STUDI PERBANDINGAN DAN IMPLEMENTASI ALGORITMA *AHO-CORASICK STRING MATCHING* DENGAN ALGORITMA *INTERPOLATION SEARCH* PADA APLIKASI KAMUS KEDOKTERAN BERBASIS *MOBILE*

SKRIPSI

Oleh:

VILLA NANDA SAHARA

NIM: 09650172

JURUSAN TEKNIK INFORMATIKA

FAKULTAS SAINS DAN TEKNOLOGI

UNIVERSITAS ISLAM NEGERI (UIN) MAULANA MALIK IBRAHIM

MALANG

2014

HALAMAN PENGAJUAN

STUDI PERBANDINGAN DAN IMPLEMENTASI ALGORITMA *AHO-CORASICK STRING MATCHING* DENGAN ALGORITMA *INTERPOLATION SEARCH* PADA APLIKASI KAMUS KEDOKTERAN BERBASIS *MOBILE*

SKRIPSI

Diajukan kepada:

Fakultas Sains dan Teknologi

Universitas Islam Negeri Maulana Malik Ibrahim Malang

Untuk Memenuhi Salah Satu Persyaratan Dalam

Memperoleh Gelar Sarjana Komputer (S.Kom)

Oleh:

VILLA NANDA SAHARA

NIM: 09650172

JURUSAN TEKNIK INFORMATIKA

FAKULTAS SAINS DAN TEKNOLOGI

UNIVERSITAS ISLAM NEGERI (UIN) MAULANA MALIK IBRAHIM

MALANG

2014

HALAMAN PERSETUJUAN

STUDI PERBANDINGAN DAN IMPLEMENTASI ALGORITMA *AHO-CORASICK STRING MATCHING* DENGAN ALGORITMA *INTERPOLATION SEARCH* PADA APLIKASI KAMUS KEDOKTERAN BERBASIS *MOBILE*

SKRIPSI

Oleh :

Nama	: Villa Nanda Sahara
NIM	: 09650172
Jurusan	: Teknik Informatika
Fakultas	: Sains Dan Teknologi

Telah Disetujui, 04 Juli 2014

Dosen Pembimbing I	Dosen Pembimbing II
Irwan Budi Santoso, MT	**Zainal Abidin, M.Kom**
NIP. 19770103 201101 1 004	**NIP. 19760613 200501 1 004**

Mengetahui,

Ketua Jurusan Teknik Informatika

Dr. Cahyo Crysdian, M.Cs
NIP. 19740424 200901 1 008

HALAMAN PENGESAHAN

STUDI PERBANDINGAN DAN IMPLEMENTASI ALGORITMA *AHO-CORASICK STRING MATCHING* DENGAN ALGORITMA *INTERPOLATION SEARCH* PADA APLIKASI KAMUS KEDOKTERAN BERBASIS *MOBILE*

SKRIPSI

Oleh :

Villa Nanda Sahara
NIM. 09650172

Telah Dipertahankan Di Depan Dewan Penguji Skripsi
Dan Dinyatakan Diterima Sebagai Salah Satu Persyaratan
Untuk Memperoleh Gelar Sarjana Komputer (S.Kom)

Tanggal 11 Juli 2014

Susunan Dewan Penguji:		Tanda Tangan
1. Penguji Utama	: A'la Syauqi, M. Kom NIP. 19771201 200801 1 007	( )
2. Ketua Penguji	: Dr. Cahyo Crysdian, M.Cs NIP. 19740424 200901 1 008	( )
3. Sekretaris	: Irwan Budi Santoso, MT NIP. 19770103 201101 1 004	( )
4. Anggota Penguji	: Zainal Abidin, M. Kom NIP. 19760613 200501 1 004	( )

Mengetahui,
Ketua Jurusan Teknik Informatika

Dr. Cahyo Crysdian, M.Cs
NIP. 19740424 200901 1 008

HALAMAN PERNYATAAN

ORISINALITAS PENELITIAN

Saya yang bertandatangan di bawah ini:

Nama	:	Villa Nanda Sahara
NIM	:	09650172
Fakultas/Jurusan	:	Sains Dan Teknologi / Teknik Informatika
Judul Penelitian	:	Studi Perbandingan dan Implementasi Algoritma *Aho-Corasick String Matching* Dengan Algoritma *Interpolation Search* Pada Aplikasi Kamus Kedokteran Berbasis *Mobile*

Menyatakan dengan sebenarnya bahwa skripsi yang saya tulis ini benar-benar merupakan hasil karya saya sendiri, bukan merupakan pengambil alihan data, tulisan atau pikiran oarang lain yang saya akui sebagai hasil tulisan atau pikiran saya sendiri, kecuali dengan mencantumkan sumber cuplikan pada daftar pustaka. Apabila di kemudian hari terbukti atau dapat dibuktikan skripsi ini hasil jiplakan, maka saya bersedia menerima sanksi atas perbuatan tersebut.

