Implementation and Analysis of KMP, Boyer-Moore, and Regex For Searching IF2211 Algorithm Strategies Papers

Kevin Fernaldy 13516109
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
13516109@std.stei.itb.ac.id

***Abstract*—String matching is a common algorithm that used for searching and indexing a sub-text in one or more texts. From web page indexing like Google Search Engine to a simple find and replace tool in Notepad, string matching is a very important algorithm for almost every application. In courses' website, often there are thousands of papers which is openly-shared. How does string matching algorithm helps in minimizing the time to search the papers we wanted?**

Keywords—string matching, Knuth-Morris-Pratt, Boyer-Moore, Regular Expression

I. Introduction

We surely have been given assignments to write a paper as a college student at least once. The paper is often about our understanding of the lectures we had. The lecturer may give the students the subject(s) for the paper, or give the students a freedom to pick the subject from the given lectures. The students then submit the paper to the lecturer for one of the scoring criteria of the course he currently takes.

Sometimes, the lecturer gives an extra rule of avoiding the same title as the previous submitted papers. This can easily be done if the old submitted papers are listed in the course website and the list are small. However, a big problem can occur if the course has been running since a long time, and there are thousands of papers listed in a different web pages. It will cost a long time to search the entire list whether our paper title has already been submitted or not.

Also, when we want to read or cite from the previous submitted papers, we surely don't want to search the entire list of papers just to find the paper we wanted. We can waste more time just to search the paper than working on our own paper. This is where string matching algorithm comes very handy in aiding our problem. There are many string matching algorithm that has been made by computer engineers, and every new algorithm, the complexity are getting much better than before.

This paper will explore three string matching algorithm that are commonly used in programming : Knuth-Morris-Pratt, Boyer-Moore, and Regular Expression, implements them in searching IF2211 Algorithm Strategy papers, and analyze the time execution for every string matching algorithm stated above.

II. A Short Explanation of String Macthing Algorithm

String matching algorithm is an algorithm to determine whether a pattern is contained within a current text. From this definition, this algorithm can be expanded into multiple uses—e.g., counting how many occurrences of a pattern in a text, and getting the index of a pattern's first occurrence in a text. As stated in Introduction, there are 3 commonly used string matching algorithm used in programming, each will be explained below.

A. Knuth-Morris-Pratt (KMP)

KMP algorithm was conceived and published by Donald E. Knuth, James H. Morris, and Vaughan R. Pratt in 1977. This algorithm's running time is proportional to the sum of the length of the pattern, making it low enough to make this algorithm practical to use[1]. Although KMP algorithm is a very old algorithm, it is still commonly used in string matching algorithm for its easy to implement and low running time.

The way how KMP algorithm works is to find how many shifts of pattern we can do to minimize the comparison. This can be done by using a KMP Border Function. KMP Border Function pre-process the pattern to find matches of prefixes of the pattern with the pattern itself[2]. This is useful because we can shift our pattern farther, while also minimizing the comparison by ignoring the characters that are definitely same as the text character. For example we have a pattern P of ABAAAB with the border function below.

j	0	1	2	3	4	5
P[j]	A	B	A	A	A	B
k	-	0	1	2	3	4
b(k)	-	0	0	1	1	1

j = position of patterns

P[j] = the character of pattern in position j

k = position before pattern mismatch

b(k) = the size of the largest equal substring

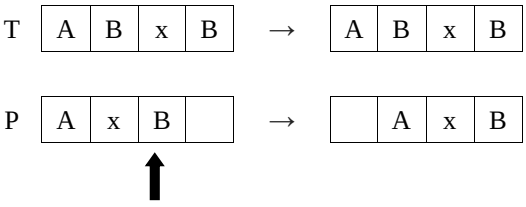
Table 2.1 KMP Border Function

We take b(3) as an example. We take a substring i as a prefix of the pattern before P[3], which is ABAA. Then we take a substring j as a suffix of pattern between P[1..3], which is BAA. From that substrings, we find the size of largest prefix of i that is also a suffix of j, in this example A is the prefix of i and the suffix of j. Now we can calculate the length of string "A", which is 1 and make it as a result of b(3). This calculations are repeated until all of the KMP Border Function are filled.

