ISSN 2085-4552

Studi Perbandingan Implementasi Algoritma Boyer-Moore, Turbo Boyer-Moore, dan Tuned Boyer-Moore dalam Pencarian String

Vina Sagita, Maria Irmina Prasetiyowati
Program Studi Teknik Informatika, Universitas Multimedia Nusantara, Tangerang, Indonesia
vinasagita@ymail.com

Diterima 31 Agustus 2012
Disetujui 31 Januari 2013

Abstrak—*String searching* merupakan suatu proses yang umum dilakukan dalam proses-proses yang dilakukan komputer karena teks merupakan bentuk utama penyimpanan data. Terdapat beberapa macam cara yang dapat dilakukan untuk mencari sebuah *string* pada kumpulan *string* lain yang lebih besar. Beberapa diantaranya adalah algoritma Boyer-Moore, Turbo Boyer-Moore dan Tuned Boyer-Moore. Guna mengetahui bagaimana performa algoritma-algoritma tersebut, terutama di bidang waktu yang diperlukan, maka dibuatlah aplikasi yang dapat digunakan untuk mengetahui waktu yang diperlukan untuk mencari suatu *pattern* dalam *text*. Aplikasi dibangun menggunakan metode *prototyping* dan menggunakan Microsoft Visual Studio dengan bahasa C# untuk pembangunannya. Aplikasi ini mendukung pencarian dengan penggunaan tiga algoritma (Boyer-Moore, Turbo Boyer-Moore, Tuned Boyer-Moore), pengubah kata (*replace*), *highlight* kata yang dicari, dan pemberian informasi waktu yang dibutuhkan masing-masing algoritma untuk pencarian serta algoritma mana yang membutuhkan waktu paling sedikit untuk pencarian. Dari penelitian yang dilakukan, dapat disimpulkan bahwa algoritma Boyer-Moore adalah algoritma yang paling cepat dalam pencarian *string*.

Kata kunci—Boyer-Moore, desktop application, kecepatan algoritma, string searching, Tuned Boyer-Moore, Turbo Boyer-Moore

I. Pendahuluan

Tidak dapat dipungkiri lagi teknologi telah berkembang sangat cepat pada zaman sekarang. Hampir semua manusia modern memanfaatkan teknologi untuk mempermudah pekerjaan dan hidupnya. Setiap teknologi memiliki jenis penyimpanan yang berbeda-beda untuk setiap datanya, tapi teks tetap merupakan bentuk utama penyimpanan data (Purwoko, 2006).

Seringkali operasi pada teks melibatkan proses pencarian terhadap kemunculan suatu string dan lokasinya pada teks yang bersangkutan (Sulistyo, 2006). Karena itu lah pencarian string merupakan salah satu hal yang sangat dibutuhkan.

Permasalahan *string searching* adalah untuk mencari beberapa karakter (*pattern*) dalam sejumlah besar teks (Hartoyo, Vembrina, & Meilana, 2011). Dengan demikian dapat disimpulkan bahwa pencarian *string* atau *string searching* adalah pencarian sebuah *pattern* dari teks.

Contoh implementasi dari pencarian *string* adalah pencarian *string* dalam sebuah aplikasi *text editor* seperti contohnya Microsoft Word dan Notepad. Contoh yang lebih besar lagi adalah pencarian dengan suatu kata kunci pada internet seperti yang dilakukan oleh Google, Yahoo, dan Bing (Hartoyo, 2011).

Algoritma yang digunakan untuk pencarian *string* semakin berkembang dari hari ke hari. Tujuan utamanya tentu saja untuk mencari *string* seakurat dan secepat mungkin. Hingga saat ini algoritma pencarian *string* terbagi atas 3 kategori berdasarkan arah pencocokan *string* yaitu dari kiri ke kanan, kanan ke kiri, dan dari arah yang ditentukan secara spesifik. Metode pencocokan dari kiri ke kanan merupakan metode yang paling natural karena sesuai dengan arah membaca, pencocokan *string* dari kanan ke kiri merupakan metode yang dianggap paling efisien dalam praktiknya, dan pencocokan *string* dari arah yang ditentukan secara spesifik merupakan algoritma yang memiliki hasil yang paling baik secara teoritis (Kumara, 2009).