Malang, 07 Juli 2014

Yang Membuat Pernyataan,

Villa Nanda Sahara

NIM. 09650172

MOTTO

لَا يُكَلِّفُ ٱللَّهُ نَفۡسًا إِلَّا وُسۡعَهَاۚ بَلَىٰٓۚ إِن تَصۡبِرُواْ وَتَتَّقُواْ

وَيَأۡتُوكُم مِّن فَوۡرِهِمۡ هَٰذَا يُمۡدِدۡكُمۡ رَبُّكُم بِخَمۡسَةِ ءَالَٰفٖ مِّنَ

ٱلۡمَلَٰٓئِكَةِ مُسَوِّمِينَ ١٢٥

"Ya (cukup), jika kamu bersabar dan bersiap-siaga, dan mereka datang menyerang kamu dengan seketika itu juga, niscaya Allah menolong kamu dengan lima ribu malaikat yang memakai tanda". *(QS: Al-Imran: 125)*

Halaman Persembahan

Tuhan Yang Maha Kuasa, *beserta* ***RasulNya*** *yang terus tampa hentinya memberikan rahmat kepada hambanya*

Sebagai tanda bakti, hormat, ucapan terimakasih atas segala dukungan, senantiasa menguatkan, doa yang tiada henti, dan kasih sayang yang tiada terhingga.

Kupersembahkan karya tulis sederhana ini untuk segenap keluargaku dan keluarga besarku yang ada di rumah. Ibu ***Mutmainnah*** *yang telah merawat, merawat, mendidikku hingga mencapai saat ini, Almarhum ayah* ***Takrib Muchaedor*** *yang selalu membuatku tetap semangat ketika mengingat semangatmu dalam mendukungku. Tidak lupa juga kakakku* ***Vaesol Wahyu Eka Irawan*** *yang selalu mendukungku, dan saudara kembarku* ***Velly Nindi Tursinei,*** *dan yang paling cantik sekeluarga* ***Wafiq Failila Azizah,*** *semoga kelak cita-citamu tercapai. Terima kasih atas segala doa, semangat, dan dukungan kalian yang selama ini mengiringi hari-hariku*

Terima kasih kepada dosen-dosen dan para staf Teknik Informatika khususnya pembimbing skripsiku Bpk. Irwan Budi Santoso dan Bpk. Zainal Abidin, yang secara langsung maupun tidak langsung telah membantu dalam proses pembuatan skripsi ini.

Terima kasih juga untuk para sahabat teknik informatika UIN '09 yang mendukungku dalam proses pembuatan skripsi ini.

KATA PENGANTAR

بسم الله الرحمن الرحيم

Assalamu'alaikum Wr. Wb.

Puji syukur kepada Allah SWT yang telah memberikan kesehatan, kekuatan, segala nikmat dan kemudahan kepada penulis sehingga dapat menyelesaikan studi di Jurusan Teknik Informatika, Fakultas Sains dan Teknologi Universitas Islam Negeri Maulana Malik Ibrahim Malang sekaligus dapat menyelesaikan skripsi dengan judul "*Studi Perbandingan Dan Implementasi Algoritma Aho-Corasick String Matching Dengan Algoritma Interpolation Search Pada Aplikasi Kamus Kedokteran Berbasis Mobile*"

Sholawat serta salam selalu tercurahkan kepada Rasulullah Muhammad SAW yang telah membimbing umatnya dari zaman jahiliyah menuju zaman yang terang dan menjadikan saya bangga sebagai salah satu umatnya.

