ISSN 2355-3286

Perbandingan Algoritma Horspool dan Algoritma Zhu-Takaoka dalam Pencarian *String* Berbasis Desktop

Adhi Kusnadi[1], Abraham Khrisnandi Wicaksono[2]
Fakultas Teknologi Informasi dan Komunikasi, Universitas Multimedia Nusantara, Gading Serpong, Indonesia
adhi.kusnadi@umn.ac.id[1], lincoln.abra92@gmail.com[2]

Diterima 14 Maret 2017
Disetujui 12 Juni 2017

Abstract*— String searching is the search process by using the index to find text that can help in information retrieval systems. Continuing previous research, this study uses an algorithm Horspool and Zhu-Takaoka to find the performance of each of these algorithms in the search for a pattern in the text. So they make a useful desktop-based application to measure performance of both algorithm, particulary the time required to perform string searching process. Using the method of prototyping and Microsoft Visual Studio with C# programming language implementation. Result obtained from this application is the number of words found, and the processing time of each algorithm. From this study, the Horspool algorithm is 19,82845 percent faster in first test with the pattern "swan" in a text file 50 multiples 1000 words and 15.9442 percent in the second trial using text files 70000 words with different pattern than the number of characters Zhu algorithm -Takaoka in the process of searching string.

***Index Terms*-String searching, Horspool, Zhu-Takaoka, Microsoft Visual Studio, application, processing time.**

I. PENDAHULUAN

Pada zaman sekarang ini penggunaan komputer sudah merakyat dan hampir selalu digunakan untuk menjalankan berbagai aktivitas manusia. Diantara banyak fungsi komputer yang digunakan oleh manusia adalah pencarian data serta pengurutan data [1]. Perkembangan teknologi informasi dan komunikasi berdampak luas dalam kehidupan manusia. Semenjak dikembangkannya komputer pada pertengahan abad ke-20, peradaban manusia memulai babak yang baru. Teknologi Informasi dan Komunikasi membawa manusia pada era baru, era dimana percepatan perubahan kebudayaan sangatlah dramatis. Suatu era dimana informasi menjadi hal yang sangat penting bagi kehidupan manusia. Era ini kemudian dinamakan Era Informasi [2]. Oleh karena itu teknologi informasi sangat penting dan berguna di setiap aspek kehidupan bagi manusia.

String searching adalah komponen yang penting dari berbagai macam masalah, seperti *text editing*, data *retrieval* dan *symbol manipulation*. Walaupun penggunaan indeks digunakan untuk mencari teks, *string searching* dapat membantu dalam sistem pencarian informasi. Sebagai contoh, digunakan untuk mencari persamaan yang potensial dalam pencarian teks atau untuk mencari istilah yang akan ditampilkan melalui perangkat output [3]. *String searching* merupakan sarana untuk mengembangkan sistem pencarian informasi.

Permasalahan pencocokan *string* (*string matching*) merupakan permasalahan yang sangat terkenal dalam dunia informatika. Contoh implementasi dari permasalahan pencocokan *string* adalah pada pencocokan sebuah *string* pada Microsoft Word, *string editor*, atau dalam kasus yang lebih besar lagi, yaitu pencocokan *website* dengan memasukkan kata-kata kunci sebagaimana yang telah diimplementasikan pada *search engine*, seperti Yahoo atau Google [4]. Pencocokan *string* (*string matching*) telah menjadi kebutuhan yang digunakan untuk mencari informasi melalui berbagai aplikasi.

Pada penelitian sebelumnya yang dilakukan oleh Vina Sagita disimpulkan bahwa algoritma Boyer-Moore adalah algoritma yang paling cepat dalam pencarian *string* [5] dan memberikan saran untuk dapat dilakukan penelitian kembali untuk mencari kecepatan dari tiap-tiap algoritma pencarian *string* dan menemukan algoritma yang dapat melakukan pencarian *string* paling cepat.

