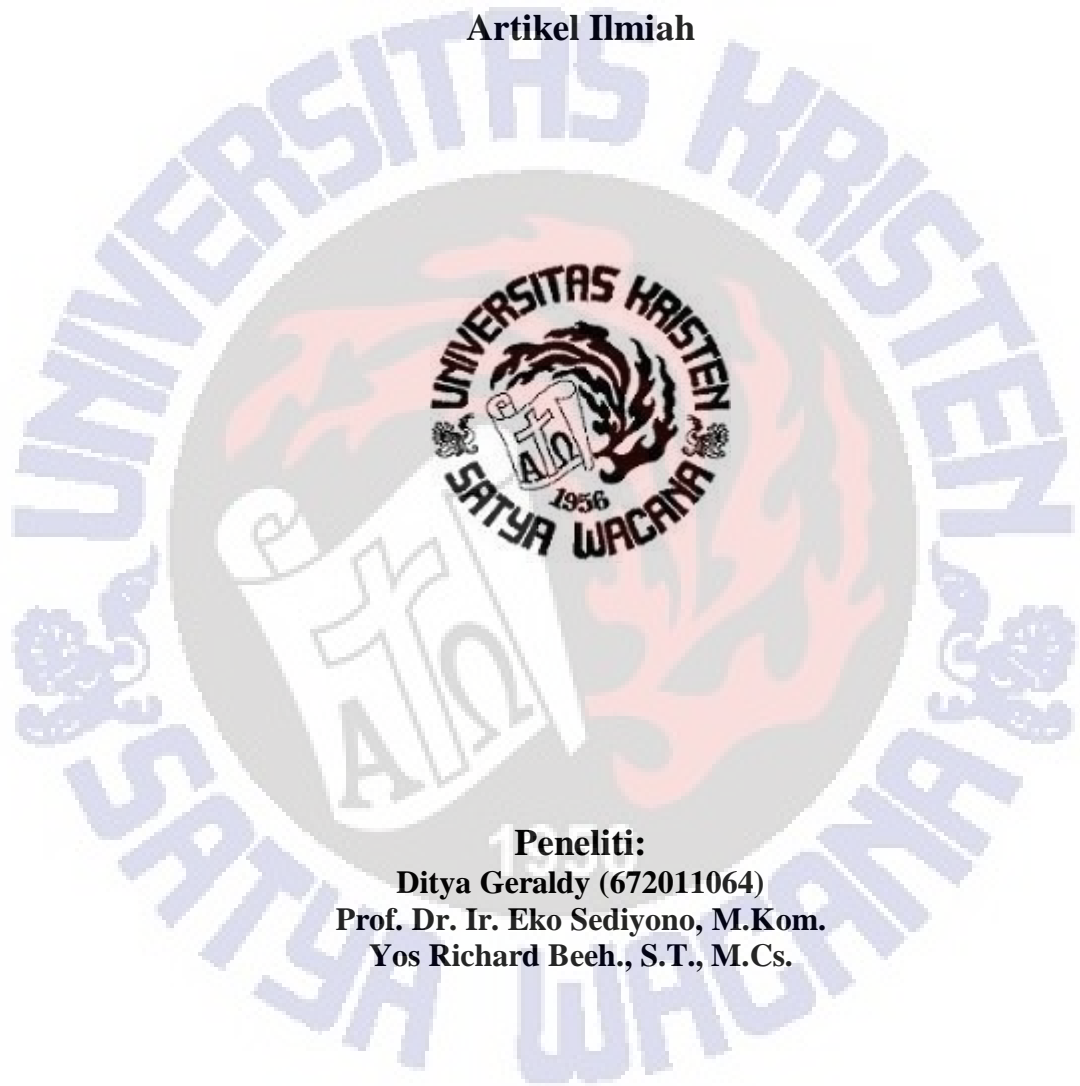


Studi Perbandingan Algoritma Brute Force, Algoritma Knuth-Morris-Pratt, Algoritma Boyer-Moore untuk Identifikasi Kesalahan Penulisan Teks berbasis Android

Artikel Ilmiah



Peneliti:

**Ditya Geraldny (672011064)
Prof. Dr. Ir. Eko Sedyono, M.Kom.
Yos Richard Beeh., S.T., M.Cs.**

**Program Studi Teknik Informatika
Fakultas Teknologi Informasi
Universitas Kristen Satya Wacana
Salatiga
2016**

Studi Perbandingan Algoritma Brute Force, Algoritma Knuth-Morris-Pratt, Algoritma Boyer-Moore untuk Identifikasi Kesalahan Penulisan Teks berbasis Android

Artikel Ilmiah

**Diajukan kepada
Fakultas Teknologi Informasi
untuk memperoleh gelar Sarjana Komputer**



Peneliti:

**Ditya Geraldny (672011064)
Prof. Dr. Ir. Eko Sedyono, M.Kom.
Yos Richard Beeh., S.T., M.Cs.**

**Program Studi Teknik Informatika
Fakultas Teknologi Informasi
Universitas Kristen Satya Wacana
Salatiga
Juni 2016**



PERPUSTAKAAN UNIVERSITAS
UNIVERSITAS KRISTEN SATYA WACANA
Jl. Diponegoro 52 – 60 Salatiga 50711
Jawa Tengah, Indonesia
Telp. 0298 – 321212, Fax. 0298 321433
Email: library@adm.uksw.edu ; http://library.uksw.edu

PERNYATAAN TIDAK PLAGIAT

Saya yang bertanda tangan di bawah ini:

Nama : Ditya Gerald
NIM : 672011064 Email : 672011064@student.uksw.edu
Fakultas : TEKNOLOGI INFORMASI Program Studi : TEKNIK INFORMATIKA
Judul tugas akhir : STUDI PERBANDINGAN ALGORITMA BRUTEFORCE, ALGORITMA KNOOTH-MORRIS-PRATT, ALGORITMA BOYER-MOORE UNTUK IDENTIFIKASI KESELAHAN PENULISAN TEKS BERBASIS ANDROID.
Pembimbing : 1. PROF. DR. IR. EKO SEDIYONO, M.KOM
2. YOS RICHARD BEEH, S.T, M.CS.

Dengan ini menyatakan bahwa:

1. Hasil karya yang saya serahkan ini adalah asli dan belum pernah diajukan untuk mendapatkan gelar kesarjanaan baik di Universitas Kristen Satya Wacana maupun di institusi pendidikan lainnya.
2. Hasil karya saya ini bukan saduran/terjemahan melainkan merupakan gagasan, rumusan, dan hasil pelaksanaan penelitian/implementasi saya sendiri, tanpa bantuan pihak lain, kecuali arahan pembimbing akademik dan narasumber penelitian.
3. Hasil karya saya ini merupakan hasil revisi terakhir setelah diujikan yang telah diketahui dan disetujui oleh pembimbing.
4. Dalam karya saya ini tidak terdapat karya atau pendapat yang telah ditulis atau dipublikasikan orang lain, kecuali yang digunakan sebagai acuan dalam naskah dengan menyebutkan nama pengarang dan dicantumkan dalam daftar pustaka.

Pernyataan ini saya buat dengan sesungguhnya. Apabila di kemudian hari terbukti ada penyimpangan dan ketidakbenaran dalam pernyataan ini maka saya bersedia menerima sanksi akademik berupa pencabutan gelar yang telah diperoleh karena karya saya ini, serta sanksi lain yang sesuai dengan ketentuan yang berlaku di Universitas Kristen Satya Wacana.

Salatiga, 23 JUNI 2016

METERAI TEMPEL
FA656ADF601719414
6000
ENAM RIBURUPIAH
DITYA GERALD
Tanda tangan & nama terang mahasiswa



PERNYATAAN PERSETUJUAN AKSES

Saya yang bertanda tangan di bawah ini:

Nama : Ditya Gerald
NIM : 672011064 Email : 672011064@student.uksw.edu
Fakultas : TEKNOLOGI INFORMASI Program Studi : TEKNIK INFORMATIKA
Judul tugas akhir : Studi perbandingan Algoritma Brute Force, Algoritma
Knuth-Morris-Pratt, Algoritma Boyer-Moore untuk
identifikasi kesalahan penulisan teks Berbasis Android.

Dengan ini saya menyerahkan hak non-eksklusif* kepada Perpustakaan Universitas – Universitas Kristen Satya Wacana untuk menyimpan, mengatur akses serta melakukan pengelolaan terhadap karya saya ini dengan mengacu pada ketentuan akses tugas akhir elektronik sebagai berikut (beri tanda pada kotak yang sesuai):

- ☒ a. Saya mengizinkan karya tersebut diunggah ke dalam aplikasi Repositori Perpustakaan Universitas, dan/atau portal GARUDA
- ☐ b. Saya tidak mengizinkan karya tersebut diunggah ke dalam aplikasi Repositori Perpustakaan Universitas, dan/atau portal GARUDA**

* Hak yang tidak terbatas hanya bagi satu pihak saja. Pengajar, peneliti, dan mahasiswa yang menyerahkan hak non-eksklusif kepada Repositori Perpustakaan Universitas saat mengumpulkan hasil karya mereka masih memiliki hak copyright atas karya tersebut.