Penulis menyadari keterbatasan pengetahuan yang dimiliki, Oleh karena itu tanpa adanya keterlibatan dari berbagai pihak, skripsi ini akan sulit untuk dapat diselesaikan oleh penulis, Maka dari itu dengan segenap kerendahan hati penhulus mengucapkan terimakasih kepada :

1. Prof. Dr. H. Mudjia Rahardjo, selaku rektor UIN Maulana Malik Ibrahim Malang.
2. Dr. Drh. Hj Bayyinatul Muchtaromah, M.Si selaku Dekan Fakultas Sains dan Teknologi Universitas Islam Negeri Maulana Malik Ibrahim Malang.

3. M. Irwan Budi Santoso, M. T dan Zainal Abidin, M. Kom. selaku dosen pembimbing I dan II yang telah banyak memberikan bimbingan skripsi ini.
4. Dr. Cahyo Crysdian selaku Ketua Jurusan Teknik Informatika UIN Malanag
5. Segenap civitas akademika Jurusan Teknik Informatika, seluruh dosen dan karyawan , terimakasih atas segenap ilmu dan bimbingannya.
6. Ibu dan kakak serta adik-adik ku yang senantiasa memberikan doa dan dukungan kepada penulis dalam menuntut ilmu.
7. Semua teman-teman seperjuangan Teknik Informatika UIN Malang angkatan '09, sukses selalu buat kalian.
8. Semua pihak yang telah memberikan kontribusi tenaga, pikiran, motivasi kepada penulis selama pengerjaan skripsi ini.

Semoga skripsi ini dapat bermanfaat dan menambah khasanah ilmu pengetahuan.

Wassalamualaikum. Wr. Wb

Malang, 30 Juni 2014

Penulis

DAFTAR ISI

DAFTAR GAMBAR

DAFTAR TABEL

ABSTRAK

Sahara, Villa Nanda. 2014. **Studi Perbandingan dan Implementasi Algoritma *Aho Corasick String Matching* dengan *Algoritma Interpolation Search* Pada Aplikasi Kamus Kedokteran Berbasis *Mobile*.** Skripsi. Jurusan Teknik Informatika Fakultas Sains dan Teknologi Universitas Islam Negeri Maulana Malik Ibrahim Malang.

Pembimbing: (I) Irwan Budi Santoso, MT (II) Zainal Abidin, M. Kom

Kata Kunci : *Pencarian, Kamus Kedokteran, Pattern matching, Algoritma Aho Corasick String Matching, Algoritma Interpolation Search*

Algoritma pencarian merupakan salah satu dari banyak algoritma yang memiliki fungsi dalam kehidupan sehari-hari, penggunaan algoritma ini sangat dibutuhkan jika berkaitan dengan proses pencarian. Algoritma *aho corasick string matching* merupakan salah satu algoritma pencarian yang tergolong pencocokan kata atau *pattern matching*, kemudian algoritma *interpolation search* yang merupakan perkembangan dari algoritma *binary search, sequential search.* Dua algoritma tersebut tergolong algoritma pencarian meskipun beda dalam proses pencarian. Tujuan penelitian ini adalah membandingkan antara algoritma *aho corasick string matching* dengan algoritma *interpolation search*, meskipun berbeda prosesnya tapi masih tergolong metode pencarian. Hal ini yang mendasari dibuatnya penelitian ini. Pengujian dari perbandingan kedua algoritma tersebut diterapkan melalui proses pencarian kata istilah kedokteran yang diambil dari kamus kedokteran *Webster's New World*. Hasil pengujian dilakukan untuk menentukan konsumsi waktu yang diperlukan oleh masing-masing algoritma, ketika proses pencarian sedang berlangsung. Uji coba akan dilakukan beberapa kali untuk mendapatkan rata-rata waktu konsumsi yang diperlukan. Hasil dari uji coba menunjukkan algoritma *aho corasick string matching* memiliki konsumsi waktu yang lebih sedikit daripada algoritma *interpolation search.* Pengujian aplikasi kamus ini dilakukan pada perangkat *mobile* yang menggunakan *platform* Android.

ABSTRACT

Sahara, Villa Nanada. 2014. **Comparison and Implementation of *Aho Corasick String Matching* Algorithm Between *Algoritma Interpolation Search* Algorithm on Medical Dictionary Application Based on Mobile**. Thesis. Information Techniques Department. Fakulty of Science dan Technology. The State Islamic University Maulana Malik Ibrahim Malang.

