Comparison of String Matching Algorithms For Searching Large Amount of Text

Dariel Valdano - 13512079
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
dariel.valdano@students.itb.ac.id

***Abstract*— There are 3 basic string matching algorithms. The classical, naive brute-force technique it compares every single character, stepping forward when a mismatch is found, The Knuth-Morris-Pratt algorithm, which improvises the naive algorithm to allow smarter shift, and the Boyer-Moore algorithm with its peculiar reverse-searching looking-glass method of searching. This paper will compare all three algorithms by finding some sentence in a large corpus of text, and finally outlines the advantages and disadvantages of each algorithm.**

***Index Terms*— Boyer-Moore, Knuth-Morris-Pratt, String Matching Comparison, String Matching Algorithms**

I. INTRODUCTION

From the early age of computers, the need for searching a subset of string from a larger string becomes more necessary. The requirement of processing string data instead of numerical ones becomes greater and now string matching becomes an essential part of a computer system. Early computers process mostly numerical data for helping in scientific computations, but nowadays string-based data is often calculated.

String matching is particularly useful in Compilers, Data Compressors and unpackers, Database operations, general text-editing work, encryption and more. In compilers, the computer might need to find a referenced function or a variable by searching it over thousands or even millions of lines of code. Using a naive brute force string matching technique will require an astronomical amount of time, and are considered to be inefficient and ineffective. Data compressor and unpackers will also require string matching to search for patterns in data that can be compressed. Database systems such as MySQL will often process millions of records at any given time, searching for a particular record which satisfies a predefined rule. In general text editing work, applications such as Microsoft® Word will analyze typed sentences on the fly, comparing it with a list of grammatical patterns, notifying the user when a seemingly anomalous grammar is detected. The user can then revise the sentence easily.

The need of fast string matching algorithm is absolutely essential to shave precious amount of time in processing time in all the aforementioned applications. Without a fast string matching algorithm, Compiling a Linux kernel would took days instead of hours, Compressing and decompressing data would took 10 times longer than it is now, word processors will be unable to predict and observe typed sentences and provide the relevant correction suggestions, Internet-based search engines such as Google would took minutes processing every query instead of milliseconds.

The world of computing as we know it today will feel much more slower; not because the hardware that is underdeveloped, but because of a computing-power-wasting string matching algorithm that does not utilize the true advancements in string matching algorithm.

There are many algorithms for string matching. The very basic of string matching algorithm is the brute force technique, where a string is compared to a bigger string directly, positioning the substring (the string to be searched) initially in line with the reference string, with its first character location positioned in line with the first character of the reference string. The character is then compared on the first location. If the character is a match, comparison is continued on the second location, until either the entire substring is matched, of a mismatch is found. If a mismatch is found, the entire substring will be shifted one character to the reference string, and then they will be re-compared. The procedure is then repeated until either a match is found or the end of reference string is encountered. This algorithm will be reviewed in more detail in the respective chapter.

The second algorithm is the Knuth–Morris–Pratt Algorithm. It is an improved version of the Brute Force Technique, which skips an obvious mismatch when it occurs. It precomputes a partial match table and then skips n characters when a mismatch is found, according to that table. This algorithm will be reviewed in more detail in the respective chapter.

The third algorithm is the Boyer-Moore algorithm. It uses a looking glass technique, where it finds the string being searched for backwards through the string being searched. Then it uses a more advanced algorithm to decide how much shift is needed if a mismatch is found. This algorithm will be reviewed in more detail in the respective chapter.

These three algorithm will be tested and benchmarked by searching a set of sentences in a corpus. A Text corpus is a big file of text with structured sentences, often in English, that can be used for linguistics research or general research. In this paper the corpus will be used to compare all three algorithms. The text corpus used in this paper will be the scriptment of the movie AVATAR by James Cameron. The time taken to search through the corpus will be measured and compared on all three algorithm. The author will then create a conclusion over which string matching is best used for searching through a text corpus.

II. The Three Algorithms

A. Brute Force Algorithm

Brute Force string matching algorithm searches for a string pattern in a file with brute force, meaning it uses computing power to search for the pattern without using any advanced algorithms that can be used. The Brute Force algorithm tries to find the string to be looked in the string of reference by comparing each character, advancing one character forward if a mismatch is found.

This Algorithm can be very time intensive. For example, if there is a reference text that contains the string "ABCBC" repeated for one thousand times, appended at the last with the characted "D", and the string to be searched for is "ABCABCABCD", the brute force algorithm will search every single reference text 1000 x 10, which results in 10000 character comparisons. Scale it up to a million, and the computing power needed just to find a string would be astronomical.

Due to those reason, it is clear that the brute force will be the worst performing algorithm between the three, but it will still be benchmarked for reference point.