** Hanya akan menampilkan halaman judul dan abstrak. Pilihan ini harus dilampiri dengan penjelasan/ alasan tertulis dari pembimbing I dan diketahui oleh pimpinan fakultas (dekan/kaprodi).

Demikian pernyataan ini saya buat dengan sebenarnya.

Salatiga, 23 Juni 2016

1956

Ditya Gerald

Tanda tangan & nama terang mahasiswa

Mengetahui,

Prof. Dr. Ir. Eko Sedyono, M.Kom.

Tanda tangan & nama terang pembimbing I

Yos Richard Beeh, S.T., M.Sc.

Tanda tangan & nama terang pembimbing II

Studi Perbandingan Algoritma Brute Force, Algoritma Knuth-Morris-Pratt, Algoritma Boyer-Moore untuk Identifikasi Kesalahan Penulisan Teks berbasis Android

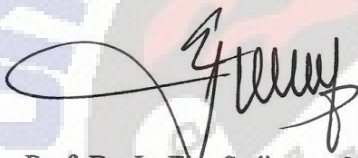
Oleh,

Ditya Geraldly
NIM : 672011064

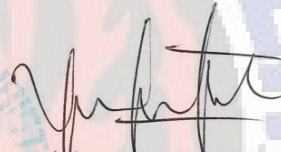
Artikel Ilmiah

Diajukan Kepada Program Studi Teknik Informatika, Fakultas Teknologi Informasi guna memenuhi sebagian dari persyaratan untuk mencapai gelar Sarjana Komputer

Disetujui oleh,

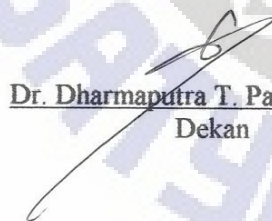


Prof. Dr. Ir. Eko Sedyono, M.Kom.
Pembimbing 1



Yos Richard Beeh, S.T., M.Cs.
Pembimbing 2

Diketahui oleh,



Dr. Dharmaputra T. Palekahelu, M.Pd.
Dekan



Suprihadi, S.Si., M.Kom.
Ketua Program Studi

FAKULTAS TEKNOLOGI INFORMASI
UNIVERSITAS KRISTEN SATYA WACANA
SALATIGA
2016

Lembar Pengesahan

Judul Tugas Akhir : Studi Perbandingan Algoritma Brute Force, Algoritma Knuth- Morris-Pratt, Algoritma Boyer-Moore untuk Identifikasi Kesalahan Penulisan Teks Berbasis Android
Nama Mahasiswa : Ditya Geraldy
NIM : 672011064
Program Studi : Teknik Informatika
Fakultas : Teknologi Informasi

Menyetujui,

Prof. Dr. Ir. Eko Sedyono, M.Kom.
Pembimbing 1

Yos Richard Beeh, S.T., M.Cs.
Pembimbing 2

Mengesahkan,

Dr. Dharmaputra T. Palekahelu, M.Pd.
Dekan

Supriyadi, S.Si., M.Kom.
Ketua Program Studi

Dinyatakan Lulus Ujian tanggal: 10 Juni 2016

Penguji:

1. Alz Danny Wowor, S.Si., M.Cs.
2. Radius Tanone, S.Kom., M.Cs.



FAKULTAS TEKNOLOGI INFORMASI
UNIVERSITAS KRISTEN SATYA WACANA
Jalan Diponegoro 52 - 60
Phone: (0298) 321212 (Hunting)
Fax: (0298) 321433
E-mail: fti@uksw.edu
Salatiga 50711 - INDONESIA



LEMBAR PERSETUJUAN PUBLISH JURNAL

Dengan mempertimbangkan isi dari jurnal mahasiswa :

Nama Mahasiswa :
NIM :

Ditya Geraldny
672011064

Maka jurnal ini dinyatakan :

LAYAK TERBIT / TIDAK LAYAK TERBIT

Menyetujui,

(.....)

Pembimbing 1

Prof. Dr. Ir. Eko Sedyono, M.kom

(.....)

Pembimbing 2

Yos Richard Beeh, S.T., M.Cs.

(.....)

Penguji 1

Alz Danny Wowor, S.Si., M.Cs.

Penguji 2

Radius Tanorie, S.kom., M.Cs.

**PERNYATAAN PERSETUJUAN PUBLIKASI
TUGAS AKHIR UNTUK KEPENTINGAN AKADEMIS**

Sebagai sivitas akademik Universitas Kristen Satya Wacana (UKSW), saya yang bertanda tangan di bawah ini:

Nama : Ditya Gerald
NIM : 672011064
Program-studi : Teknik Informatika
Fakultas : Teknologi Informasi
Jenis karya : Skripsi/ ~~Tesis/ Disertasi~~ (Coret yang tidak sesuai)

demi pengembangan ilmu pengetahuan, menyetujui untuk memberikan kepada UKSW Hak bebas royalti non-eksklusif (*Non-exclusive royalty free right*) atas karya ilmiah saya yang berjudul:

Studi perbandingan Algoritma Brute Force, Algoritma Knuth-Morris-Pratt, Algoritma Boyer-Moore untuk Identifikasi Kesalahan Penulisan Teks berbasis Android.

berserta perangkat yang ada (jika diperlukan).

Dengan hak bebas royalti non-eksklusif ini, UKSW berhak menyimpan, mengalihmedia/ formatkan, mengelola dalam bentuk pangkalan data, merawat, dan mempublikasikan tugas akhir saya, selama tetap mencantumkan nama saya sebagai penulis/ pencipta dan sebagai pemilik hak cipta.

Demikian pernyataan ini saya buat dengan sebenarnya.

Dibuat di : Salatiga
Pada tanggal : 15 Juni 2016

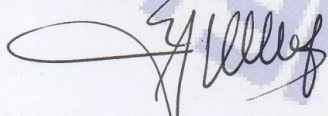
Yang menyatakan



Ditya Gerald

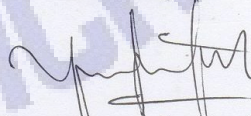
Mengetahui,

Pembimbing I



Prof. Dr. Ir. Eko Sedyono, M.Kom.

Pembimbing II



Yos Richard Beem, S.T., M.Sc.

Pernyataan

Yang bertandatangan di bawah ini,

Nama : Ditya Geraldly

NIM : 672011064

Program Studi : Teknik Informatika

Fakultas : Teknologi Informasi, Universitas Kristen Satya Wacana

menyatakan dengan sesungguhnya bahwa tugas akhir dengan judul Studi Perbandingan Algoritma Brute Force, Algoritma Knuth-Morris-Pratt, Algoritma Boyer-Moore untuk Identifikasi Kesalahan Penulisan Teks berbasis Android yang dibimbing oleh:

1. Prof. Dr. Ir. Eko Sedyono, M.Kom.
2. Yos Richard Beeh., S.T., M.Cs.,

adalah benar-benar hasil karya saya.

Di dalam tugas akhir ini tidak terdapat keseluruhan atau sebagian tulisan atau gagasan orang lain yang saya ambil dengan cara menyalin atau meniru dalam bentuk rangkaian kalimat atau gambar serta simbol yang saya akui seolah-olah sebagai karya saya tanpa memberikan pengakuan pada penulis atau sumber aslinya.

Salatiga, 15 Juni 2016

Yang memberi pernyataan,



Ditya Geraldly

1. Pendahuluan

Kesalahan penulisan teks dalam perangkat *mobile* sering ditemukan, karena banyaknya pengguna yang mengutamakan kecepatan dalam pengetikan. Padahal dalam berbagai keadaan, kesalahan penulisan teks dapat menimbulkan berbagai macam dampak. Misalnya kesalahan penulisan teks dalam *short message service*, dapat mengakibatkan informasi yang disampaikan pengirim tidak dapat dimengerti oleh penerima. Tentunya kesalahan penulisan teks disebabkan oleh kesalahan pengetikan yang disebabkan oleh beberapa faktor^[1], antara lain letak huruf pada *keyboard* yang berdekatan, kesalahan karena slip pada tangan atau jari, kesalahan mekanis, kesalahan yang disebabkan oleh ketidaksengajaan. Pengecekan kesalahan penulisan teks secara manual dalam perangkat *mobile* dapat dilakukan. Namun, efisiensi waktu yang dibutuhkan dan kemungkinan adanya *human error* dapat mengakibatkan proses pengecekan kata menjadi tidak optimal. Karena itu diperlukan suatu sistem untuk mendeteksi kesalahan penulisan teks dalam perangkat *mobile*. Salah satu pendekatan yang dapat diterapkan untuk mengidentifikasi kesalahan penulisan teks adalah metode pencarian *string*^[2].

Algoritma yang digunakan untuk pencarian *string* bermacam-macam dan semakin berkembang dari hari ke hari. Tujuan utamanya untuk mencari *string* se-efisien mungkin. Algoritma Brute Force merupakan algoritma pencarian *string* yang termudah dan berguna untuk studi pembandingan. Namun pada berbagai keadaan, langkah-langkah yang dilakukan algoritma Brute Force dalam menemukan solusi permasalahan tidak berguna atau sia-sia sehingga dilakukan pengembangan algoritma seperti yang dilakukan oleh Donald Knuth, Joseph Morris dan Vaughan Pratt yang dikenal sebagai algoritma Knuth-Morris-Pratt^[3]. Algoritma Boyer Moore merupakan algoritma pencarian *string* yang telah banyak dikenal oleh masyarakat dan dianggap paling efisien untuk pencarian *string*^[4].