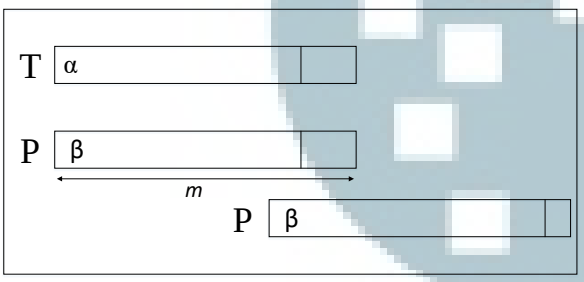
Dengan mengetahui pentingnya pencarian *string* di dalam pencarian informasi, penelitian ini bertujuan untuk mengetahui algoritma manakah yang paling cepat dalam mencari *string* yang berguna untuk mengoptimalkan sistem pencarian informasi melalui sarana berupa teks secara lebih cepat dan akurat. Dalam penelitian ini, dilakukan percobaan untuk membandingkan kecepatan dari algoritma Horspool dan Zhu-Takaoka dalam kemampuan pencarian *string* yang diimplementasikan ke dalam sebuah aplikasi berbasis desktop.

II. ALGORITMA DASAR

A. *Algoritma Boyer-Moore*

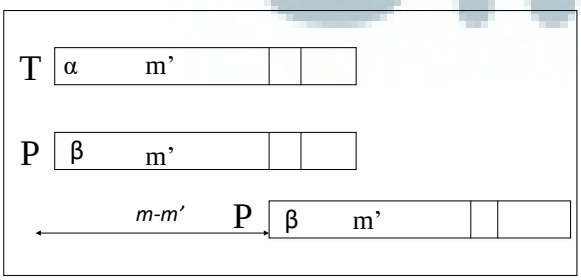
Pada tahun 1977, R.S Boyer dan J.S Moore merancang algoritma pencarian *string* linear. Proses komparasi pattern dengan *text* yang dicari dilakukan dari kanan ke kiri. Algoritma Boyer-Moore sangat populer dan paling sering digunakan dalam algoritma pencocokan *string*. Algoritma Boyer-Moore membangun dua tabel *preprocessing* yaitu *Boyer-Moore bad characters* (BMbc) yang sering disebut pergeseran ketidakcocokan dan *Boyer-Moore good suffix* (BMgs) yang disebut pergeseran kecocokan. Pergeseran pattern dilakukan berdasarkan nilai dari tabel BMbc dan BMgs [6].α

Pergeseran BMbc memiliki dua kasus pergeseran. Kasus pertama adalah jika ketidakcocokan ada pada karakter pertama dari pattern P (posisi karakter ke mth) dengan karakter dari *text* T yang dicocokan. Selain itu ketidakcocokan karakter dari *text* T juga tidak muncul di pattern P sehingga pergeseran pattern sebanyak panjang pattern m yang ditunjukkan pada Gambar 1 [6].



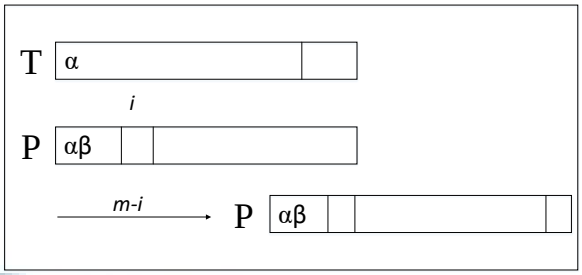
Gambar 1 Pergeseran BMbc Kasus Pertama

Sedangkan untuk kasus kedua jika karakter terakhir m' dari pattern P sesuai dengan karakter m' yang berkorespondensi *text* T kemudian terjadi ketidakcocokan setelah karakter m' kemudian pattern digeser sebanyak (m – m') yang ditunjukkan pada Gambar 2 [6].



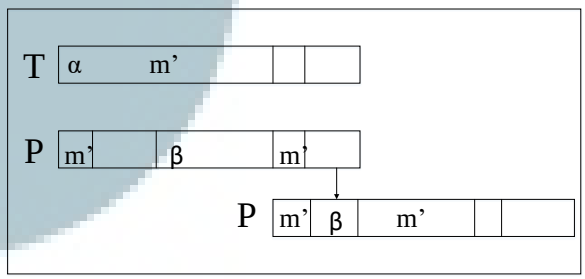
Gambar 2 Pergeseran BMbc Kasus Kedua

Untuk pergeseran BMgs memiliki dua kasus dalam melakukan pergeseran. Kasus pertama adalah jika ketidakcocokan ada di karakter pertama dari pattern P (posisi ke mth) dengan karakter yang berkorespondensi dengan *text* T tetapi ketidakcocokan karakter dari *text* T juga muncul di pattern P pada posisi ke ith kemudian pergeseran pattern dilakukan sebanyak (m-i)th. Di mana, i adalah posisi dari karakter yang cocok di pattern P yang ditunjukkan pada Gambar 3 [6].