Algoritma yang dianggap memiliki hasil yang paling baik dalam praktiknya merupakan algoritma yang bergerak mencocokan *string* dari arah kanan ke kiri. Algoritma Boyer-Moore adalah salah satu contoh algoritma yang menggunakan arah dari kanan ke kiri. Algoritma ini telah banyak dikenal oleh masyarakat dan dianggap paling efisien untuk pencarian *string*. Ide dibalik algoritma ini adalah bahwa dengan memulai pencocokkan karakter dari kanan, dan bukan dari kiri, maka akan lebih banyak informasi yang didapat (Fauzy, 2011). Algoritma ini kemudian dikembangkan

sehingga tercipta algoritma Turbo Boyer-Moore dan Tuned Boyer-Moore.

Untuk mengetahui manakah algoritma yang mampu mencari *string* paling cepat, maka muncullah ide untuk meneliti kemampuan dari algoritma-algoritma ini. Oleh karena itu, ditawarkanlah sebuah aplikasi yang mengimplementasi kedua algoritma tersebut untuk mencari *pattern* dari *string* yang diberikan.

II. Tinjauan Pustaka

A. String Searching

Permasalahan pencarian *string* (*string searching*) pada dasarnya adalah mencari sebuah *string* yang terdiri dari beberapa karakter (yang biasa disebut *pattern*) dalam sejumlah besar *text*. Pencarian *string* juga biasa digunakan untuk mencari pola bit dalam sejumlah besar *file binary*. Contoh permasalahannya adalah dalam program *text editor* seperti Microsoft Word, atau dalam lingkup yang lebih besar adalah pencarian web seperti Google, Bing, dan Yahoo. (Hartoyo, Vembrina, & Meilana, 2010).

Proses pencocokan *string* yang merupakan bagian dalam proses pencarian *string* memegang peranan penting untuk mendapatkan dokumen yang sesuai dengan kebutuhan informasi. Pencocokan *string* secara garis besar dapat dibedakan menjadi dua yaitu pencocokan *string* secara eksak/sama persis (*exact string matching*) dan pencocokan *string* berdasarkan kemiripan (*inexact string matching/fuzzy string matching*). Pencocokan *string* berdasarkan kemiripan masih dapat dibedakan menjadi dua yaitu berdasarkan kemiripan penulisan (*approximate string matching*) dan berdasarkan kemiripan ucapan (*phonetic string matching*). Contoh *phonetic string matching* adalah kata *step* akan menunjukkan kecocokan dengan kata *step, sttep, stepp, sstep, stepe, steb*. Sedangkan bila kita menggunakan *exact string matching* kata *step* hanya akan menunjukkan kecocokan dengan kata *step* saja (Syaroni, 2005)..

Algoritma pencarian *string* atau sering disebut juga pencocokan *string* adalah algoritma untuk melakukan pencarian semua kemunculan *string* pendek *pattern*[0..n − 1] yang disebut *pattern* di *string* yang lebih panjang teks[0..m − 1] yang disebut teks (Wiramuda, 2008).

B. Algoritma

Menurut Silvina Irwanti pada jurnalnya yang berjudul Implementasi Algoritma *Backtracking* dalam Perancangan Perangkat Lunak *Game* Anagram, kata algoritma diambil dari nama ilmuwan muslim dari Uzbekistan Abu Ja'far Muhammad bin Musa Al-Khuwārizmi (780-846M), sebagaimana tercantum pada terjemahan karyanya dalam bahasa latin dari abad ke-12 "Algorithmi de numero Indorum". Awalnya kata algoritma adalah istilah yang merujuk kepada aturan-aturan aritmatis untuk menyelesaikan persoalan dengan menggunakan bilangan numerik arab. Pada abad ke-18, istilah ini berkembang menjadi algoritma, yang mencakup semua prosedur atau urutan langkah yang jelas dan diperlukan untuk menyelesaikan suatu permasalahan. Pemecahan sebuah masalah pada hakekatnya adalah menemukan langkah-langkah tertentu yang jika dijalankan efeknya akan memecahkan masalah tersebut.

Dalam matematika dan komputasi, algoritma atau algoritme merupakan kumpulan perintah untuk menyelesaikan suatu masalah. Kompleksitas dari suatu algoritma merupakan ukuran seberapa banyak komputasi yang dibutuhkan algoritma tersebut untuk menyelesaikan masalah. Secara informal, algoritma yang dapat menyelesaikan suatu permasalahan dalam waktu yang singkat memiliki kompleksitas yang rendah, sementara algoritma yang membutuhkan waktu lama untuk menyelesaikan masalahnya mempunyai kompleksitas yang tinggi (Simanullang, 2011).