Banyaknya algoritma pencarian *string*, membuka peluang untuk membandingkan performa algoritma-algoritma pencarian *string* dalam berbagai sistem operasi. Android merupakan sistem operasi *mobile* yang paling banyak digunakan di dunia^[5]. Perkembangan aplikasi berbasis Android sangat maju dari tahun ke tahun, sehingga dibutuhkan perbandingan algoritma-algoritma sebagai referensi pemilihan algoritma yang lebih baik untuk pembuatan aplikasi berbasis Android. Perbandingan beberapa algoritma pencarian *string* seperti algoritma Brute Force, algoritma Knuth-Morris-Pratt, algoritma Boyer-Moore belum pernah dilakukan dalam aplikasi Android. Sehingga diperlukan sistem untuk membandingkan algoritma pencarian *string* dalam aplikasi Android.

Berdasarkan latar belakang tersebut, rumusan masalah pada penelitian ini adalah bagaimana membandingkan algoritma pencarian *string* Brute Force, Knuth-Morris-Pratt dan Boyer Moore untuk identifikasi kesalahan penulisan teks berbasis Android? Adapun perbandingan berdasarkan waktu yang dibutuhkan setiap algoritma pencarian *string* dan data yang bersumber dari kata dasar (lema) dalam Kamus Besar Bahasa Indonesia.

2. Tinjauan Pustaka

Penelitian terdahulu yang berkaitan dengan penelitian ini berjudul “Perbandingan String Searching Brute Force, Knuth-Morris-Pratt, Boyer-Moore dan Karp Rabin pada Teks Alkitab Bahasa Indonesia”. Pada penelitian ini membahas hubungan panjang pola (*pattern*) yang dicari terhadap waktu penemuannya. Hasil dari penelitian ini adalah semakin panjang pola (*pattern*), waktu pencarian algoritma Brute Force tetap, waktu pencarian algoritma Knuth-Morris-Pratt dan Karp Rabin cenderung meningkat, waktu pencarian algoritma Boyer-Moore semakin singkat. Untuk pencarian pola (*pattern*) dalam teks Alkitab algoritma Boyer-Moore paling cepat dibandingkan algoritma Brute Force, Knuth-Morris-Pratt dan Karp Rabin. Perbedaan penelitian ini dengan penelitian yang dikerjakan adalah penelitian ini berbasis *desktop* dengan sistem operasi Windows XP, sedangkan penelitian yang dikerjakan berbasis *mobile* dengan sistem operasi Android. Manfaat bagi penelitian yang dikerjakan adalah sebagai referensi perbandingan algoritma Brute Force, Knuth-Morris-Pratt, Boyer-Moore.

Penelitian terdahulu yang berkaitan dengan penelitian ini berjudul “Studi Perbandingan Implementasi Algoritma Boyer-Moore, Turbo Boyer-Moore, dan Tuned Boyer-Moore dalam Pencarian String”. Pada penelitian ini membahas perbandingan kinerja dari tiga varian algoritma Boyer-Moore. Hasil dari penelitian ini adalah algoritma Boyer-Moore memiliki kinerja yang paling cepat, algoritma Turbo Boyer-Moore tercepat kedua dan yang paling lambat adalah Tuned Boyer-Moore. Perbedaan penelitian ini dengan penelitian yang dikerjakan adalah algoritma-algoritma pencarian *string* yang dibandingkan. Manfaat bagi penelitian yang dikerjakan adalah acuan penggunaan waktu yang dibutuhkan setiap algoritma dalam pencarian *string* sebagai tolak ukur perbandingan.

Algoritma adalah urutan langkah-langkah logis pada penyelesaian masalah yang disusun secara sistematis. Algoritma pencarian *string* adalah algoritma untuk melakukan pencarian semua kemunculan *string* pendek pada *string* yang lebih panjang^[7]. Algoritma Brute Force melakukan pencocokan satu persatu (dari kiri ke kanan) karakter di *pattern* dengan karakter di *text*^[6]. Misalkan T menyatakan *text*, P menyatakan *pattern*, n menyatakan panjang *text*, m menyatakan panjang *pattern*, x menyatakan indeks *text* (0 sampai n-1), y menyatakan indeks *pattern* (0 sampai m-1), kotak pencarian diasumsikan sebagai kotak di dalam *text* sepanjang dengan panjang *pattern* untuk mewakili langkah-langkah pencocokan *string*. Selanjutnya, mula-mula posisi kotak pencarian diletakan di posisi paling kiri. Lalu dilakukan perbandingan pertama antara T[0] dan P[0], jika terjadi kecocokan maka indeks *text* dan *pattern* ditambah 1. Kemudian dilakukan perbandingan selanjutnya sampai batas kanan kotak pencarian sama dengan batas kanan *text*. Jika y sama dengan m-1 maka *pattern* ditemukan dalam *text*.

	T	H	I	S	I	S	A	S	I	M	P	L	E	L	I	N	E	<< text
1	S	I	M	P	L	E												<< pattern
2		S	I	M	P	L	E											
3			S	I	M	P	L	E										
4				S	I	M	P	L	E									
5					S	I	M	P	L	E								
6						S	I	M	P	L	E							
7							S	I	M	P	L	E						
8								S	I	M	P	L	E					
9									S	I	M	P	L	E				
10										S	I	M	P	L	E			
11											S	I	M	P	L	E		
12												S	I	M	P	L	E	
13													S	I	M	P	L	E
14														S	I	M	P	L
15															S	I	M	P
16																S	I	M
17																	S	I
18																		S

Gambar 1 Contoh pencarian menggunakan algoritma Brute Force.

Algoritma Knuth-Morris-Pratt

Algoritma Knuth-Morris-Pratt (KMP) melakukan pencocokan karakter dari kiri ke kanan dengan menyingkat jumlah perbandingan yang dilakukan oleh algoritma Brute Force dengan cara menghitung dari dimulai dari ketidakcocokan ditemukan, dari ketidakcocokan tersebut akan dihitung dari mana pencarian selanjutnya dimulai. Apabila terjadi ketidakcocokan karakter pada *pattern* dengan karakter pada *text*, maka dicari panjang pergeserannya yang ditentukan dalam dfa (deterministic finite automaton)^[8].

	T	H	I	S	I	S	A	S	I	M	P	L	E	L	I	N	E	<< teks
1	S	I	M	P	L	E												<< pattern
2		S	I	M	P	L	E											
3			S	I	M	P	L	E										
4				S	I	M	P	L	E									
5					S	I	M	P	L	E								
6						S	I	M	P	L	E							
7							S	I	M	P	L	E						
8								S	I	M	P	L	E					
9									S	I	M	P	L	E				
10										S	I	M	P	L	E			
11											S	I	M	P	L	E		
12												S	I	M	P	L	E	
13													S	I	M	P	L	E

Gambar 2 Contoh pencarian menggunakan algoritma Knuth-Morris-Pratt.

Algoritma Boyer-Moore

Algoritma Boyer-Moore adalah salah satu algoritma pencarian string yang dikembangkan oleh Robert S. Boyer, J. Stroher Moore dan dipublikasikan pada tahun 1977. Pada dasarnya cara kerja algoritma ini mirip dengan algoritma Knuth-Morris-Pratt dimana kedua algoritma ini akan melakukan lompatan pengecekan dalam proses pencarian *string*. Namun berbeda dengan algoritma KMP, algoritma Boyer Moore melakukan pencocokan karakter mulai dari kanan ke kiri^[9].



Gambar 3 Contoh pencarian menggunakan algoritma Boyer-Moore.

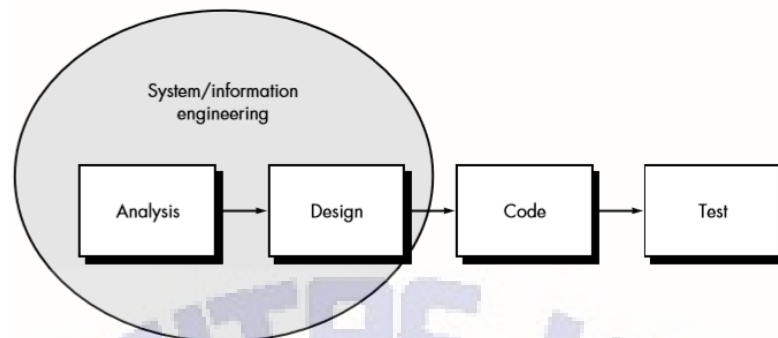
Berdasarkan Gambar 3, pencocokan dimulai dari karakter *pattern* yang paling kanan. Pada langkah 1 dilakukan perbandingan, ditemukan ketidakcocokan karakter S pada *text* dan karakter E pada *pattern*. Kemudian dilakukan pencarian (dari kanan ke kiri) karakter S dalam *pattern*, dari karakter E sampai S dihitung panjang lompatan untuk langkah berikutnya. Panjang lompatan disimpan dalam *array right*. Pada langkah 2, *pattern* digeser sepanjang nilai *array right*. Selanjutnya dilakukan pencocokan kembali dari karakter *pattern* paling kanan, sampai semua karakter *pattern* cocok dalam karakter dalam *text*.