Gambar 3 Pergeseran BMgs Kasus Pertama

Sedangkan untuk kasus kedua dalam pergeseran BMgs adalah jika karakter terakhir m' dari pattern P tidak cocok dengan karakter ke m' dari *text* T tetapi karakter ke m' terulang kembali di pattern P maka pergeseran pattern dilakukan dengan mensejajarkan karakter yang sesuai dengan *text* dan pattern yang ditunjukkan pada Gambar 4 [6].



Gambar 4 Pergeseran BMgs Kasus Kedua

B. *Algoritma Horspool*

Algoritma Horspool adalah modifikasi dari algoritma Boyer-Moore dengan sedikit perubahan. Tidak seperti algoritma Boyer-Moore, algoritma Horspool hanya menggunakan satu tabel (*bad character shift*) dimana algoritma Boyer-Moore menggunakan dua tabel: *bad character shift* dan *good suffix shift* [7].

Algoritma Horspool mencari pattern dari kiri ke kanan dan untuk *shift value* berdasarkan ukuran dari pattern yang dicari dalam *bad character shift* tabel. Algoritma Horspool tidak efisien untuk kata yang pendek. Tetapi, jika *text* yang panjang dibandingkan dengan pattern yang dicari seperti yang terjadi dengan tabel ASCII dan pencarian kata yang ada di *text* editor, algoritma Horspool menjadi sangat berguna. Algoritma Horspool hanya menggunakan pergeseran bad-character pada bagian kanan karakter [8].

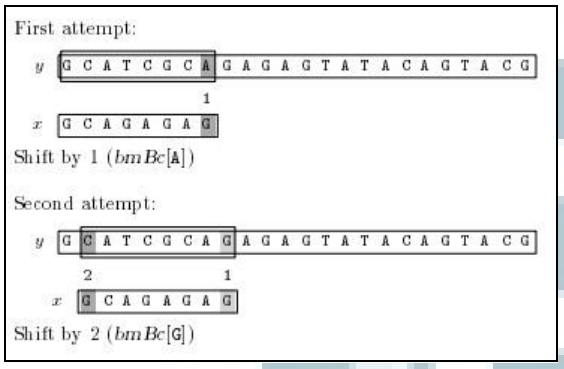
Sebagai contoh terdapat *text* dengan karakter GCATCGCAGAGAGTATACAGTACG kemudian

dengan pattern yaitu GCAGAGAG. Pada algoritma horspool dilakukan subproses dengan menghitung tabel *bad character shift* dari algoritma Boyer-Moore.

a	A	C	G	T
$bmBc[a]$	1	6	2	8

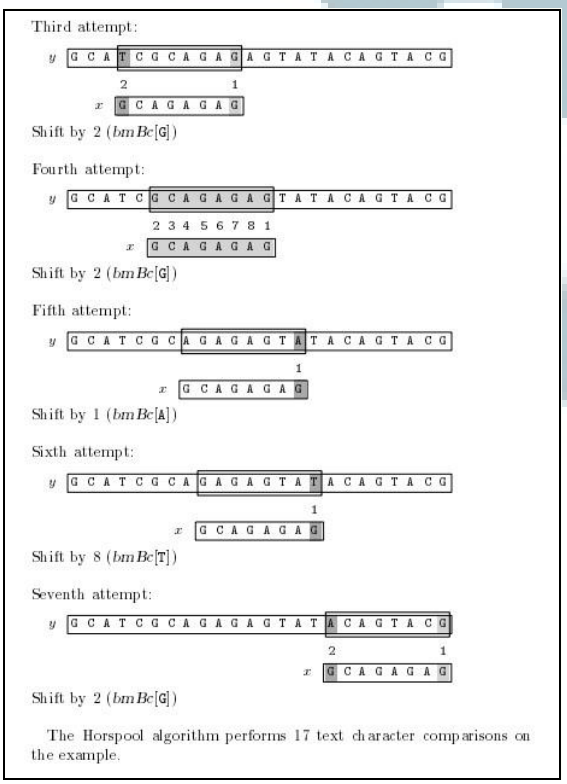
Gambar 5. *Bad character shift* Algoritma Horspool[8]

Langkah-langkah yang dilakukan seperti pada gambar berikut :



Gambar 6 *Searching Phase* Algoritma Horspool [8]

Kemudian dilanjutkan seperti pada gambar dibawah berikut :



Gambar 7. *Searching Phase* Algoritma Horspool [8]

Pada Gambar 6 dan 7 menunjukkan langkah-langkah pencocokan *string* dari algoritma Horspool pergeseran dilakukan menggunakan *bad character shift* dari karakter paling kanan dari *text* y yang dicocokan dengan pattern x.