B. Knuth-Morris-Pratt Algorithm

The Knuth-Morris-Pratt Algorithm "readies" the string to be searched for by precomputing a table using a function called "Border Function". This will create a table with columns the same size with the string length. The KMP Border Function is defined as "the size of the largest prefix of P[1..k] that is also a suffix of P[1..k][2]. P is the string to be searched in the longer text. When a mismatch is found, the entire P string is shifted as much as the table specifies. For example, if a mismatch is found at P[5], then the entire string is shifted as much as BorderFunction(P[5]). This avoids wasteful comparisons, and can improve string the string matching time.

Knuth-Morris-Pratt algorithm the advantage to the fact that it will never need to move backward in the text to be searched, and thus is very useful to process large files or stream of file[2].

The Disadvantage of Knuth-Morris-Pratt is that it works well if the alphabet size is small. When the number of alphabet becomes very big, it will have more chances of mismatch, and due to the large amount of mismatch, the will also tend to occur early in the pattern, thus reducing the speed of the Knuth-Morris-Pratt algorithm to nearly as slow as Brute Force.

Using the previous example in the brute force algorithm, we can see that it can shave most comparisons off by shifting it more than one character at a time.

The algorithm was invented in 1974 by Donald Knuth, Vaughan Pratt and James H. Morris.

B. Boyer-Moore Algorithm

Boyer-Moore Algorithm also pre-computes the string to be searched in advance, called the Last Occurrence Function. This function will preprocess P to find the last occurrence at position x, where L(x) is defined as the largest index I such that P[i]==x or -1 if no such index

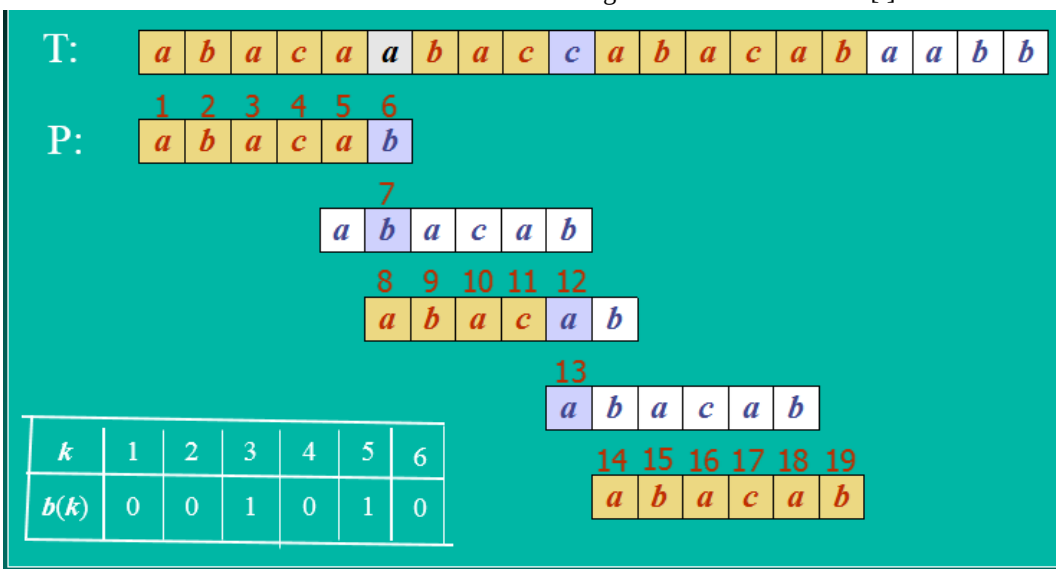


Figure II.1 – example of the Knuth-Morris-Pratt Algorithm [2]

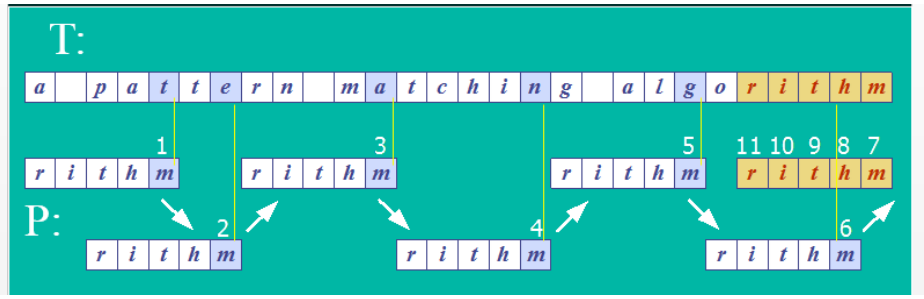


Figure II.2 – example of the Boyer-Moore Algorithm [2]

exists[2].