Android

Android adalah sistem operasi *mobile* yang berbasis kernel Linux dan dikembangkan oleh Google. Android dirancang terutama untuk perangkat *mobile* yang menggunakan *touchscreen* seperti *smartphone* dan komputer *tablet*. Android dioperasikan dengan *user interface* berupa sentuhan seperti *wiping*, *taping*, *draging*. Android adalah sistem operasi *mobile* yang bersifat *open source*. Google merilis kodenya di bawah lisensi Apache, yang memungkinkan *software* dalam Android untuk dimodifikasi secara bebas dan didistribusikan oleh para pembuat perangkat, operator nirkabel dan *developer*^[10].

3. Metode dan Perancangan Sistem

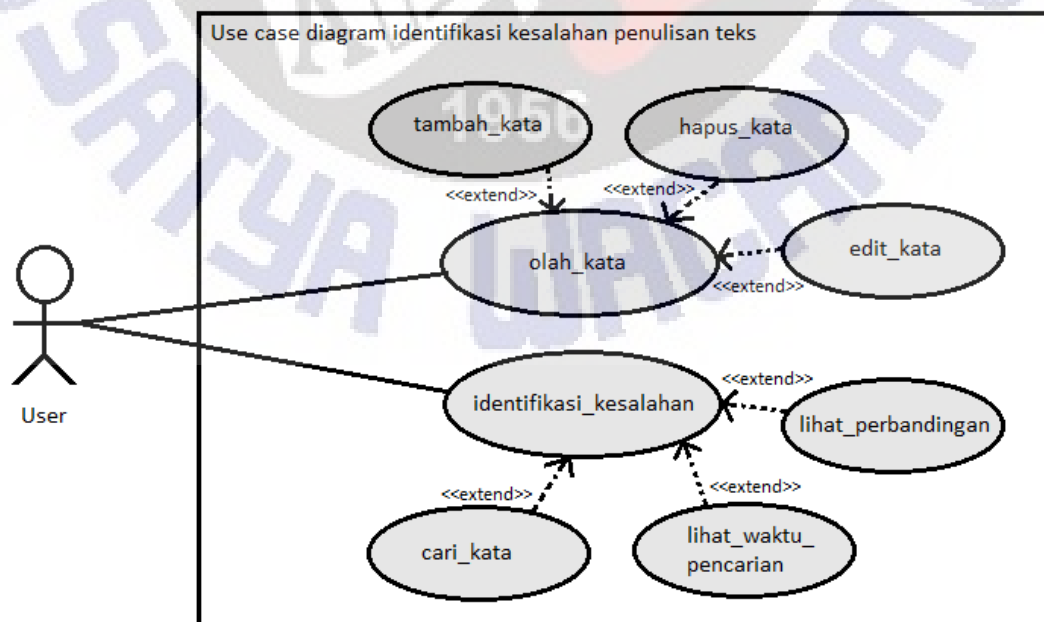
Penelitian yang dilakukan menerapkan metodologi pengembangan perangkat lunak, dengan menggunakan model sekuensial linier yang terbagi dalam empat tahapan^[11], yaitu: (1) Analisis, (2) Desain, (3) Pengkodean, (4) Pengujian.



Gambar 4 Model sekuensial linier^[11].

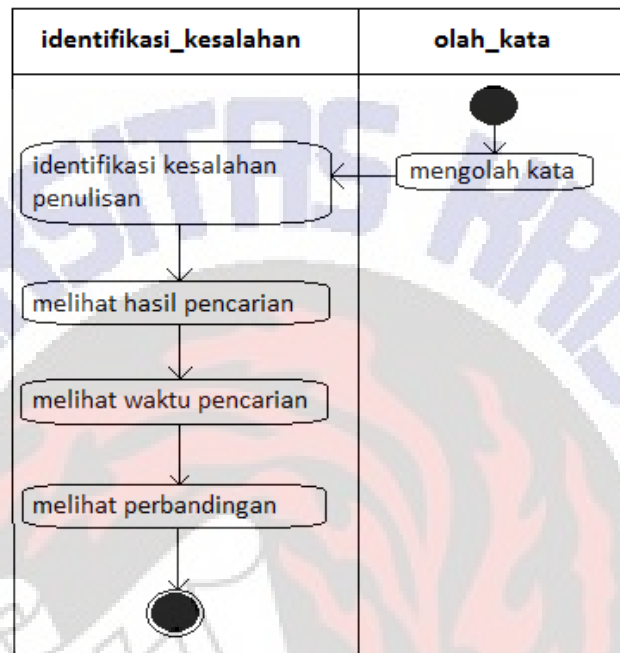
Penerapan tahapan penelitian, model sekuensial pada Gambar 4 dapat dijelaskan sebagai berikut. Tahap pertama: analisis, mengkaji algoritma Brute Force, Knuth-Morris-Pratt dan Boyer-Moore, dan kebutuhan yang diperlukan. Tahap kedua: desain, merancang sistem untuk menerapkan algoritma-algoritma pencarian *string* yang digunakan. Tahap ketiga: pengkodean, membuat aplikasi berdasarkan perancangan sistem pada tahap kedua. Aplikasi yang dibuat dalam bentuk *mobile* berbasis Android. *Integrated Development Environment* (IDE) yang digunakan adalah Eclipse dengan bahasa pemrograman Java. *Database* yang digunakan adalah SQLite database. Tahap keempat: pengujian, melakukan pengujian terhadap aplikasi yang telah dibuat dengan melakukan perbandingan dari segi waktu yang dibutuhkan setiap algoritma dalam pencarian *string*.

Perancangan sistem sebagai solusi dari permasalahan yang ada, dilakukan dengan menggunakan perangkat pemodelan *Unified Modeling Language* (UML). Diagram yang dibuat adalah *use case diagram*, *activity diagram* dan *class diagram*. *Use case diagram* menampilkan interaksi antara *user* dengan sistem.



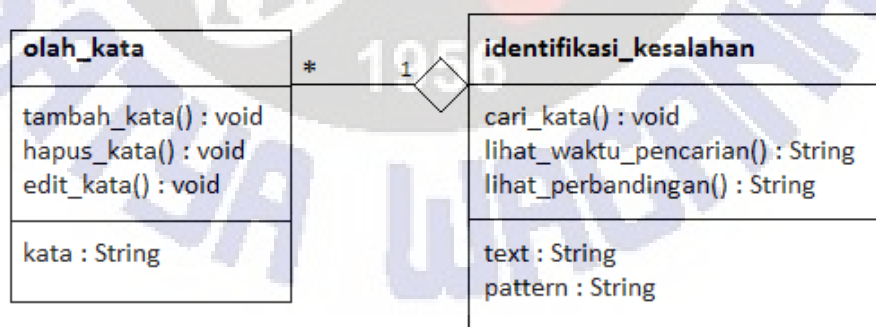
Gambar 5 Use case diagram identifikasi kesalahan penulisan teks.

Sistem terdiri dari satu *actor* yaitu *user*. *Use case* *olah_kata* berfungsi untuk menyimpan kata yang dijadikan acuan untuk melakukan identifikasi. Sedangkan, *use case* *identifikasi_kesalahan* berfungsi untuk mendeteksi kesalahan penulisan pada teks dan menampilkan hasilnya.



Gambar 6 Activity diagram identifikasi kesalahan penulisan teks.

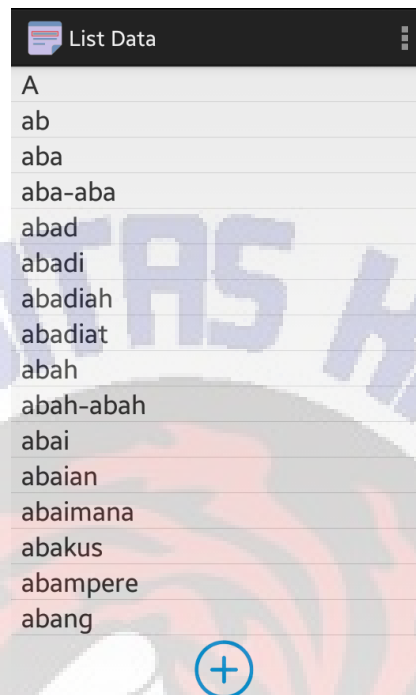
Gambar 6 menunjukan *activity diagram* yang dilakukan *user* dalam melakukan proses identifikasi kesalahan penulisan teks. Alur kerjanya dimulai dari atas ke bawah.



Gambar 7 Class diagram identifikasi kesalahan penulisan teks.

Gambar 7 menunjukan hubungan antar *class-class* yang digunakan dalam sistem. *Class* *olah_kata* terhubung dengan *class* *identifikasi_kesalahan*. Hal ini berarti kata yang ada dalam *olah_kata* digunakan sebagai *text* untuk mengidentifikasi kesalahan penulisan.