C. Algoritma Boyer-Moore

Menurut Edward Rompah pada artikelnya yang membahas tentang algoritma Boyer-Moore, algoritma Boyer-Moore dipublikasikan oleh Robert S. Boyer, dan J. Strother Moore pada tahun 1977. Ide utama dari algoritma ini adalah dengan melakukan pencocokan dari paling kanan *string* yang dicari. Dengan menggunakan algoritma ini, secara rata-rata proses pencarian akan lebih cepat dibandingkan dengan proses pencarian lainnya. Ide dibalik algoritma ini adalah bahwa dengan memulai pencocokan karakter dari kanan, dan bukan dari kiri, maka akan lebih banyak informasi yang didapat (Mufthy, 2011).

D. Algoritma Turbo Boyer-Moore

Algoritma Turbo Boyer-Moore adalah sebuah variasi dari algoritma Boyer-Moore. Bila dibandingkan dengan algoritma Boyer-Moore, algoritma ini membutuhkan ruang lebih tapi tidak memerlukan pemrosesan ekstra. Ruang ekstra yang diperlukan berguna untuk mengingat faktor dari teks yang cocok dengan akhiran dari string yang dicari selama *attempt* terakir dan hanya jika *good-suffix shift* dilakukan (Charras & Lecroq, 2009).

Teknik ini mempunyai dua keunggulan (Charras & Lecroq, 2009):

1. Teknik ini memungkinkan untuk melompati faktor dari teks tersebut.
2. Teknik ini mengijinkan Pergeseran turbo.

E. Algoritma Tuned Boyer-Moore

Algoritma Tuned Boyer-Moore dapat dilihat

sebagai implementasi yang efisien algoritma Horspool (Jansen, Margraf, Mastrolilli, & Rolim, 2003). Fitur utama dari algoritma ini adalah simplifikasi dari algoritma Boyer-Moore, mudah untuk diimplementasikan, hanya menggunakan *bad-character shift*, dan sangat cepat dalam praktiknya (Lee & Cheng, 2007).

Pada dasarnya dalam algoritma ini yang menjadi fokus utama adalah karakter terakhir dari *pattern* dan menggeser *pattern* untuk mencocokan bagian paling akhir dari *pattern* tersebut (Lee & Cheng, 2007).

III. Analisis dan Perancangan

A. Metode Penelitian

Penelitian yang dilakukan melalui beberapa tahap. Tahap-tahap tersebut adalah studi literatur, perancangan *design* aplikasi, pembuatan atau pembangunan aplikasi, uji coba, dan kemudian pengumpulan data hasil.

B. Design Sistem

Sebagai garis besar dari apa yang harus dilakukan oleh sebuah aplikasi, diperlukan rancangan umum mengenai apa saja yang diperlukan terutama *input* dan *output* karena akan langsung berhubungan dengan *user*. Rancangan untuk *input* dan *output* yang diperlukan aplikasi yang dibangun tercantum sebagai berikut.

Masukan yang dibutuhkan oleh aplikasi:

1. *String* yang akan digunakan sebagai *string* dimana *keyword* akan dicari. *String* ini dapat ditulis secara manual maupun dengan *load* teks dari *text file* (doc, docx, php, html, txt). Untuk selanjutnya *string* ini akan disebut dengan *text*.
2. *String keyword* yang merupakan *string* yang akan dicari di dalam *text*. Untuk selanjutnya *string* ini akan disebut dengan *pattern*.

Output yang akan dihasilkan oleh aplikasi:

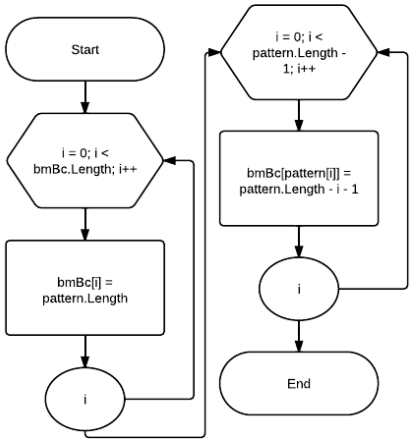
1. Jumlah total *pattern* yang ditemukan dalam *text*.
2. Waktu yang diperlukan oleh masing-masing algoritma untuk mencari pattern dalam text.
3. *Pattern* yang ditemukan di *text* akan ter-*highlight* dengan warna kuning.

C. Flowchart

Diagram alir atau *flowchart* merupakan bagan yang memperlihatkan urutan dan hubungan antar proses beserta instruksinya (Maing, 2008). Fungsinya adalah memberikan gambaran secara garis besar untuk program atau aplikasi yang dibuat.