Advisor: (I) Irwan Budi Santoso, MT (II) Zainal Abidin, M.Kom

Key terms: Search, Medical Dictionary, Pattern matching, Algorithm *Corasick String Matching*, *Algoritma Interpolation Search*

Searching algorithm is one of algorithms which has a function in dialy, the algorithm usage is needed if it is related to searching process. *Aho corasick string matching* algorithm is one of searching algorithm pertained adaptation words or pattern matching, afterwards *interpolation search* algorithm is the development from *binary search*, *sequential search* algorithm. Both of algorithm is pertained searching algorithm eventhough there is difference on searching process. The purpose of this research is compare between *aho corasick string matching* algorithm between *interpolation search* algorithm eventhough there is difference in process, but it is pertained in searching methode. That is the reason that the researcher manufactures this research. The testing from comparing both algorithm are applied by searching pocess of medical terms taken from dictionary medical *Webster's New World*. The result of testing is done for determine the time needed by each algorithm when the searching process is ongoing. The examination is cundocted several times for get mean of time needed. The reseult of examination shows that *aho corasick string matching* algorithm has time fewer than *interpolation search* algorithm. The examination of dictionary application is conducted on mobile peripheral used android platform.

BAB I

PENDAHULUAN

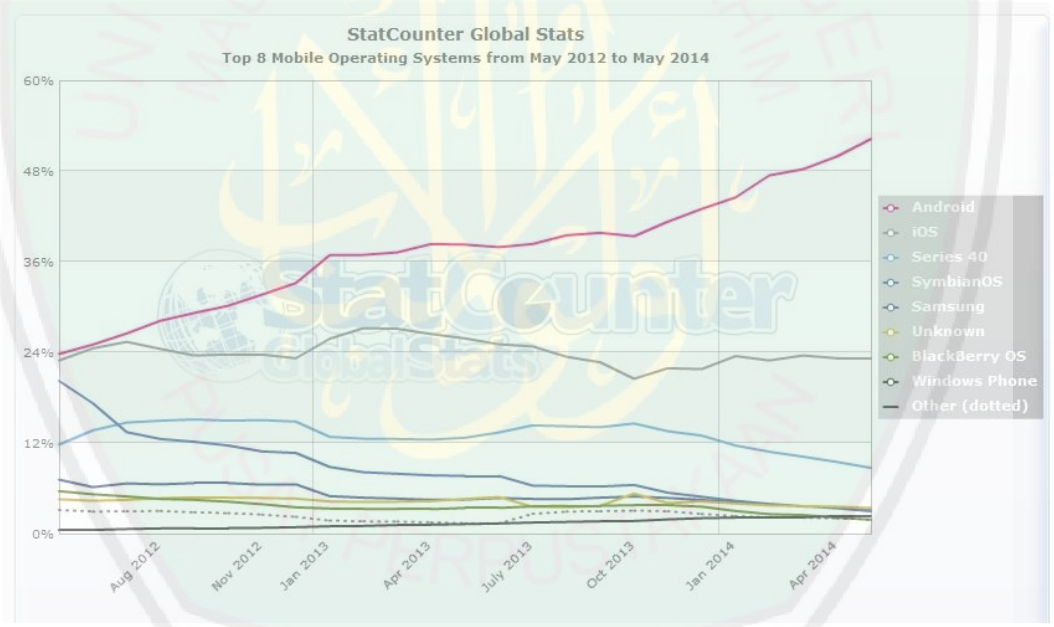
1.1 Latar Belakang

Kamus merupakan sejenis rujukan yang digunakan untuk menerangkan makna kata-kata atau istilah yang sedang dicari. Selain menerangkan arti dari suatu kata, kamus juga memberikan contoh dalam penggunaan kata dalam suatu kalimat, selain itu ada sebagian dari kamus yang memberikan ilustrasi untuk memperjelas makna dan pemakaian dari suatu kata. Menurut M. Dahlan Al Barry(2000:301) menyatakan bahwa "Kamus adalah kitab yang berisi kata-kata dan arti atau keterangan yang disusun secara alfabetik". Sehingga secara tidak langsung, kamus dapat membantu seseorang dalam mengenal suatu kata.