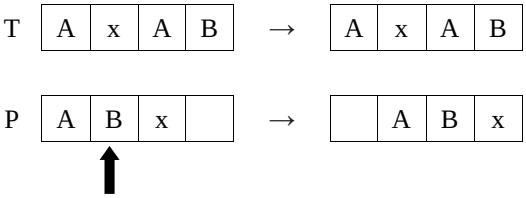
B. Boyer-Moore

Boyer-Moore algorithm was developed by Robert S. Boyer, and J Strother Moore in 1977[3]. Boyer-Moore algorithm, rather than matching the characters of the pattern from the front, it matches the characters backwards starting at its end. Booyer-Moore algorithm can also jump backwards and forwards depending on the pre-process algorithm used, unlike the KMP algorithm. There are 3 jumps that Booyer-Moore can do, based on its mismatch cases[4], which is

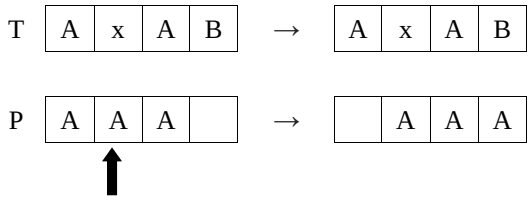
1. if pattern contains x to the left of mismatched character, then shift the pattern to the right to align the last occurrence of x in the pattern with the text



2. if pattern contains x, but a shift right is not possible, then shift the pattern to the right by one character



3. if case one and two don't apply, then shift the pattern to align the first character of the pattern to the location of the mismatch in the text



Unlike KMP, Booyer-Moore excels in long pattern. This is because the longer the character, more jumps can be made, making the string matching faster.

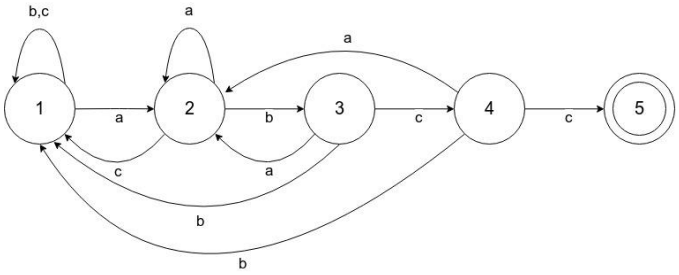
C. Regular Expression (Regex)

Regex is a series of characters that defines a pattern. Regex was described by mathematician Stephen Cole Kleene in 1951[5]. Rather than using a standard, alphabetic and numeric characters for a pattern, Regex uses special characters that generates a pattern based on our preferences using a finite state machine. Based on the Regex input, we can create multiple patterns at once without re-inputing the different pattern manually.

A finite state machine is a machine that accepts a string as and input and outputs an answer whether the string is accepted or rejected. How a string is accepted or rejected is again, based on the Regex pattern we define as a string generator. For an example we will make a Regex pattern of (a*b*cc)

P = (a*b*cc)

This will create a possible pattern of cc, acc, bcc, abcc, aabcc, aabbcc, aaabcc, and so on. The letter a and b can be made indefinitely, while cc are static. From the pattern P, we can get an example pattern of "abcc". This patter now will be created as a finite state machine to determine whether a text contains "abcc" as a sub-text. The finite state machine can be seen below

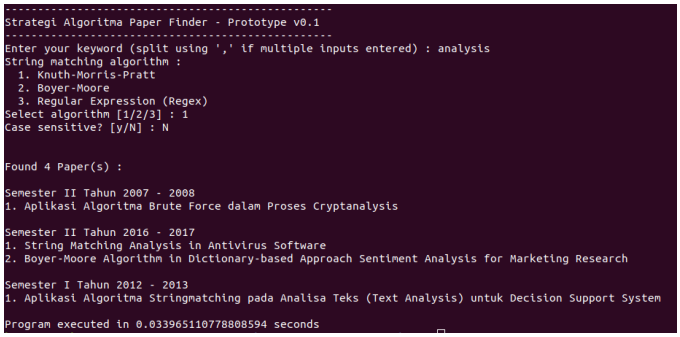


Picture 2.1 Finite State Machine of Pattern "abcc"