C.1. Flowchart Proses Pembangunan Tabel Bad-Character Shift

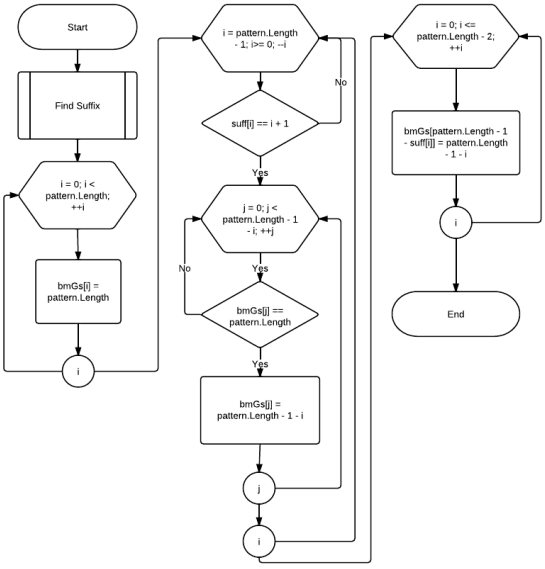
Tabel *bad-character shift* berisi nilai-nilai pergeseran yang dapat dilakukan untuk setiap karakter yang terdapat di alfabet, nomor, dan tanda baca yang umum digunakan. Setiap karakter yang ada di *pattern* diberi nilai sesuai dengan ukuran jauhnya karakter tersebut dari karakter paling akhir dari *pattern* dan untuk karakter yang tidak terdapat di dalam pattern akan diberi nilai yang sama yaitu panjang dari *pattern* yang dimasukkan.



Gambar 1. Flowchart proses pembangunan tabel bad-character shift

C.2. Flowchart Proses Pembangunan Tabel Good-Suffix Shift

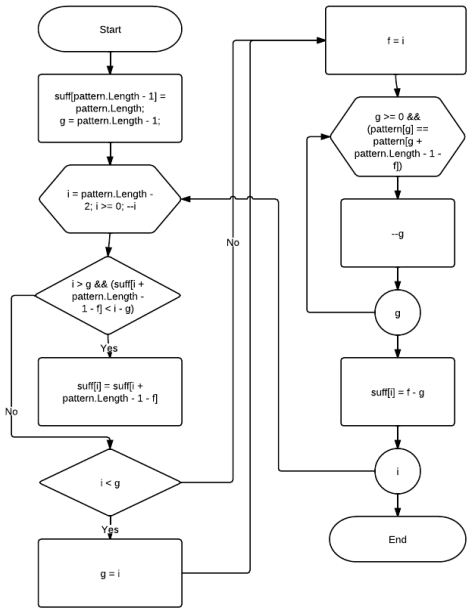
Tabel *good suffix shift* berisi nilai yang dapat digunakan untuk pergeseran ketika ketidakcocokan ditemukan berdasarkan karakter pada posisi keberapa yang menyebabkan ketidakcocokan. Untuk menentukan nilai-nilainya, perlu dihitung dulu tabel *suffix* yang fungsinya memberi tanda jika terdapat perulangan akhiran. Dari tabel *suffix* inilah tabel *good-suffix shift* akan dihitung. Nilai dari tiap karakter yang ada dalam *pattern* bergantung pada apakah adanya perulangan akhiran (*suffix*) atau tidak. Semakin banyak perulangan, semakin kecil nilai pergeseran.



Gambar 2. Flowchart proses pembangunan tabel good-suffix shift

C.3. Flowchart Proses Perhitungan Tabel Suffix

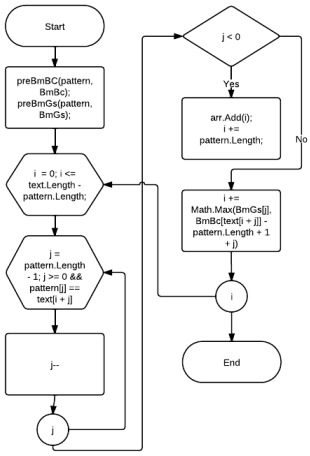
Gambar dibawah menunjukkan alur perhitungan tabel *suffixes* yang berguna untuk menghitung tabel *good-suffix shift* nantinya. Tabel ini berisi nilai-nilai dari tiap karakter yang ada di *pattern* yang menunjukkan adanya perulangan akhiran (*suffix*) atau tidak dan dimana letak perulangan tersebut sehingga ketika proses perhitungan tabel *good-isuffix shift*, dapat diketahui seberapa banyak karakter yang dapat digeser untuk pencocokan karakter yang selanjutnya.



