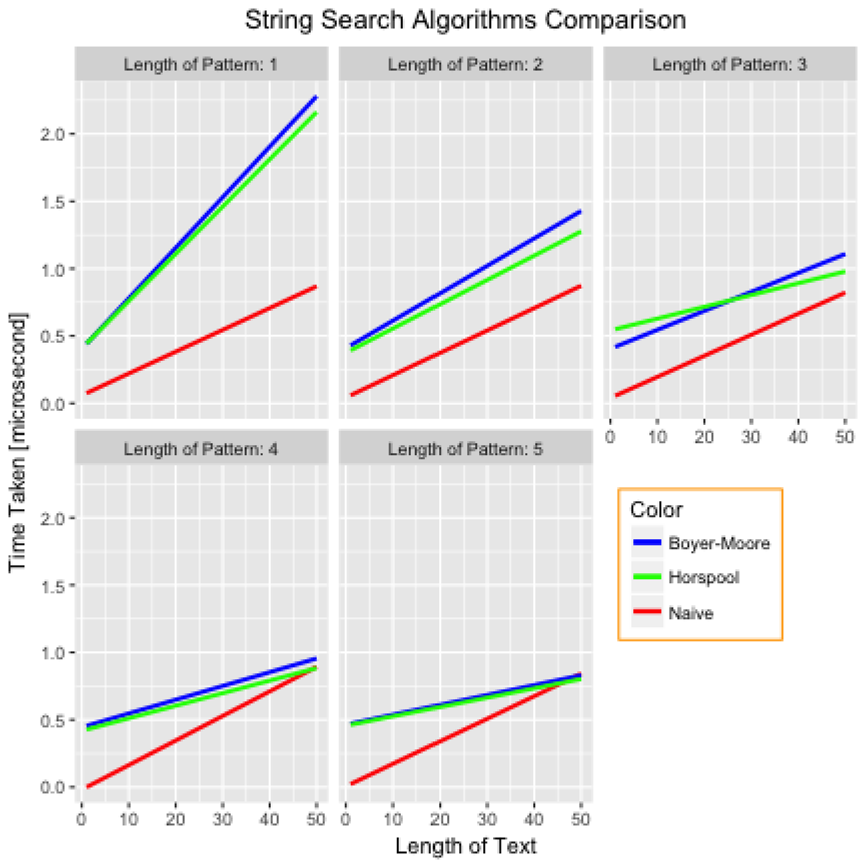
Bad match table (reference table) is constructed based on the pattern text using the formula value = length – index – 1.

Loop through the matching string, for each of character reference of the matching string, refer to the bad match table, for how many characters the loop should shift to for each mismatch. Note that bad match table is only referenced when there is a need for shift of index i.

Results Based on Experiments



We can tell from experiments we plotted using R’s ggplot2 that naïve has the worst case, where horspool takes longer in the starting, due to the construction time of the bad character table. However it will run faster than boyer-moore at one point.



Naïve will run the fastest initially, because it doesn't require any construction or pre-processing. However naïve method become slower as length of text and pattern gets bigger.

All of the graphs above shows that Horspool will run faster than the original implementation of Boyer-Moore as the length of matching text increases.

Conclusion!

Worst case

Naive String Search (Brute Force)

vs

Boey-Moore

vs

Boyer-Moore-Horspool’s String Search

Worst Case Analysis using Asymptotic Analysis

Input: text, n, pattern, m
text, the string to be checked
n, the length of text
pattern, the pattern to be found
m, the length of pattern

1. Naive Method (Brute Force):

```
for i <- 0 to n-m do                                  n
        j <- 0                                        n
        for j to m-1 do                               n * m
                if text[i+j] != pattern[j] then    n * m
                        break                         n
        if j = m then                                 n
                match                                 n
                                                      O(n * m)
```

2. Boyer-Moore Method:

```
// Create last occurrence table.
for i <- 0 to m-1 do                                   m
    table[pattern[i] <- i                              m
table[others] <- -1

i <- m-1                                     1
j <- m-1                                     1

while i < n do                                         n
    if text[i] = pattern[j] then                       n
        if j = 0 then                                  n
            match                                      n
        else                                           n
            i <- i - 1                       n * m
            j <- j - 1                       n * m
    else                                               n
        i = i + m - min(j, table[text[i]]+1) n
        j = m - 1                            n
                                                       O(n * m)
```

3. Boyer-Moore-Horspool Method:

```
// Create a mismatch table.
for i <- 0 to m-2 do                                   m
    table[pattern[i]] <- m-i-1                         m
table[others] <- m


i <- m-1                                     1
while i < n do                                         n
    t <- m-1                                 n
    for j <- i to i-m+1 do                             n * m
        if text[j] != pattern[t] then                  n * m
            break                                      n
        t <- t - 1                           n * m
    if t = -1 then                                     n
        match                                          n
    i <- i + table[text[i]]                  n
                                                       O(n * m)
```

Therefore, we can conclude that Boyer-Moore-Horspool’s worst case is as bad as the naive method.

Boyer-Moore-Horspool’s algorithm would cost more operations and memory for the need of constructing the mismatch table.

Best case

Naive String Search (Brute Force)

vs

Boyer-Moore-Horspool’s String Search

Best Case Analysis using example

1. Naive Method (Brute Force):

```
// Text: ABCABCABC, Pattern: 000

ABCABCABC
0 -> break, go next

ABCABCABC
 0 -> break, go next

ABCABCABC
  0 -> break, go next and so on.

This results in 7 comparisons,
which is 9 - 3 + 1 = 7 => n - m + 1 => n - m
Therefore, Omega(n)
```

2. Boyer-Moore and Boyer-Moore-Horspool’s Method:

```
// Text: ABCABCABC, Pattern: 000

+------------+——-+——-+
| character  | 0 | * |
+------------+—--+---+
| skip value | 1 | 3 |
+------------+---+---+

// Table for pattern, 000.
// All other characters NOT 0 will be m, the length of the pattern, 3

ABCABCABC
  0 -> increment m times, 3

ABCABCABC
    0 -> increment 3

ABCABCABC
       0 -> increment 3

This results in 3 comparisons,
which is 9 / 3 = 3 => floor(n / m)
Therefore, Omega(n / m)
```

Average case

Average case can be proven by the graph on top.

Finally we can conclude that, the overall experiments show that horspool implementation of Boyer-Moore is easier to implement and it’s faster than the original Boyer-Moore because of the reference table.