Selain dapat membantu seseorang dalam mengenal suatu kata, kamus juga membantu menerangkan makna kata, cara-cara pengucapan kata dan menerangkan asal kata beserta contoh penggunaannya. Kamus juga memiliki berbagai macam jenis, sesuai dengan isi yang terkandung di dalamnya. Ada kamus bahasa yang digunakan untuk mengenal bahasa-bahasa asing dan kamus istilah yang digunakan untuk menerangkan kata-kata ilmiah. Selain itu ada kamus jenis lain yang digunakan sebagai pedoman dari ilmu tertentu, misalnya kamus komputer dan kamus kedokteran.

Pada zaman sekarang, kamus tidak lagi hanya ada dalam bentuk yang konvensional yakni buku. Akan tetapi kamus telah mengalami banyak kemajuan seiring dengan semakin berkembangnya teknologi. Kamus yang ada sekarang tidak hanya kamus dalam bentuk buku, tetapi telah semakin berkembang dan

muncul pula kamus dalam bentuk aplikasi. Dimana kamus dalam bentuk aplikasi ini lebih mudah digunakan jika dibandingkan dengan kamus dalam bentuk buku. Munculnya kamus dalam bentuk aplikasi ini tidak lain merupakan imbas dari semakin majunya teknologi sekarang terutama bidang mobile. Teknologi mobile telah berkembang dengan pesat terutama mobile yang mengusung sistem operasi android. Bahkan bila dilihat dari pengguna mobile yang menggunakan sistem operasi android, maka dari tahun ke tahun mengalami kenaikan yang signifikan. Seperti yang terlihat pada gambar berikut:



Gambar 1.1 Pengguna Smartphone Indonesia (*Sumber: gs.statcounter.com,* Mei 2012-Mei 2014)

Oleh karena itu, melihat semakin banyaknya pengguna mobile khususnya yang menggunakan sistem operasi android, maka peneliti memutuskan untuk membuat aplikasi kamus kedokteran berbasis mobile dengan menggunakan sistem operasi android. Dalam pembuatannya, peneliti mengimplementasikan dua algoritma yang akan digunakan dalam proses pencariannya. Algoritma yang akan

digunakan adalah algoritma *Interpolation Search* dan algoritma *Aho-Corasick String Matching.*

Kedua algoritma ini memiliki kelebihan masing-masing. Untuk algoritma *Interpolation Search,* algoritma ini merupakan algoritma pencarian yang didasari pada pencarian nomor telepon pada buku telepon, dimana proses pencariannya dengan menggunakan nilai indeks yang ada pada buku telepon. Keunggulan dari algoritma *Interpolation Search* yaitu algoritma ini lebih efisien dibandingkan dengan algoritma *Binary* dan *Sequential Search,* hal ini dikarenakan dalam proses pencariannya algoritma *Interpolation Search* tidak perlu melakukan penjelajahan seluruh elemen dari tabel. Teknik pencarian ini dapat digunakan hanya pada tabel dengan elemen data yang sudah terurut, baik secara menaik atau menurun. Dalam melakukan pencarian, algoritma *Interpolation Search* memperkirakan seberapa jauh letak kemungkinan data yang akan dicari dari posisi saat itu (setelah dilakukan pendekatan dengan menggunakan rumus *interpolasi*).

Sedangkan algoritma *Aho-Corasick String Matching* adalah algoritma yang dapat digolongkan dalam tipe pencarian *multi pattern*, sehingga dalam proses pencariannya tidak perlu mengulang seluruh artikel atau database kata. Selain *aho corasick string matching,* algoritma lain yang tergolong *pattern matching* adalah KMP(*Knuth-Morris-Prat*) dan *Boyer-Moore.* Keunggulan dari algoritma *aho corasick string matching* adalah dapat melakukan beberapa pencocokan kata dalam satu kali proses tanpa mengulang dari awal proses. Serta fungsi *fail* yang dapat mencari karakter lain yang cocok, jika karakter yang dicari tidak cocok.

Berdasarkan perbedaan yang telah dijelaskan di atas, maka peneliti menjadikan perbedaan tersebut sebagai landasan untuk melakukan penelitian dengan cara membandingkan kedua algoritma tersebut. Dimana nantinya dari proses membandingkan ini, peneliti akan mengetahui mana algoritma yang lebih efisien dalam proses melakukan pencarian kata.

Penelitian ini akan mengimplementasikan dua algoritma yang akan dilakukan perbandingan, terhadap aplikasi kamus kedokteran pada perangkat *mobile* yang menggunakan sistem operasi Android dengan harapan dapat diaplikasikan di mana saja dan kapan saja.