Gambar 3. Flowchart proses perhitungan tabel suffix

C.4. Flowchart Algoritma Boyer-Moore

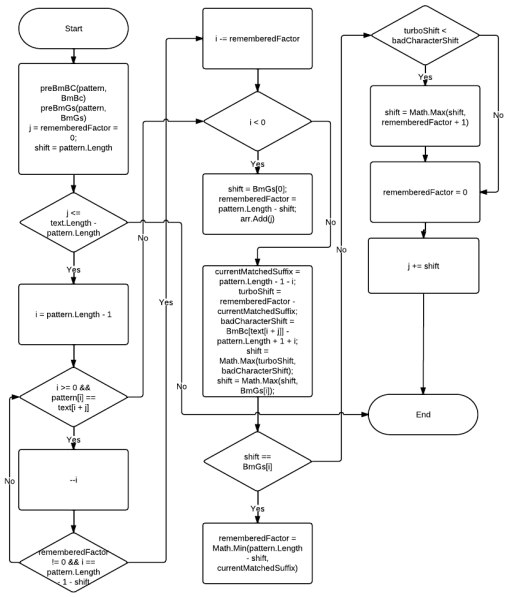
Dalam pencarian Boyer-Moore seperti yang ditunjukkan gambar 3.9 proses awal yang dilakukan adalah penempatan *window* dari *pattern* di *text* yang tersedia. Proses pencarian dimulai dari karakter paling kanan *pattern*. Setiap karakter akan dibandingkan satu per satu. Jika terjadi ketidakcocokan, maka akan dicek nilai pergeseran yang mungkin dilakukan ke tabel *good suffix shift* dan *bad character shift*. Nilai terbesar yang didapat akan diambil dan pergeseran *window* akan dilakukan sesuai dengan nilai tersebut.



Gambar 4. Algoritma Boyer-Moore

C.5. Algoritma Turbo Boyer-Moore

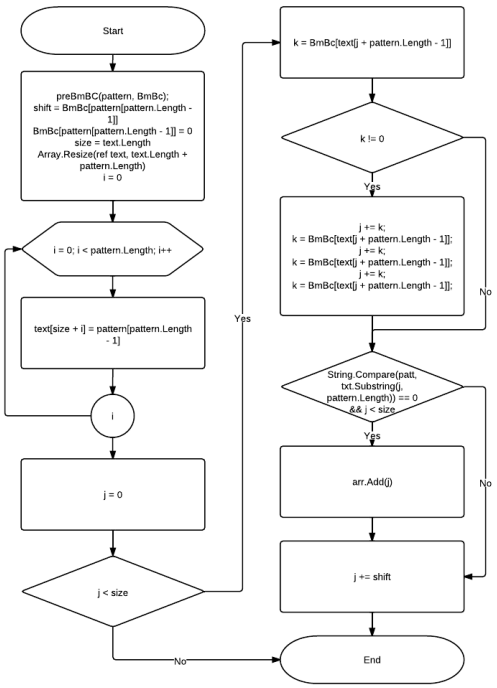
Untuk fase pencarian dalam algoritma Turbo Boyer-Moore, proses yang dilakukan hampir sama dengan fase pencarian pada algoritma Boyer-Moore. Yang membedakan adalah adanya variabel yang berfungsi untuk menampung nilai pergeseran apabila di putaran sebelumnya nilai yang diambil untuk pergeseran berasal dari tabel *good suffix shift*. Nilai ini nantinya akan digunakan sebagai kandidat nilai yang mungkin digunakan untuk melakukan pergeseran *pattern*.



Gambar 5. Algoritma Turbo Boyer-Moore

C.6. Algoritma Tuned Boyer-Moore

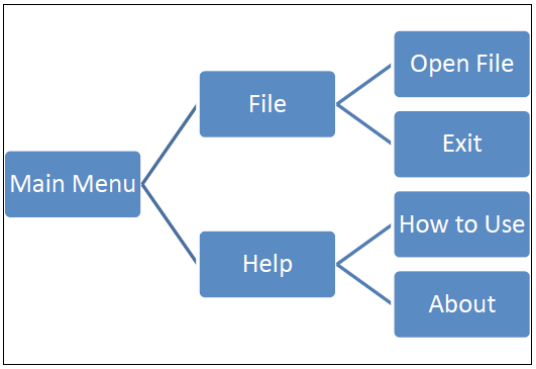
Algoritma Tuned Boyer-Moore akan melakukan tiga kali *blind-shift* pada tahap awal pencariannya. *Shift* yang dilakukan pada tiga kali *blind-shift* ini mengikuti aturan pergeseran sesuai tabel *bad-character shift*. Setelah tiga kali dilakukan *blind-shift*, karakter dari *pattern* akan mulai dibandingkan dengan *text*. Jika terjadi ketidakcocokan, *pattern* akan digeser lagi sesuai dengan nilai *shift* yang sebelumnya telah diinisialisasi.