Imagine a string to be searched called T and string to search P. When a mismatch occurs in location T[i] and P[j], there will be three possible cases tried at once:

First, the algorithm will check whether P contains x (where x = T[i]). If P contains x, then shift P to the right to align to the last occurrence of x in P with T[i].

Second, if the first condition is not satisfied, because a shift right to the last occurrence is not possible, shift P right by one character to T[i+1].

Lastly, if neither of the first or second condition is not satisfied, then shift P to align P[1] with T[i+1] [2].

This Algorithm is good when Knuth-Morris-Pratt is not; it easily handle strings with large alphabet, yet slow if the alphabet is small.

III. The Benchmark

The comparison will be performed on a text corpus, roughly 200000 characters long. The corpus used is the scriptment from the movie AVATAR written by James Cameron. Each algorithm will be performed on a search multiple times, the average taken. Then all three algorithms will be compared side by side in a graph, and the algorithm that requires less processing time will be considered as the best algorithm for that test.

> A GAS-GIANT PLANET called POLYPHEMUS, ringed with dozens of moons which cast beauty-mark shadows on its vast face. The ISV diminishes away from us toward the largest MOON-- a blue and surprisingly Earth-like world called PANDORA. The ship dwindles to a speck against the BLUE MOON. CUT TO: EXT. PANDORA ORBIT ISV Venture Star drifts above a spectacular vista -- the sapphire seas and unfamiliar continents of Pandora. CLOSE ON ISV -- two massive “VALKYRIE” SHUTTLES are mated to a DOCKING NODE. One of them separates from the starship and moves away, its thrusters FIRING in short bursts. As the shuttle moves away, descending toward Pandora, we hear the sound of DRUMS, building, louder and louder until we-- CUT

Figure III-1 an Except from the corpus

The source code were made by the author folowing the specifications of each algorithm, referencing to reference [2] when possible. The program is specifically made to test and benchmark the three algorithms. There will be three tests:

One, will be a search on the corpus using 5 word as the text to be searched.

Two, will be a search on the corpus using 2 entire sentence as the text to be searched.

Three, will be a search on the corpus using an entire paragraph as the text to be searched.

IV. Test Results

	Brute Force	KMP	BM
Test 1	25028546ns	19915864ns	Error
Test 2	2283576ns	39538085ns	Error
Test 3	2042389ns	74467964ns	Error

Figure IV-1 – The test results

Due to programming errors in BM source code algorithm, and the constraint of time, the author cannot fix the error in the allocated time. Due to this, the Boyer-Moore algorithm cannot be compared to the remaining two. There seems to be a program quirk and or compiler optimization that actually makes the brute force method of string matching faster than the KMP equivalent. Perhaps if further investigation is performed, the nature of this strange enhancement can be traced and turned off for this comparison. But due to the constrainst of time, this could not be performed

V. Conclusion

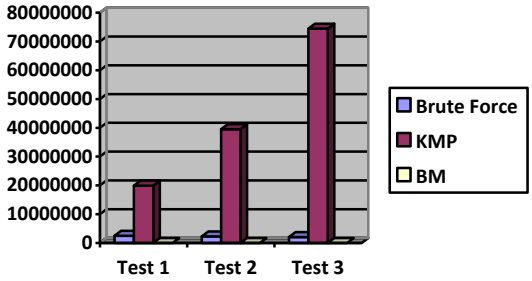
From the test results, the following conclusion is made:

1. KMP is the fastest on test 1, but in the remaining test, the Brute force is faster, due to an unknown optimization performed by the compiler.
2. More time is needed to find out the reason to the brute force optimization, and to fix the Boyer-Moore source code error.

VI. APPENDIX

	Brute Force	KMP	BM
Test 1	25028546ns	19915864ns	Error
Test 2	2283576ns	39538085ns	Error
Test 3	2042389ns	74467964ns	Error

Appendix 1 – The test results



Appendix 2 – Visualization of the test results

VII. ACKNOWLEDGMENT

First, the author would like to express his highest praise to God Almighty for the grace and enlightenment that allows me to complete this paper. Author would also like to give his biggest thanks to his parents, who's without their guidance, the author might not be able to have a chance in studying this high. Author would also like to express his thanks to Mr. Rinaldi Munir, for his teachings allows the author to understand the concepts behind all three string matching algorithm which this paper is based on.

REFERENCES

[1] Cameron, James. AVATAR Movie Scriptment © 2007 TWENTIETH CENTURY FOX.
[2] Dr. Davison, Andrew. Pattern Matching Slide, WiG Lab, CoE

STATEMENT

I hereby declare that I wrote this paper with my own writing, not adaptation, or translation of someone else's paper, and not plagiarism. Any citations will be properly attributed to their original authors in the References chapter.

Bandung, 18 May 2014

Dariel Valdano - 13512079