4. Hasil dan Pembahasan



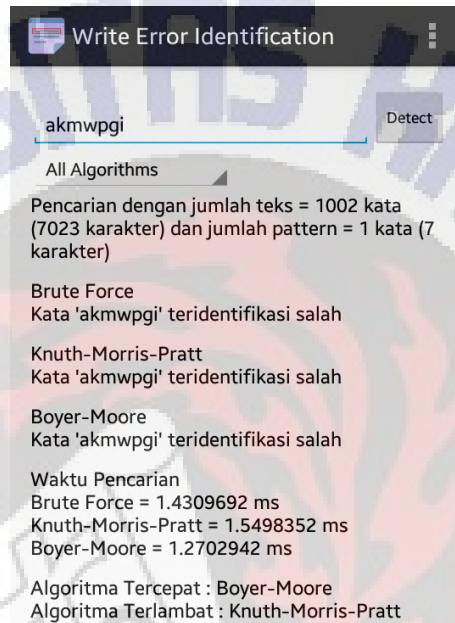
Gambar 8 Tampilan halaman olah kata.

Halaman olah kata menampilkan data yang digunakan sebagai *text* untuk identifikasi kesalahan penulisan. Data berupa kata yang diambil dari Kamus Besar Bahasa Indonesia. Pada halaman ini, data dapat ditambah, diubah, dihapus untuk menyesuaikan masukan dari *user*.



Gambar 9 Tampilan halaman identifikasi kesalahan untuk mendeteksi penulisan teks yang benar.

Gambar 9 menunjukkan penulisan teks yang benar. Inputan *user* digunakan sebagai *pattern* yang kemudian dibagi ke dalam *string array* berdasarkan spasi dan tanda baca. Kalimat “aku mau pergi” dibagi menjadi 3 *string* yaitu aku, mau dan pergi. Selanjutnya saat *button* detect berfungsi, setiap *string pattern* dicari di dalam *text* menggunakan algoritma Brute Force, Knuth-Morris-Pratt, Boyer-Moore. Selanjutnya setiap algoritma pencarian *string* menampilkan verifikasi dan waktu yang dibutuhkan dalam pencarian.



Gambar 10 Tampilan halaman identifikasi kesalahan untuk mendeteksi penulisan teks yang salah.

Gambar 10 menunjukkan penulisan teks yang salah. Kalimat “akmwpgi” dibaca sebagai 1 *string*, kemudian dicari di dalam *text* menggunakan algoritma Brute Force, Knuth-Morris-Pratt, Boyer-Moore. Selanjutnya apabila *pattern* tidak ditemukan di dalam *text*, maka setiap algoritma pencarian *string* menampilkan pemberitahuan kesalahan penulisan dan waktu yang dibutuhkan dalam pencarian. Setiap algoritma pencarian *string* diterapkan dalam kode program. Kode program adalah suatu rangkaian pernyataan atau deklarasi yang ditulis dalam bahasa pemrograman komputer yang terbaca manusia.

Kode Program 1 Metode pencarian *string* Brute Force.

```
1 String bruteForceSearch(String text, String[] pattern) {
2     String outputInfo = "Brute Force\nSemua kata teridentifikasi benar";
3     char[] T = text.toCharArray();
4     int n = T.length;
5     for (int a = 0; a < pattern.length; a++) {
6         char[] P = pattern[a].toCharArray();
7         int m = P.length;
8         int x, y;
9         for (x = 0, y = 0; x < n && y < m; x++) {
10             if (T[x] == P[y]) {
11                 y++;
12             } else {
13                 x -= y;
14                 y = 0;
15             }
16         }
17         if (y == m) {
18             if (a == pattern.length - 1)
19                 return outputInfo;
20             } else {
21                 outputInfo = "Brute Force\nKata " + pattern[a] + " teridentifikasi salah";
22                 return outputInfo;
23             }
24         }
25     return outputInfo;
26 }
```

Baris 1 menunjukkan tipe data metode, nama metode dan paramater yang digunakan dalam metode. Baris 2, deklarasi *string* outputInfo untuk verifikasi. Baris 3, deklarasi *char array* T untuk membagi karakter pada *text* dalam *array*. Baris 4, deklarasi *integer* n untuk mengukur panjang karakter dalam *text*. Pada baris 5 sampai 25, dilakukan perulangan sejumlah banyaknya *string pattern*. Baris 6, deklarasi *char array* P untuk membagi karakter pada *pattern* dalam *array*. Baris 7, deklarasi *integer* m untuk mengukur panjang karakter dalam *pattern*. Baris 8 deklarasi *integer* x untuk digunakan sebagai indeks pencarian *text* dan y untuk digunakan sebagai indeks pencarian *pattern*. Pada baris 9 sampai 16, dilakukan perulangan dengan kondisi x lebih kecil dari panjang karakter *text* dan y lebih kecil dari panjang karakter *pattern*. Baris 10, dilakukan pencocokan *text* dan *pattern*, apabila cocok (baris 11) maka indeks *pattern* bertambah 1 (pencocokan karakter berikutnya). Jika ditemukan ketidakcocokan (baris 12 sampai 15) maka indeks x sama dengan x dikurani y (kotak pencarian digeser ke kanan) dan y sama dengan 0 (indeks *pattern* dimulai dari awal). Baris 17, apabila indeks y sama dengan panjang *pattern* (seluruh karakter dalam *pattern* cocok) maka dilakukan pengecekan (baris 18). Kemudian, jika pencocokan sampai pada *string pattern* yang paling terakhir maka metode mengembalikan nilai *string* outputInfo (baris 19). Pada baris 20 sampai 23, apabila indeks y tidak sama dengan panjang *pattern* (*pattern* tidak terdapat dalam *text*), maka metode mengembalikan nilai *string* outputInfo yang diinisialisasi ulang untuk pemberitahuan kesalahan penulisan.

Kode Program 2 Metode pencarian *string* Knuth-Morris-Pratt.

```
1 String knuthMorrisPrattSearch(String text, String[] pattern) {
2     String outputInfo = "Knuth-Morris-Pratt\nSemua kata teridentifikasi benar";
3     char[] T = text.toCharArray();
4     int n = T.length;
5     for (int a = 0; a < pattern.length; a++) {
6         char[] P = pattern[a].toCharArray();
7         int m = P.length;
8         int R = 256;
9         int[][] dfa = new int[R][m];
10        dfa[P[0]][0] = 1;
11        for (int x = 0, y = 1; y < m; y++) {
12            for (int c = 0; c < R; c++)
13                dfa[c][y] = dfa[c][x];
14            dfa[P[y]][y] = y + 1;
15            x = dfa[P[y]][x];
16        }
17        int i, j;
18        for (i = 0, j = 0; i < n && j < m; i++) {
19            j = dfa[T[i]][j];
20        }
21        if (j == m) {
22            if (a == pattern.length - 1)
23                return outputInfo;
24        } else {
25            outputInfo = "Knuth-Morris-Pratt\nKata " + pattern[a] + " teridentifikasi salah";
26            return outputInfo;
27        }
28    }
29    return outputInfo;
30 }
```

Baris 1 menunjukkan tipe data metode, nama metode dan paramater yang digunakan dalam metode. Baris 2, deklarasi *string* outputInfo untuk verifikasi. Baris 3, deklarasi *char array* T untuk membagi karakter pada *text* dalam *array*. Baris 4, deklarasi *integer* n untuk mengukur panjang karakter dalam *text*. Pada baris 5 sampai 30, dilakukan perulangan sejumlah banyaknya *string pattern*. Baris 6, deklarasi *char array* P untuk membagi karakter pada *pattern* dalam *array*. Baris 7, deklarasi *integer* m untuk mengukur panjang karakter dalam *pattern*. Baris 8, deklarasi *integer* R sebagai *radix* senilai jumlah karakter yang terdidefinisikan dalam standar ASCII. Baris 9, deklarasi *integer array* 2 dimensi dfa (deterministic finite automaton) yang digunakan untuk menentukan pergeseran kotak pencarian. Baris 10, inisialisasi dfa senilai 1 (untuk pergeseran minimal). Pada baris 11 sampai 16 dengan kondisi y lebih kecil dari panjang *pattern*. Pada baris 12 dilakukan perulangan untuk baris 13 dengan keadaan c lebih kecil dari *radix*. Baris 13, dilakukan inisialisasi dfa pada indeks c dan indeks y senilai dfa pada indeks c dan indeks x. Baris 14, dilakukan inisialisasi dfa pada indeks *pattern* indeks y dan indeks y senilai dengan y ditambah 1. Baris 15, dilakukan inisialisasi nilai x senilai dfa pada indeks *pattern* indeks y dan indeks x. Baris 17, deklarasi *integer* i dan j untuk menentukan indeks dfa dalam perulangan. Pada

baris 11 sampai 16, dilakukan perulangan dengan kondisi i lebih kecil dari panjang karakter *text* dan j lebih kecil dari panjang karakter *pattern*. Baris 19, dilakukan inisialisasi j senilai dfa pada indeks *text* indeks i dan indeks j (menentukan panjang pergeseran). Baris 21, apabila indeks j sama dengan panjang *pattern* (seluruh karakter dalam *pattern* cocok) maka dilakukan pengecekan (baris 22). Kemudian, jika pencocokan sampai pada *string pattern* yang paling terakhir maka metode mengembalikan nilai *string* outputInfo (baris 23). Pada baris 24 sampai 27, apabila indeks y tidak sama dengan panjang *pattern* (*pattern* tidak terdapat dalam *text*), maka metode mengembalikan nilai *string* outputInfo yang diinisialisasi ulang untuk pemberitahuan kesalahan penulisan (baris 26).