Gambar 6. Algoritma Tuned Boyer-Moore

C.7. Rancangan Sistem

Berikut adalah rancangan navigasi dari aplikasi yang akan dibangun untuk penelitian.



Gambar 7. Rancangan navigasi aplikasi

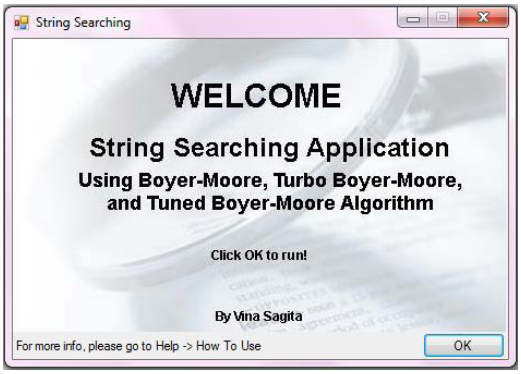
IV. Implementasi Sistem

A. Tampilan Sistem

Dari hasil rancangan yang telah dibuat, berikut adalah tampilan *graphical user interface* dari aplikasi yang dibangun.

A.1. Welcome Page

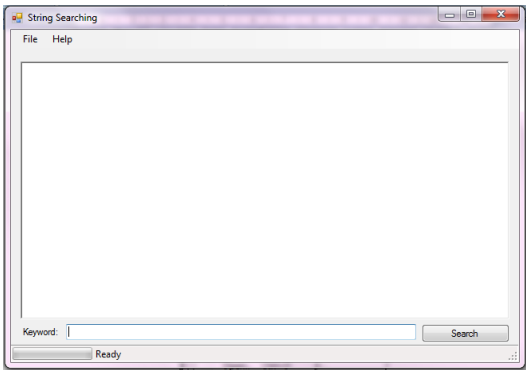
Welcome page merupakan halaman pertama yang muncul ketika aplikasi pertama kali dijalankan. Halaman ini memberi sedikit informasi tentang aplikasi.



Gambar 8. Welcome page

A.2. Halaman Utama Aplikasi

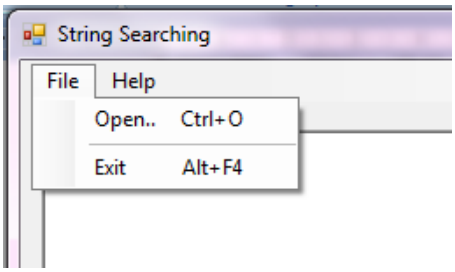
Halaman utama menampung *menu bar* yang dilengkapi beberapa pilihan navigasi, *text box* besar yang mampu menampung *multiple line text*, *text box* tempat dimana *pattern* dimasukkan, *button* untuk mencari *pattern* dalam *text*, dan *progress bar* yang memberi informasi pada *user* apa saja yang sedang terjadi dalam aplikasi.



Gambar 9. Halaman utama aplikasi

A.3. Sub-menu File

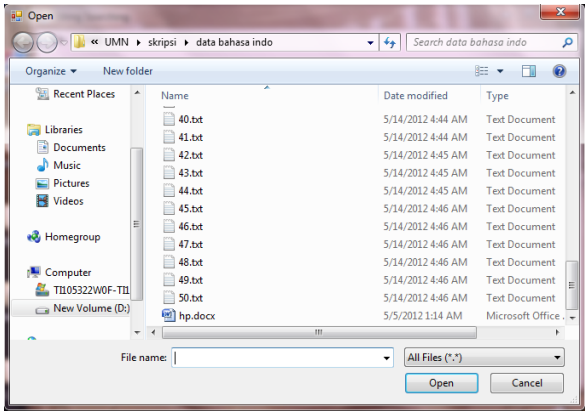
Pada menu File, terdapat dua pilihan yaitu Open dan Exit. Open berfungsi untuk membuka *window open file dialog*, sementara Exit berfungsi untuk keluar dari aplikasi.