C. Algoritma Zhu-Takaoka

Algoritma Zhu-Takaoka diciptakan oleh R.F.Zhu dan T.Takaoka pada tahun 1989. Algoritma ini menggunakan dua *text* karakter untuk menghitung pergeseran karakter berdasarkan *bad-character*. Untuk mencari ketidakcocokan atau mencari seluruh *text* menggunakan *preprocessing hashing*. Hal ini efektif untuk pencarian *string* dua dimensi [9].

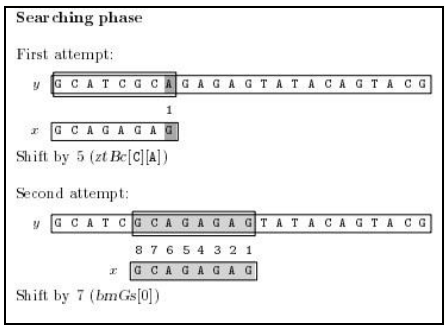
Algoritma Zhu-Takaoka menjelaskan bahwa pencarian pattern dilakukan mulai dari tiap baris mulai dari baris ke 0 dan berakhir pada baris ke n1 – m1. Zhu dan Takaoka merancang sebuah algoritma yang dapat bekerja menggunakan pergeseran dengan *bad-character* untuk dua karakter *text* secara berturut-turut. Selama fase pencarian pencocokan karakter dilakukan dari kanan ke kiri dan ketika proses berada di posisi y[j..j + m-1] dan ketidakcocokan terjadi diantara x[m-k] dan y[j+m-k] ketika x [m – k+1..m-1] = y[j+m-k+1..j+m-1] pergeseran dilakukan dengan pergeseran *bad-character* untuk *text* karakter y[j+m-2] dan y[j+m-1]. Tabel good-suffix dari algoritma Boyer-Moore juga digunakan untuk menghitung pergeseran [8].

Pada awal proses algoritma Zhu-Takaoka melakukan *preprocessing* untuk menghitung *bad character shift* dan *good suffix* dari algoritma Boyer-Moore.

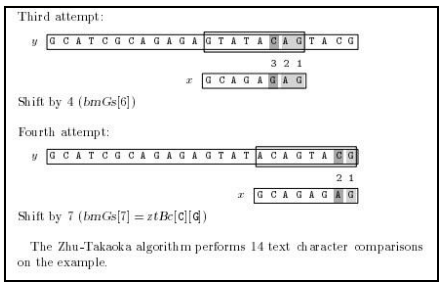
$ztBc$	A	C	G	T
A	8	8	2	8
C	5	8	7	8
G	1	6	7	8
T	8	8	7	8

i	0	1	2	3	4	5	6	7
$x[i]$	G	C	A	G	A	G	A	G
$bmGs[i]$	7	7	7	2	7	4	7	1

Gambar 8. *Bad character shift* dan *Good suffix* Shift Zhu-Takaoka



Gambar 9. *Searching Phase* Algoritma Zhu-Takaoka



Gambar 10. *Searching Phase* Algoritma Zhu-Takaoka

Pada Gambar 9 dan Gambar 10 merupakan langkah-langkah pencarian *string* dari algoritma Zhu-Takaoka jika terjadi ketidakcocokan maka akan dilakukan pergeseran berdasarkan nilai terbesar antara bmGs[i] dengan ztBc[y[j+m-2]][y[j+m-1]]. Sedangkan jika ditemukan kecocokan maka pergeseran dilakukan dengan nilai i dari bmGs pada posisi 0 atau nilai i=0 [8].