Kode Program 3 Metode pencarian *string* Boyer-Moore.

```

1 String boyerMooreSearch(String text, String[] pattern) {
2     String outputInfo = "Boyer-Moore\nSemua kata teridentifikasi benar";
3     char[] T = text.toLowerCase().toCharArray();
4     int n = T.length;
5     boolean wordFound;
6     for (int a = 0; a < pattern.length; a++) {
7         wordFound = false;
8         char[] P = pattern[a].toLowerCase().toCharArray();
9         int m = P.length;
10        int Radix = 256;
11        int[] right = new int[Radix];
12        for (int i = 0; i < Radix; i++) {
13            right[i] = -1;
14        }
15        for (int j = 0; j < m; j++) {
16            right[P[j]] = j;
17        }
18        int skip;
19        for (int x = 0; x <= n - m; x += skip) {
20            skip = 0;
21            for (int y = m - 1; y >= 0; y--) {
22                if (P[y] != T[x + y]) {
23                    skip = Math.max(1, y - right[T[x + y]]);
24                    break;
25                }
26            }
27            if (skip == 0) {
28                wordFound = true;
29                if (a == pattern.length - 1)
30                    return outputInfo;
31                else
32                    break;
33            }
34        }
35        if (wordFound == false) {
36            outputInfo = "Boyer-Moore\nKata '" + pattern[a] + "' teridentifikasi salah";
37            return outputInfo;
38        }
39    }
40    return outputInfo;
41 }

```

Baris 1 menunjukan tipe data metode, nama metode dan paramater yang digunakan dalam metode. Baris 2, deklarasi *string* outputInfo untuk verifikasi. Baris 3, deklarasi *char array* T untuk membagi karakter pada *text* dalam *array*. Baris 4, deklarasi *integer* n untuk mengukur panjang karakter dalam *text*. Baris 5,

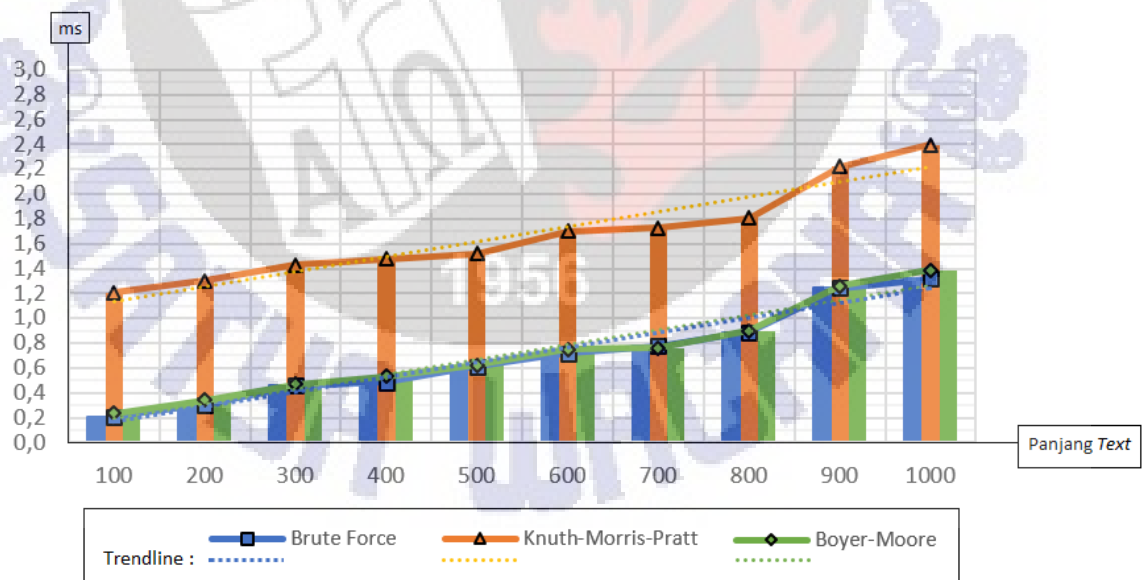
deklarasi *Boolean* wordFound untuk proteksi *pattern* yang di cek. Pada baris ke 6 sampai ke 39 sejumlah banyaknya *string pattern*. Baris 7, inisialisasi *boolean* wordFound menjadi *false*. Baris 8, deklarasi *char array* P untuk membagi karakter pada *pattern* dalam *array*. Baris 9, deklarasi *integer* m untuk mengukur panjang karakter dalam *pattern*. Baris 10, deklarasi *integer* Radix senilai jumlah karakter yang terdefiniskan dalam standar ASCII. Baris 11, deklarasi *integer array* right sepanjang nilai Radix. Pada baris 12 sampai 17 adalah proses menentukan nilai *array* right yang digunakan untuk menentukan pergeseran pencarian. Baris 18, inisialisasi *integer* skip untuk mengeser kotak pencarian. Pada baris 19 sampai 34, dilakukan perulangan dengan kondisi x lebih kecil sama dengan nilai panjang *text* dikurang panjang *pattern*. Baris 20, inisialisasi *integer* skip sama dengan 0. Pada baris ke 21 sampai 26 dilakukan perulangan dengan kondisi nilai y lebih besar sama dengan 0. Pada baris ke 22 dilakukan pengecekan apabila terjadi ketidakcocokan antara karakter *pattern* pada indeks y dan *text* pada indeks x ditambah y, kemudian *integer* skip diinisialisasi dengan nilai terbesar diantara 1 dan nilai y dikurang right pada indeks *text* pada indeks x ditambah y (baris 23), selanjutnya dilakukan break untuk menghentikan perulangan (baris 24). Baris 27, dilakukan pengecekan apabila nilai skip sama dengan 0 (*pattern* ditemukan dalam *text*, tidak ada karakter lagi yang dapat digeser), inisialisasi *boolean* wordFound menjadi *true* (baris 28). Kemudian pada baris 29, jika pencocokan merupakan *string pattern* yang paling terakhir (baris 30) maka metode mengembalikan nilai *string* outputInfo, jika pencocokan *string* belum yang terakhir maka dilakukan break untuk menghentikan perulangan. Pada baris 35 sampai 38 dilakukan pengecekan apabila *boolean* wordFound sama dengan *false* (*pattern* tidak ditemukan dalam *text*), selanjutnya metode mengembalikan nilai *string* outputInfo yang diinisialisasi ulang untuk pemberitahuan kesalahan penulisan.

Pengujian dilakukan berdasarkan waktu eksekusi algoritma pada proses *runtime*. Proses *runtime* dipengaruhi oleh kode program, *compiler*, arsitektur mesin dan sistem operasi. Sehingga waktu eksekusi yang dicatat berdasarkan rata-rata dari 100 eksekusi yang dilakukan. Perangkat pengujian yang digunakan adalah *smartphone* Samsung Galaxy Grand Prime dengan CPU quadcore 1.2 GHz Cortex-A53, RAM 1 GB dan menggunakan sistem operasi Android OS Lollipop versi 5.1. Pengujian menggunakan kata dasar (lema) yang terdapat dalam Kamus Besar Bahasa Indonesia sebagai *text*.

Tabel 1 Pengujian berdasarkan panjang *text* menggunakan 5 kata sebagai *pattern*.

No	Panjang Text (kata)	Waktu eksekusi algoritma (millisecond)		
		Brute Force	Knuth-Morris-Pratt	Boyer-Moore
1	100 (625 karakter)	0,208	1,209	0,234
2	200 (1299 karakter)	0,303	1,301	0,340
3	300 (2020 karakter)	0,459	1,432	0,469
4	400 (2654 karakter)	0,486	1,479	0,530
5	500 (3414 karakter)	0,610	1,522	0,624
6	600 (4122 karakter)	0,721	1,702	0,748
7	700 (4847 karakter)	0,775	1,726	0,760
8	800 (5536 karakter)	0,889	1,805	0,901
9	900 (6229 karakter)	1,244	2,222	1,257
10	1000 (7015 karakter)	1,320	2,395	1,386

Tabel 1 menunjukan pengujian pertama yang dilakukan berdasarkan panjang *text*, menggunakan 5 kata (26 karakter) sebagai *pattern* yang teridentifikasi benar dan panjang *text* 100 sampai 1000 kata. *Index* posisi *pattern* dalam *text* letaknya urut dari kecil ke besar. *Pattern* yang digunakan adalah abadi abang abdi abiotik abjad. *Index* posisi penambahan *text* letaknya urut dari kecil ke besar. Rata-rata waktu pencarian algoritma Brute Force adalah 0.70 ms, algoritma Knuth-Morris-Pratt adalah 1.68 ms, algoritma Boyer-Moore adalah 0,72 ms.



Gambar 11 Grafik pengujian pertama.