Gambar 10. Sub-menu file

A.4. Open File Dialog

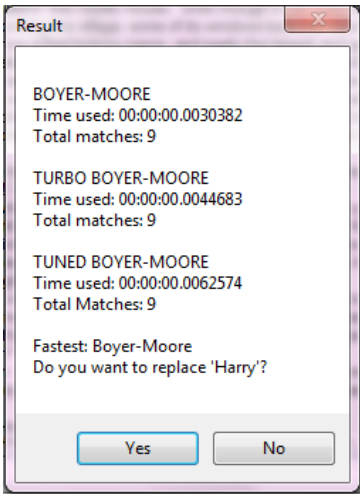
Window open file dialog adalah sebuah *window* dimana *user* dapat memilih *file* yang ingin digunakan. Isi dari *file* tersebut akan di-*copy* ke dalam *text box* yang berada di aplikasi.



Gambar 11. Open file dialog

A.5. Hasil Pencarian

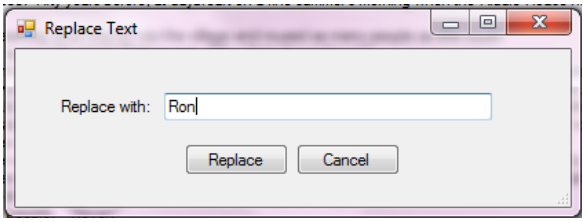
Setelah pencarian selesai, akan muncul *window* yang berisi informasi berupa berapa banyak *pattern* yang ditemukan dalam *text* dan waktu minimal yang diperlukan untuk menjalankan masing-masing algoritma, algoritma manakah yang memerlukan waktu paling cepat dalam pencarian, serta pertanyaan apakah kata yang tadi dicari ingin diubah menjadi kata lain atau yang biasanya disebut *replace*. Fitur *replace* disediakan untuk memfasilitasi *user* yang mencari suatu kata dengan tujuan untuk menggantinya dengan kata lain.



Gambar 12. Hasil pencarian

A.6. Form Replace

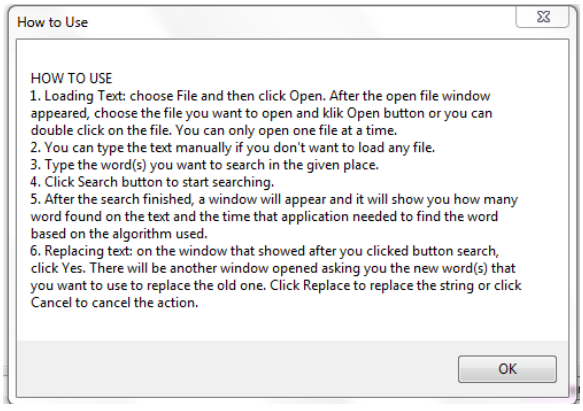
Jika *user* ingin mengubah kata yang dicari sebelumnya maka *window* baru akan muncul dan meminta kata baru yang ingin digunakan untuk mengubah kata yang lama. Ketika tombol Replace ditekan maka kata yang sebelumnya ingin diganti akan diganti dengan kata baru yang dimasukkan oleh *user*.



Gambar 13. Form replace

A.7. Halaman How to Use

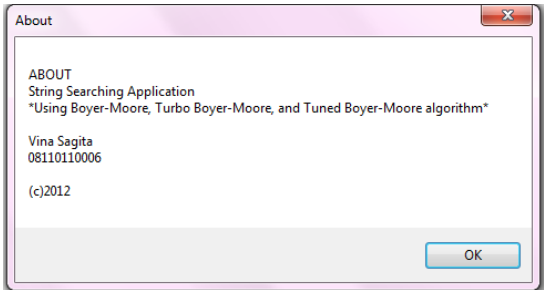
Halaman *How to Use* akan menampilkan *window* yang berisi langkah-langkah penggunaan aplikasi.



Gambar 14. Halaman how to use

A.8. Halaman About

Halaman *About* berisi sedikit informasi mengenai aplikasi tersebut serta identitas pembuat aplikasi.



Gambar 15. Halaman about

V. Simpulan dan Saran

A. Simpulan

Tujuan yang ingin dicapai dari penelitian ini adalah mengetahui mana algoritma yang paling cepat diantara 3 algoritma yang ditentukan dan hal ini sudah terjawab.