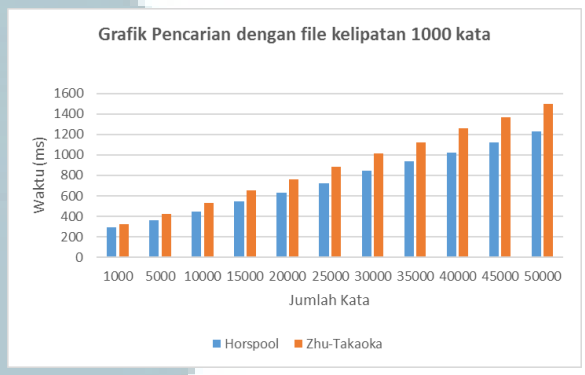
III. PENELITIAN DAN HASIL UJI COBA

Aplikasi yang telah dibuat mulai dijalankan dengan berbagai *text file* untuk membandingkan kecepatan dari tiap algoritma. Data-data yang digunakan dalam proses pengujian ini adalah *text file* dengan extensi (*.txt). *Text file* tersebut berisi kutipan kalimat dari novel fiksi *Twilight* versi bahasa inggris karangan dari Stephenie Meyer. Proses pengujian disediakan 50 file yang terdapat kalimat novel *Twilight*. Panjang kalimat tiap *text* file tersebut adalah kelipatan dari 1000 kata yang dimulai dari file yang terdiri 1000 kata hingga 50000 kata. Pattern yang dicari adalah kata ”Swan” (gambar 11). Selain itu disediakan data yang terdiri dari 70000 kata untuk melakukan ujicoba dengan pattern yang lebih panjang dengan kata ”Isabella Swan” (gambar 12).

Setelah data telah disiapkan, proses pengujian dilanjutkan dengan membuka aplikasi. Setelah aplikasi terbuka, *priority* aplikasi ini diubah menjadi *high* melalui Windows Task Manager. Tujuan mengubah *priority* yaitu untuk menyediakan *resource* yang lebih besar untuk aplikasi sehingga dapat melakukan proses pencarian secara cepat dan stabil.

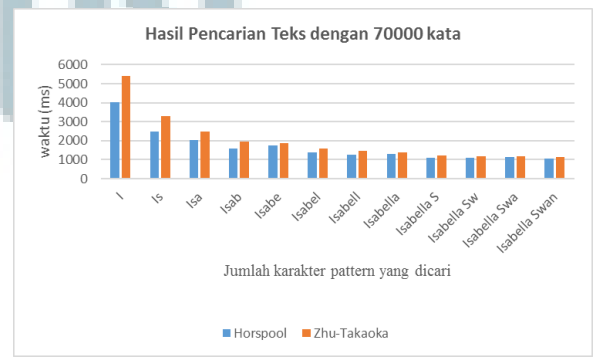
Proses pengujian dilanjutkan dengan membuka *file text* dengan submenu *Open* yang ada di dalam menu File pada aplikasi. Setelah file dibuka dan pattern berupa kata ”Swan”, proses pencarian dilakukan dengan klik button “Cari Kata”. Setiap kilk button tersebut proses pencarian dari tiap algoritma dilakukan sebanyak 500 kali dengan tujuan untuk memperoleh hasil pencarian yang stabil. Pada gambar 11 dapat dilihat grafik pencarian dengan file kelipatan 1000 kata dengan pattern ”swan”. Setiap kelipatan kata dengan pattern “swan” dengan masing-masing algoritma diuji kecepatannya dalam mencari, kemudian hasilnya dirata-rata, didapat bahwa algoritma Horspool lebih cepat 19.82845 % pada uji coba pertama menggunakan 50 file *text* file kelipatan 1000 kata dengan pattern yang sama. Kemudian dilakukan percobaan dengan pattern yang berbeda beda (gambar 12). Dengan text file 7000 kata dan 12 pattern dihasilkan rata-rata kecepatan 15.9442 %.

Setelah proses pengujian selesai, proses validasi dilakukan untuk mengetahui apakah algoritma Horspool maupun algoritma Zhu-Takaoka valid dalam melakukan proses pencarian *string*. Proses validasi yang dilakukan adalah membandingkan jumlah kata yang ditemukan oleh aplikasi ini dapat menggunakan aplikasi lain yang sejenis atau yang berfungsi sama atau juga dapat menggunakan fitur *find* yang ada di aplikasi microsoft word. Dengan hasil yang sama antara aplikasi perbandingan dan microsoft word.