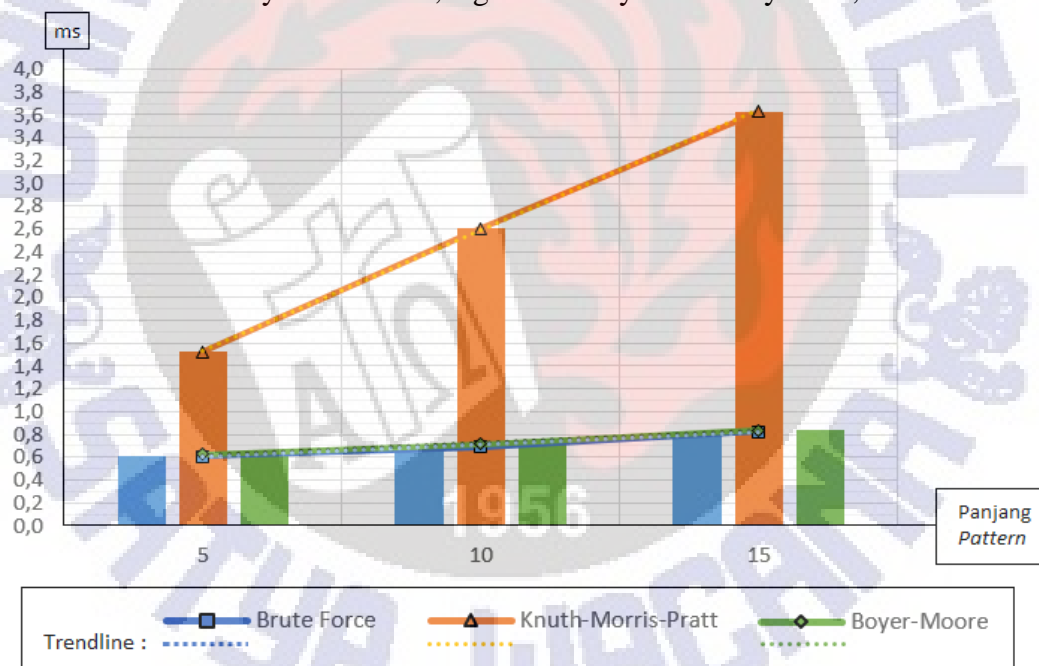
Grafik pengujian pertama menunjukan penambahan panjang *text* membuat waktu yang dibutuhkan algoritma Brute Force, Knuth-Morris-Pratt, Boyer-Moore menjadi semakin lama. Rata-rata waktu yang dibutuhkan dalam pencarian *pattern* yang teridentifikasi benar, paling cepat algoritma Brute Force dibandingkan

algoritma Boyer-Moore dan Knuth-Morris-Pratt, dan paling lambat algoritma Knuth-Morris-Pratt dibandingkan algoritma Brute Force dan Boyer-Moore.

Tabel 2 Pengujian berdasarkan panjang *pattern* menggunakan 500 kata sebagai *text*.

No	Panjang Pattern (kata)	Waktu eksekusi algoritma (millisecond)		
		Brute Force	Knuth-Morris-Pratt	Boyer-Moore
1	5 (26 karakter)	0,610	1,522	0,624
2	10 (59 karakter)	0,689	2,606	0,716
3	15 (85 karakter)	0,825	3,630	0,834

Tabel 2 menunjukkan pengujian kedua yang dilakukan berdasarkan panjang *pattern*, menggunakan 5 sampai 15 kata sebagai *pattern* yang teridentifikasi benar dan 500 kata (3414 karakter) sebagai *text*. *Index* posisi *pattern* dalam *text* letaknya urut kecil ke besar. *Index* posisi penambahan *pattern* letaknya urut dari kecil ke besar. Rata-rata waktu pencarian algoritma Brute Force yaitu 0.71 ms, algoritma Knuth-Morris-Pratt yaitu 2.59 ms, algoritma Boyer-Moore yaitu 0,72 ms.



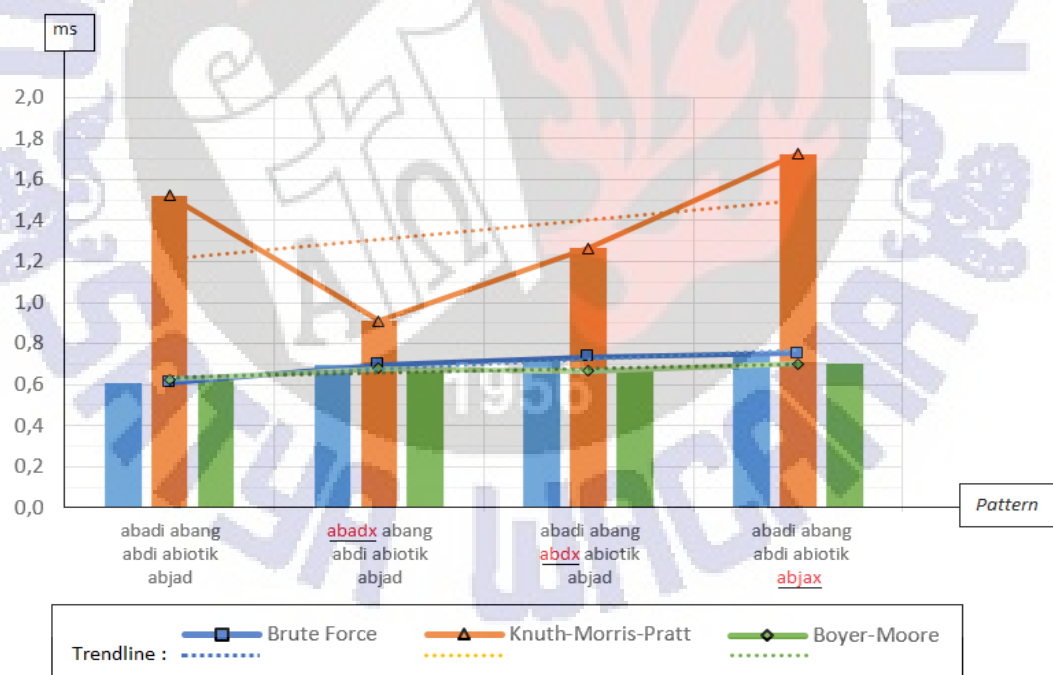
Gambar 12 Grafik pengujian kedua.

Grafik pengujian kedua menunjukkan penambahan panjang *pattern* membuat waktu yang dibutuhkan algoritma Brute Force, Knuth-Morris-Pratt, Boyer-Moore menjadi semakin lama. Rata-rata waktu yang dibutuhkan dalam pencarian *pattern* yang teridentifikasi benar, paling cepat algoritma Brute Force dibandingkan algoritma Boyer-Moore dan Knuth-Morris-Pratt, dan paling lambat algoritma Knuth-Morris-Pratt dibandingkan algoritma Brute Force dan Boyer-Moore.

Tabel 3 Pengujian berdasarkan kesalahan penulisan *pattern* menggunakan 500 kata sebagai *text*.

No	Pattern	Waktu eksekusi algoritma (millisecond)		
		Brute Force	Knuth-Morris-Pratt	Boyer-Moore
1	abadi abang abdi abiotik abjad	0,610	1,522	0,624
2	<u>abadx</u> abang abdi abiotik abjad	0,699	0,911	0,678
3	abadi abang <u>abdx</u> abiotik abjad	0,736	1,266	0,667
4	abadi abang abdi abiotik <u>abjax</u>	0,753	1,723	0,701

Tabel 3 menunjukkan pengujian ketiga yang dilakukan berdasarkan kesalahan penulisan *pattern*, menggunakan 5 *pattern* yang berbeda dan 500 kata (3414 karakter) sebagai *text*. Panjang *pattern*-*pattern* pada Tabel 3 sama yaitu 5 kata (26 karakter). Perbedaan 4 *pattern* pada Tabel 3 yaitu pada *pattern* pertama *pattern* teridentifikasi benar, pada *pattern* kedua *pattern* teridentifikasi salah dengan kesalahan pada kata pertama (awal), pada *pattern* ketiga *pattern* teridentifikasi salah dengan kesalahan pada kata ketiga (tengah), pada *pattern* keempat *pattern* teridentifikasi salah dengan kesalahan pada kata kelima (akhir). Kesalahan penulisan pada *pattern* ditandai dengan garis bawah. *Index* posisi kata dalam *text* letaknya urut dari kecil ke besar.



Gambar 13 Grafik pengujian ketiga.