For example, we have a text "aaabaccabccaabaacc". This text is accepted in this finite state machine, because sub-text "abcc" are located in index 7 (text index starts at 0). Therefore we can conclude that pattern the text does contains sub-text "abcc"

III. Implementation of String Matching Algorithm

To demonstrate the string matching algorithm in Algorithm Strategy papers finder, I have made a simple program complete with runtime execution for further analysis. The data for the program are scraped from Dr. Rinaldi Munir's website, http://informatika.stei.itb.ac.id/~rinaldi.munir/.



Picture 3.1 The Interface of Sample Program

The program takes a text from the keyboard as the pattern for the string matching algorithm. The user then select which algorithm to be used and decides whether the matching are case sensitive, and in the case of Regex, matches the whole word. The results then printed below, with the title of the papers are grouped by semester and year. The grouping will make it easier to find the paper in Dr. Rinaldi Munir's website.

To test the runtime of all the algorithm, there are two text that will be tested with the string matching algorithm : "greedy" with 6 characters long; and "implementation" with 14 characters long. Ten tests will be taken to minimize the error rate. KMP algorithm will be the first to be demonstrated, followed with Boyer-Moore and Regex.

1. KMP

a) Text = "greedy"

Tests	Results (seconds)
1	0.04796862602233887
2	0.047856807708740234
3	0.05599665641784668
4	0.051872968673706055
5	0.05208086967468262
6	0.05034351348876953
7	0.05175137519836426
8	0.04833483695983887
9	0.04435372352600098
10	0.04629707336425781

Table 3.1 KMP "greedy" Text Result Table

b) Text = "implementation"

Tests	Results (seconds)
1	0.05278611183166504
2	0.053936004638671875
3	0.054134368896484375
4	0.05591249465942383
5	0.05426597595214844
6	0.056267499923706055
7	0.04584527015686035
8	0.052819013595581055
9	0.05540657043457031
10	0.053624868392944336

Table 3.2 KMP "implementation" Text Result Table

2. Boyer-Moore

a) Text = "greedy"

Tests	Results (seconds)
1	0.08899712562561035
2	0.08296036720275879
3	0.08891606330871582
4	0.08765721321105957
5	0.08796572685241699
6	0.08844161033630371
7	0.08422493934631348
8	0.09043073654174805
9	0.08984875679016113
10	0.08875346183776855

Table 3.3 Boyer-Moore "greedy" Text Result Table

b) Text = "implementation"

Tests	Results (seconds)
1	0.07014894485473633
2	0.07564210891723633
3	0.0727989673614502
4	0.07069587707519531
5	0.07214832305908203
6	0.0709528923034668
7	0.06855225563049316
8	0.06955265998840332
9	0.07296919822692871

10	0.06373095512390137

Table 3.4 Boyer-Moore "implementation" Text Result Table

3. Regex
   a) Text = "greedy"

Tests	Results (seconds)
1	0.007326602935791016
2	0.006690025329589844
3	0.006884098052978516
4	0.0071675777435302734
5	0.006811857223510742
6	0.007042407989501953
7	0.0067479610443115234
8	0.00653839111328125
9	0.0071985721588134766
10	0.0068035125732421875

Table 3.5 Regex "greedy" Text Result Table

   b) Text = "implementation"

Tests	Results (seconds)
1	0.009277582168579102
2	0.0063571929931640625
3	0.005544900894165039
4	0.005502939224243164
5	0.005994081497192383
6	0.006643772125244141
7	0.00889134407043457
8	0.0057315826416015625
9	0.00660395622253418
10	0.007650136947631836

Table 3.6 KMP "implementation" Text Result Table

IV. Analysis

After the highest and the lowest result from each individual tests are removed, the average of all the results are calculated and yields the following results,

1. KMP
   a) Text = "greedy"

      average = 0.04956325888633728

   b) Text = "implementation"

      average = 0.05411067605018616

2. Boyer-Moore
   a) Text = "greedy"

      average = 0.0881006121635437

   b) Text = "implementation"