Dari hasil penelitian yang didapat, bisa disimpulkan bahwa algoritma Boyer-Moore memiliki waktu pencarian tercepat dari tiga varian Boyer-Moore yang telah ditentukan. Algoritma Turbo Boyer-Moore merupakan algoritma tercepat kedua dan yang paling lambat adalah Tuned Boyer-Moore.

B. Saran

Aplikasi dapat dikembangkan lebih lanjut dengan menambahkan algoritma-algoritma pencarian *string* lain seperti algoritma Horspool dan algoritma Zhu-Takaoka sehingga pengguna bisa mendapatkan lebih banyak informasi tentang kecepatan masing-masing algoritma. Selain itu dapat dikembangkan pula teknik untuk mengukur waktu yang lebih *advanced* sehingga hasil yang ditampilkan aplikasi tidak berubah-ubah.

Daftar Pustaka

[1] Charras, Christian & Thierry Lecroq. 2009. "Handbook of Exact-String Matching Algorithms". URL: http://ebookee.org/Handbook-of-Exact-String-Matching-Algorithms_892540.html. Diakses tanggal 28 Februari 2012.

[2] Fauzy, Mufthy. 2011. "Sekilas Tentang Ilmu Komputer & Informatika". URL: http://mufthyrocket.blogspot.com/2011/11/sekilas-tentang-ilmu-komputer.html. Diakses tanggal 24 Januari 2012.

[3] Hartoyo, Eko, Yus Vembrina, dkk. 2010. "Analisis Algoritma Pencarian String (String Matching)". URL: http://www.docstoc.com/docs/40337710/Analisis-Algoritma-Pencarian-String-(-String-Matching-). Diakses tanggal 12 Desember 2011.

[4] Irwanti, Silvina. 2010. Implementasi Algoritma Backtracking dalam Perancangan Perangkat Lunak Game Anagram. Skripsi: Jurusan Ilmu Komputer, Fakultas Matematika dan Ilmu Pengetahuan Alam, Universitas Sumatra Utara.

[5] Jansen, Klaus, Marian Margraf, dkk. 2003. Experimental and Efficient Algorithms. Switzerland: Springer.

[6] Kumara, Gozali Harda. 2009. Visualisasi Beberapa Algoritma Pencocokan String dengan Java. Jurnal: Jurusan Teknik Informatika, Fakultas Elektro Informatika, Institut Teknologi Bandung.

[7] Lee, R. C. T. & C. W. Cheng. 2007. Tuned Boyer Moore Algorithm [Power Point Slides]. Diunduh dari http://www.slidefinder.net/t/tuned_boyer_moore_algorithm_adviser/tunedbm/11229673.

[8] Maing, Irwan. 2008. "Pengertian Dasar dan Simbol Flowchart". URL: http://irma14.blogspot.com/2008/09/pengertian-dasar-dan-simbol-flowchart.html. Diakses tanggal 26 April 2012.

[9] Purwoko, Petrus Dwi. 2006. "Perbandingan Algoritma Turbo BM, Algoritma Quick Search dan Algoritma Shift-Or". URL: http://elib.unikom.ac.id/gdl.php?mod=browse&op=read&id=jbptunikompp-gdl-s1-2006-petrusdwip-2790. Diakses tanggal 10 Desember 2011.

[10] Rompah, Edward. 2009. "Algoritma Boyer-Moore". URL: http://edwardgr.wordpress.com/2009/01/06/algoritma-boyer-moore. Diakses tanggal 18 Januari 2012.

[11] Rompah, Edward. 2009. "Contoh Kasus Sederhana Penerapan Algoritma Boyer-Moore". URL: http://edwardgr.wordpress.com/2009/10/19/contoh-kasus-sederhana-penerapan-algoritma-boyer-moore/. Diakses tanggal 18 Januari 2012.

[12] Simanullang, Okto Duapan. 2011. "Sejarah Algoritma dan Pemrograman". URL: http://okto-sumberdayaalam-okto.blogspot.com/2011/04/sejarah-algoritma-dan-pemograman.html. Diakses tanggal 27 Mei 2012.

[13] Sulistyo, Imam, Adie Pratipto, dkk. 2006. "Algoritma Boyer-Moore dalam Pencarian String". URL: http://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2005-2006/Makalah2006/MakalahStmik2006-54.pdf. Diakses tanggal 10 Desember 2011.

[14] Wiramuda, Amirul. 2008. "Algoritma pencarian string". URL: http://amnemonix.wordpress.com/2010/11/28/algoritma-pencarian-string/. Diakses tanggal 24 Januari 2012.