1.2 Identifikasi Masalah

Masalah yang dapat diidentifikasi pada penelitian ini adalah:

1. Manakah algoritma yang lebih cepat untuk proses pencarian istilah kedokteran jika dibandingkan antara algoritma *Aho Corasick String Matching* dengan *Interpolation Search*?
2. Seberapa cepat beda waktu yang diperlukan antara dua algoritma di atas?

1.3 Tujuan Penelitian

Tujuan penelitian ini adalah:

1. Membandingkan kinerja algoritma *Aho Corasick String Matching* dan *Interpolation Search* jika ditinjau berdasarkan parameter konsumsi waktu.

2. Mengetahui beda waktu yang dibutuhkan antara *Aho Corasick String Matching* dan *Interpolation Search* dalam proses pencarian istilah kedokteran.

1.4 Manfaat Penelitian

Manfaat yang dapat diambil dengan adanya penelitian ini adalah dapat memberikan pengetahuan dalam membandingkan efektivitas konsumsi minimum waktu yang dibutuhkan oleh algoritma *Interpolation Search* dengan algoritma *Aho-Corasick String Matching* dalam proses pencarian kata pada aplikasi kamus kedokteran berbasis *mobile*.

1.5 Batasan Masalah

Dalam pembuatan aplikasi ini, permasalahan yang akan diselesaikan hanya dibatasi pada permasalahan berikut:

1. Aplikasi ini akan diimplementasikan pada Android OS *Mobile*
2. Istilah kedokteran yang dicari pada aplikasi akan ditemukan penjelasannya apabila istilah tersebut telah terdapat dalam basis data.
3. Pencarian istilah yang akan dicari hanya terbatas pada istilah-istilah kedokteran saja.

1.6 Sistematika Penulisan

BAB I PENDAHULUAN

Bab ini berisikan latar belakang mengenai alasan diadakannya penelitian, rumusan masalah dari penelitian, tujuan, manfaat, batasan masalah, serta sistematika penulisan yang bertujuan untuk memudahkan pembaca memahami pokok pernasalahan yang dibahas.

BAB II TINJAUAN PUSTAKA

Dalam bab ini dijelaskan teori-teori atau argumentasi ilmiah yang dipakai sebagai referensi sesuai dengan permasalahan yang diambil, diantaranya tentang kamus, algoritma *Interpolation Search,* algoritma *Aho-Corasick String Matching,* dan sistem *Android.*

BAB III ANALISIS DAN PERANCANGAN APLIKASI

Bab ini menjelaskan tentang pembuatan analisis dan perancangan program aplikasi kamus, yang meliputi *analisa masalah, analisa system, analisa kebutuhan aplikasi, use case, activity diagram, class diagram,* dan desain *interface.*

BAB IV HASIL DAN PEMBAHASAN

Hasil penelitian dari aplikasi yang telah dibuat, dijelaskan secara rinci pada bab ini. Mulai dari implementasi algoritma *Aho Corasick String Maching* dan *Interpolation Search* terhadap aplikasi *mobil,* penjelasan mengenai source kode yang digunakan pada aplikasi, alur proses pencarian menggunakan masing-masing algoritma, pengujian fungsi dari aplikasi, dan hasil uji coba dari penggunaan algoritma *Aho Corasick String Matching* dan *Interpolation Search* ketika digunakan dalam proses pencarian kata.

BAB V PENUTUP

Bab ini merupakan penutup, yang di dalamnya berisi kesimpulan dari hasil uji coba yang telah dilakukan pada bab sebelumnya, dan rangkuman dari pembahasan penelitian ini, serta berisi saran yang diharapkan dapat bermanfaat untuk pengembangan pembuatan program aplikasi selanjutnya.