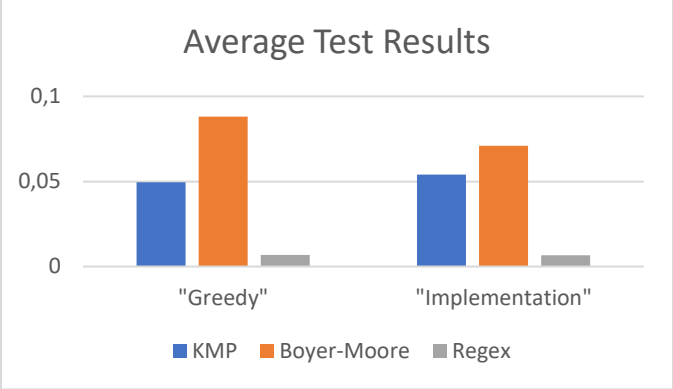
      average = 0.07097738981246948

3. Regex
   a) Text = "greedy"

      average = 0.0069182515144348145

   b) Text = "implementation"

      average = 0.006677120923995972



Picture 3.2 Graph of The Average of Test Results

Based on the calculations, KMP algorithm results are the most average, with both results are close to 50 milliseconds of runtime. However, KMP algorithm runtime are parallel to the pattern length, meaning the longer the pattern length, the longer the time needed to process the text. This makes KMP algorithm excels in short length of pattern, shown in the average result. KMP algorithm also excels in continuous data processing , since KMP algorithm does not need to move backwards to check the text.

Boyer-Moore algorithm on the other hand, has the slowest results. The "greedy" text's runtime is almost 90 milliseconds long. The "implementation" text's runtime however, is only around 70 milliseconds long. That's because Boyer-Moore algorithm runtime is divergent from the pattern characters length, meaning the longer the pattern's length, the shorter the runtime. This makes Boyer-Moore algorithm excels searching a long series of characters, rather than searching short characters or binary[6]

Regex has the best runtime results in both of the text, with both text have an impressive result close to 7 milliseconds. This is because the text are tested instead of the pattern. The pattern makes an automaton that checks the text character by character until it stops in a state. This makes the string matching runs very fast, because we only need to compile a Regex pattern and use

it to check the text, rather than passing the text into an algorithm. This makes Regex a common tool for finding and replacing texts in text editors, since the runtime of the search is very fast.

V. Conclusion

In conclusion, string matching algorithm benefits greatly for students who wants to search the old submitted papers for citing and reading, also to check whether their current paper title is already submitted before or not.

Also, from the 3 string matching algorithm that has been analyzed, KMP algorithm is best for short pattern length and continuous text, Boyer-Moore algorithm is best for long pattern length, and Regex is best for multiple applications that in most of some defeats KMP algorithm and Boyer-Moore algorithm.

Acknowledgment *(Heading 5)*

I would like to thank you for, my family and for giving me emotional and financial support in making this paper, my close friends who also giving me lots emotional support, and *Gojek* drivers for delivering me food take-outs in the busy-times of making this paper

References

[1] G. Eason, B. Noble, and I.N. Sneddon, "SIAM Journal on Computing," vol. 6, No.2, Society for Industrial and Applied Mathematics, pp. 323-350, 1977.

[2] Andrew Davison, "Pattern Matching", Presentation, WiG Lab (teachers room), COE, updated by Dr. Rinaldi Munir, Informatika – STEI, Institut Teknologi Bandung.
[http://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2017-2018/Pencocokan-String-(2018).pdf, accessed on 12 May 2018]

[3] "Communications of the ACM", Magazine, vol. 20, Issue 10, New York: ACM, pp. 762-772, October 1977.

[4] Andrew Davison, op cit., pp. 42-45

[5] S. C. Kleene, "Representation of Events in Nerve Nets and Finite Automata", RM-704, California: U.S. Air Force Project RAND, 15 December 1951.

[6] Andrew Davison, op cit., pp. 55

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 14 Mei 2018

Kevin Fernaldy
13516109