Gambar 11. Grafik Pencarian Dengan File Kelipatan 1000 Kata Dengan Pattern”Swan”

Perbandingan waktu pencarian algoritma Horspool dan algoritma Zhu-Takaoka yang telah di uji coba. Grafik tersebut merupakan perbandingan waktu proses dengan jumlah kata yang ada dalam *text* file.



Gambar 12. Grafik Pencarian Dengan File 70000 Kata Dengan Berbagai Jumlah Pattern

Dari gambar tersebut menunjukkan grafik perbandingan antara waktu dengan jumlah pattern yang dicari. Dalam grafik tersebut algoritma Horspool lebih cepat daripada algoritma Zhu-Takaoka dalam

melakukan proses pencarian *string*. Perlu ada penelitian lanjutan mengapa algoritma Horspool lebih cepat dari algoritma Zhu-Takaoka dan menganalisa hasil percobaan yang telah dilakukan.

IV. SIMPULAN

Berdasarkan tujuan penelitian untuk mengetahui perbandingan performa antara algoritma Horspool dan algoritma Zhu-Takaoka, dibuatlah aplikasi pembanding algoritma Horspool dan algoritma Zhu-Takaoka yang dapat mengukur performa antara kedua algoritma tersebut. Performa yang diukur adalah waktu dari algoritma Horspool dan algoritma Zhu-Takaoka dalam melakukan pencarian pattern di dalam suatu *text* file.

Dari hasil pengujian aplikasi, bahwa algoritma Horspool lebih cepat 19.82845 persen pada uji coba pertama menggunakan 50 file *text* file kelipatan 1000 kata dengan pattern yang sama dan 15.9442 persen pada ujicoba kedua dengan file *text* 7000 kata. Sehingga dapat disimpulkan algoritma Horspool lebih cepat daripada algoritma Zhu-Takaoka.

DAFTAR PUSTAKA

[1] Handoyo, Ria Arini. 2004. "Perbandingan Waktu Proses Pencarian Data Antara Algoritma Raita dengan Algoritma Zhu-Takaoka" Dalam http://karyailmiah.tarumanagara.ac.id/index.php/S1TI/article/view/2315 Diakses tanggal 29 Agustus 2014

[2] Munggaran, Abdi Halim. 2009. "Perilaku Pencarian Informasi Mahasiswa yang Memanfaatkan Layanan Search Engines dalam Menyusun Skripsi: Studi Kasus Mahasiswa S1 Program Studi Ilmu Perpustakaan Universitas Indonesia" Dalam http://lib.ui.ac.id/file?file=digital/123786-RB13A44p-Perilaku%20pencarian-Pendahuluan.pdf Diakses tanggal 26 Agustus 2014.

[3] Baeza-Yates, Ricardo A. Tanpa Tahun. "*String* Searching Algorithms" Dalam http://orion.lcg.ufrj.br/Dr.Dobbs/books/book5/chap10.htm Diakses tanggal 25 Agustus 2014

[4] Hartoyo, Eko Gunocipto, Yus Gias Vembrina, dkk. 2010. "Analisis Algoritma Pencarian *String* (*String* Matching)" Dalam http://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/Makalah/MakalahStmik10.pdf Diakses tanggal 25 November 2014

[5] Sagita, Vina. 2012. "Studi Perbandingan Implementasi Algoritma Boyer-Moore, Turbo Boyer-Moore, dan Tuned Boyer-Moore dalam Pencarian *String*" skripsi UMN.

[6] Bhandari, Jamnua dan Anil Kumar. 2014. "*String* Matching Rules Used By Variants of Boyer-Moore Algorithm" Dalam http://www.rroij.com/open-access/*string*-matching-rules-used-by-variants-of-boyermoore-algorithm-8-11.pdf?aid=37835 Diakses tanggal 17 Agustus 2015

[7] Lovis, C dan R.H.Baud. 2000. "Fast Exact *String* Pattern-matching Algorithms Adapted to the Characteristics of the Medical Language"

[8] Charras, Christian dan Thierry Lecroq. 2004. " Handbook of Exact *String*-Matching Algorithms" Dalam http://www-igm.univ-mlv.fr/~lecroq/*string*.pdf Diakses tanggal : 8 September 2014

[9] Prasad J.C1 & K.S.M.Panicker. 2010. "*String* Searching Algorithm Implementation-Performance Study with Two Cluster Configuration" Dalam http://www.csjournals.com/IJCSC/PDF1-2/56..pdf Diakses tanggal 8 September 2014