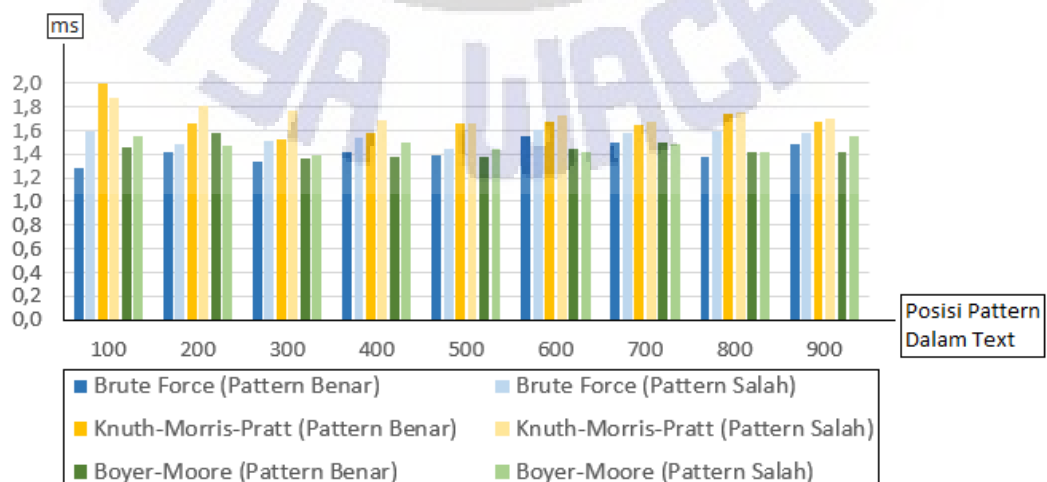
Grafik pengujian ketiga menunjukkan penambahan *index* posisi kesalahan *pattern* membuat waktu yang dibutuhkan algoritma Brute Force, Knuth-Morris-Pratt, Boyer-Moore menjadi semakin lama. Rata-rata waktu yang dibutuhkan dalam pencarian *pattern* yang teridentifikasi benar, paling cepat algoritma Brute Force dibandingkan algoritma Boyer-Moore dan Knuth-Morris-Pratt, dan paling

lambat algoritma Knuth-Morris-Pratt dibandingkan algoritma Brute Force dan Boyer-Moore. Sedangkan, rata-rata waktu yang dibutuhkan dalam pencarian *pattern* teridentifikasi salah, paling cepat algoritma Boyer-Moore dibandingkan algoritma Brute Force dan Knuth-Morris-Pratt, dan paling lambat algoritma Knuth-Morris-Pratt dibandingkan algoritma Boyer-Moore dan Brute Force.

Tabel 4 Pengujian berdasarkan posisi 1 kata *pattern* di dalam 1000 kata *text*.

No	Posisi Pattern Dalam Text (kata)	Waktu eksekusi algoritma (millisecond)					
		Brute Force		Knuth-Morris-Pratt		Boyer-Moore	
		Pattern Benar	Pattern Salah	Pattern Benar	Pattern Salah	Pattern Benar	Pattern Salah
1	100 (620-625 karakter)	1,285	1,596	1,998	1,874	1,460	1,555
2	200 (1294-1299 karakter)	1,418	1,488	1,661	1,812	1,573	1,468
3	300 (2015-2020 karakter)	1,331	1,506	1,525	1,772	1,363	1,390
4	400 (2649-2654 karakter)	1,412	1,537	1,571	1,688	1,372	1,502
5	500 (3409-2414 karakter)	1,393	1,446	1,653	1,662	1,381	1,447
6	600 (4117-4122 karakter)	1,557	1,602	1,674	1,732	1,439	1,421
7	700 (4842-4847 karakter)	1,493	1,577	1,645	1,671	1,502	1,485
8	800 (5531-5536 karakter)	1,373	1,597	1,739	1,752	1,417	1,422
9	900 (6224-6229 karakter)	1,477	1,571	1,669	1,693	1,416	1,553

Tabel 4 menunjukkan pengujian keempat yang dilakukan berdasarkan posisi *pattern* dalam *text*, dilakukan menggunakan 1 kata (5 karakter) yang berbeda sebagai *pattern* dan 1000 kata (7015 karakter) sebagai *text*. Posisi kesalahan pada *pattern* yang teridentifikasi salah sama pada huruf (karakter) terakhir setiap kata. *Index* posisi kata dalam *text* letaknya urut dari kecil ke besar. Rata-rata waktu pencarian *pattern* yang teridentifikasi benar menggunakan algoritma Brute Force adalah 1.42 ms, algoritma Knuth-Morris-Pratt adalah 1.68 ms, algoritma Boyer-Moore adalah 1.44 ms. Sedangkan, rata-rata waktu pencarian *pattern* yang teridentifikasi salah menggunakan algoritma Brute Force adalah 1.55 ms, algoritma Knuth-Morris-Pratt adalah 1.74 ms, algoritma Boyer-Moore adalah 1.47 ms.



Gambar 14 Grafik pengujian keempat.

Grafik pengujian keempat menunjukkan perubahan *index* posisi *pattern* dalam *text* tidak mempengaruhi waktu yang dibutuhkan algoritma Brute Force, Knuth-Morris-Pratt dan Boyer-Moore. Hal ini disebabkan pencocokan *pattern* dalam *text* dimulai dari karakter awal sampai akhir. Rata-rata waktu yang dibutuhkan dalam pencarian *pattern* yang teridentifikasi benar, paling cepat algoritma Brute Force dibandingkan algoritma Boyer-Moore dan Knuth-Morris-Pratt, dan paling lambat algoritma Knuth-Morris-Pratt dibandingkan algoritma Brute Force dan Boyer-Moore. Sedangkan, rata-rata waktu yang dibutuhkan dalam pencarian *pattern* yang teridentifikasi salah, paling cepat algoritma Boyer-Moore dibandingkan algoritma Brute Force dan Knuth-Morris-Pratt, dan paling lambat algoritma Knuth-Morris-Pratt dibandingkan algoritma Boyer-Moore dan Brute Force.

5. Simpulan

Dalam penelitian ini dibuat sebuah aplikasi perbandingan algoritma Brute Force, algoritma Knuth-Morris-Pratt, algoritma Boyer-Moore untuk kesalahan penulisan teks berbasis Android. Berdasarkan hasil dari 4 pengujian yang dilakukan, penambahan waktu yang dibutuhkan algoritma Brute Force, Knuth-Morris-Pratt dan Boyer-Moore dipengaruhi oleh panjang *text*, panjang *pattern*, posisi kesalahan pada *pattern*. Posisi *pattern* dalam *text* tidak mempengaruhi waktu yang dibutuhkan algoritma Brute Force, Knuth-Morris-Pratt dan Boyer-Moore. Algoritma yang rentan karena perubahan panjang *pattern* dan perubahan posisi kesalahan penulisan *pattern* adalah Knuth-Morris-Pratt, sedangkan algoritma Brute Force dan Boyer-Moore tidak banyak berubah. Rata-rata waktu pencarian untuk *pattern* teridentifikasi benar dilihat dari perubahan panjang *text*, perubahan panjang *pattern* dan perubahan posisi kesalahan pada *pattern*, paling cepat algoritma Brute Force dibandingkan algoritma Boyer-Moore dan Knuth-Morris-Pratt, dan paling lambat algoritma Knuth-Morris-Pratt dibandingkan algoritma Boyer-Moore dan Brute Force.

Saran untuk penelitian lebih lanjut adalah membandingkan algoritma pencarian *string* dari arah yang ditentukan algoritma tersebut (seperti algoritma Colussi, algoritma Crochemore-Perrin) dalam aplikasi berbasis Android.

6. Daftar Pustaka

- [1] Adriyani, Ni Made Muni. 2012. "Implementasi Algoritma Levenshtein Distance dan Metode Empiris untuk Menampilkan Saran Perbaikan Kesalahan Pengetikan Dokumen Berbahasa Indonesia". Universitas Udayang.
- [2] Rochmawati, Yeny. 2015. "Studi Perbandingan Algoritma Pencarian String dalam Metode Approximate String Matching untuk Identifikasi Kesalahan Pengetikan Teks". Universitas Diponegoro.
- [3] Sulun, Hafni Syaeful. 2007. "Penerapan Algoritma Knuth-Morris-Pratt pada Aplikasi Pencarian Berkas di Komputer". Institut Teknologi Bandung.
- [4] Sagita, Vina. 2013. "Studi Perbandingan Implementasi Algoritma Boyer-Moore, Turbo Boyer-Moore, dan Tuned Boyer-Moore dalam Pencarian String". Universitas Media Nusantara.
- [5] International Data Corporation: Smartphone OS Market Share, 2015 Q2. www.idc.com/prodserv/smartphone-os-market-share.jsp diakses pada 10 Maret 2016.
- [6] Utomo, Darmawan. 2008. "Perbandingan String Searching Brute Force, Knuth-Morris-Pratt, Boyer-Moore dan Karp Rabin pada Teks Alkitab Bahasa Indonesia". Universitas Kristen Satya Wacana.
- [7] Charras, Christian and Thierry Lecroq. 2004. "Handbook of Exact String Matching Algorithms". ISBN 0-9543006-4-5.
- [8] Sedgewick, Robert and Kevin Wayne. 2011. "Algorithms, 4th Edition". ISBN-13 978-0-321-57351-3.
- [9] Tambun, Evelyn Dwi. 2011. "Perbandingan Penggunaan Algoritma BM dan Algoritma Horspool pada Pencarian String dalam Bahasa Medis". Institut Teknologi Bandung.
- [10] Android: The Android Source Code. source.android.com/source/index.html diakses pada 20 Maret 2016.
- [11] Pressman, Roger S. 2000. "Software Engineering: a Practitioner's Approach". ISBN 0-07-365578-3.
- [12] Pusat Bahasa Departemen Pendidikan Nasional. 2008. "Kamus Bahasa Indonesia". ISBN 978-979-689-779-1.