BAB II

TINJAUAN PUSTAKA

2.1 Kamus

Menurut kamus besar bahasa Indonesia, pengertian kamus adalah buku acuan yang memuat kata dan ungkapan yang biasanya disusun menurut abjad berikut keterangan tentang maknanya, pemakaiannya dan terjemahannya. Kamus dapat juga digunakan sebagai buku rujukan yang menerangkan makna kata-kata yang berfungsi untuk membantu seseorang mengenal perkataan baru. Selain menerangkan maksud kata, kamus juga mungkin mempunyai pedoman sebutan, asal-usul (etimologi) suatu perkataan dan juga contoh penggunaan bagi suatu perkataan. Untuk memperjelas, kadang kala terdapat juga ilustrasi di dalam kamus. Terdapat banyak kamus yang populer di Indonesia, seperti: kamus bahasa Inggris, bahasa Jerman, bahasa Mandarin, bahasa Jepang dan lain sebagainya.

2.2 *Interpolation Search* (Pencarian Interpolasi)

Pencarian interpolasi merupakan metode pencarian dengan berpedoman posisi relatif kunci data. Jika digambarkan secara umum, seperti saat kita mencari nama pemilik telepon pada buku petunjuk telepon. Misalnya nama pemilik telepon berawalan "B" maka buku telepon yang akan kita buka sekitar 1/3 atau ¼ dari tebal buku. Jadi bukan mencari nama pemilik telepon dari awal buku telepon (Yuswanto,2009).

algoritma interpolation search adalah algoritma untuk pencarian tabel data yang sudah terurut atau file yang memiiki kunci skalar. Pendekatan pada tabel dilakukan dengan menyisipkan nilai kunci yang dicari dengan batasan nilai dari

tabel. Jika elemen yang dicari tidak ditemukan pada posisi yang didekati, maka tabel yang sesuai akan dipilih dan proses penyisipan nilai kunci akan diulangi sampai elemen yang diinginkan telah ditemukan, atau hasil menunjukkan kata dicari tidak ada pada elemen tabel.(Gonnet. 1980).

Algoritma *interpolation search* merupakan algoritma yang dikembangkan dari algoritma *binary search*, yang mana algoritma binary selalu mencari nilai tengah untuk perbandingan, kemudian membuang sebagian dari space yang tidak mendekati kata yang dicari. Algoritma *interpolation search* mencoba untuk mencari pendekatan letak kata yang dicari menggunakan *linear interpolation* (Kukreja,2013).

Batasan dari penggunaan algoritma ini, dalam proses pencarian data yaitu kumpulan data yang ada harus urut sesuai abjad secara *ascending* ataupun *descending.*

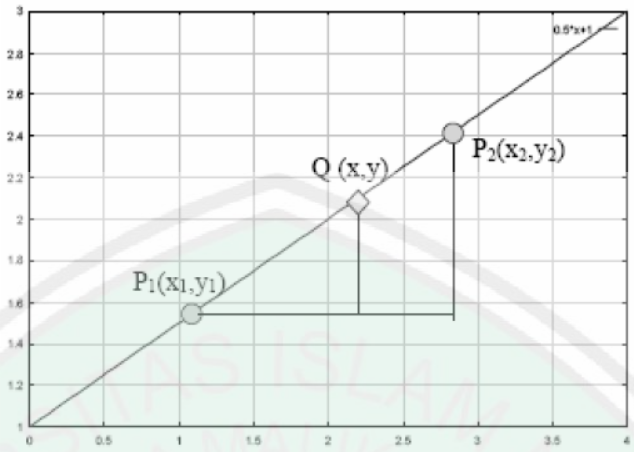
Rumus yang digunakan untuk mencari pendekatan terhadap lokasi kata yang dicari, dapat ditemukan melalui rumus *linear interpolasi* yang digunakan untuk mencari nilai tengah antara dua titik dengan satu garis lurus.(Chapra:409). Berikut rumus *linier interpolation*:

$$\frac{y - y_1}{y_2 - y_1} = \frac{x - x_1}{x_2 - x_1}$$

Jika disusun kembali maka akan menghasilkan rumus seperti berikut:

$$y = y_1 + \frac{x - x_1}{x_2 - x_1}(y_2 - y_1)$$

Yang mana (y) akan menyatakan posisi dari kata yang dicari. Gambar 2.1 menunjukkan persamaan linier interpolasi.



Gambar 2.1. *Interpolation Linear*

Dari rumus linear interpolasi yang telah didapatkan, maka proses pencarian dengan menggunakan metode *interpolation search* dapat dilakukan.

Langkah-langkah algoritma pencarian *interpolation* (Purwanto, 2008):

1. Deteksi n banyak *record array.*
2. Nomor *array*, *low* (kiri) = 0, dan *high* (kanan)= n – 1.
3. Perkirakan posisi data dengan rumus interpolasi berikut:

$$posisi = low + \frac{data\ [kunci] - data\ [low]}{data\ [high] - data\ [low]} * (high - low)$$

4. Lihat *arr*[posisi] dan cocokan dengan kunci :
   i. jika *arr*[posisi] kunci = kunci → pencarian selesai dan data ditemukan.
   ii. jika posisi *arr*[posisi].kunci > kunci → *high=posisi – 1*
   iii. jika posisi *arr*[posisi] kunci < kunci → *low*= posisi + 1
5. Jika data [posisi] ≠ data [kunci], ulangi langkah 3.
6. Jika kunci ≥ *arr*[*low*] kunci dan kunci ≤ *arr*[*high*].kunci, index = - 1, pencarian selesai dan data tidak ditemukan.

Langkah-langkah yang telah disebutkan di atas merupakan penggunaan rumus *interpolation search*, rumus tersebut hanya bisa digunakan untuk data yang berupa angka. Jika ingin menggunakan fungsi *interpolation search* untuk proses pencarian data yang berupa huruf, maka perlu diberikan perlakuan khusus ketika proses pencarian.

Dengan memberikan nilai pada awalan setiap kata yang tersedia, maka pencarian dengan menggunakan metode *interpolation search* dapat dilakukan. Gambar 2.2 merupakan pseudo code dari *interpolation search* yang dapat digunakan untuk proses pencarian menggunakan huruf.

```
Input: k[], k[]',n, x, x',p
Output: x, k[x'].
begin
    set max = n – 1; min = 0;
    while  x' >= k'[min] and x' <= k'[max] do
        p = ((x' – k[min]) * (max - min) + min) / k[max] - k[min]
        if k'[p] = x' then
                if k[p] = x then
                    x → k[p]
                elseif  k[p] < x then
                    min = p + 1
                else
                    max = p – 1
                endif
        elseif k'[p] < x' then
                min = p + 1
        else
                max = p – 1
        endif
    if k[min] = x then
        x → k[min] ← k[x']
    else
        x ≠ k[]
    endif
end
```

Keterangan:
k[] = Kumpulan Kata;
k[]' = Kumpulan data prefix per kata;
n = Jumlah elemen data;
x = data yang dicari;
x'= indeks data yang dicari;
p = indeks posisi data yang dicari;

Gambar 2.2 *Pseudo code* algoritma *interpolation search*

Pada gambar 2.2 menampilkan *pseudocode* dari algoritma *interpolation search*, pada prosesnya terdapat beberapa input data yang harus tersedia sebelum proses algoritma dijalankan, agar proses dapat berjalan semestinya. Output dari algoritma ini adalah indeks data yang dicari jika data tersebut terdapat dalam database.

2.3 Aho-Corasick String Matching

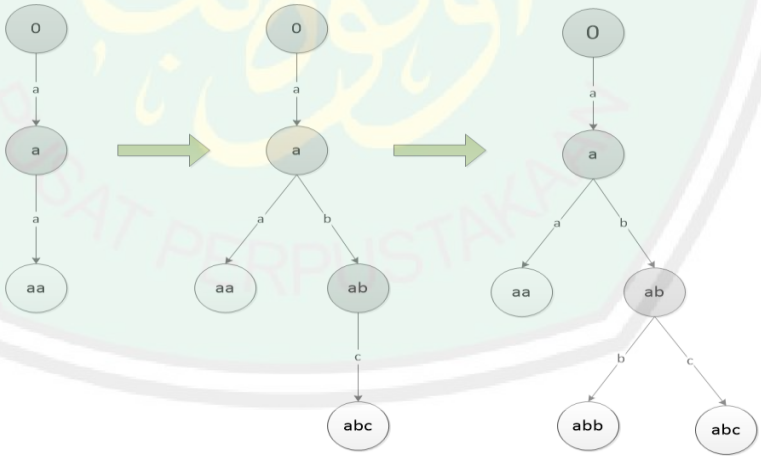
Pencarian *string* dengan menggunakan metode *brute force*, KMP, dan *Boyer-More*, merupakan pencarian satu atau *single pattern*. Jadi jika ingin mencari kata yang lebih dari satu *pattern*, maka perlu mengulang proses pencarian di atas untuk setiap *pattern*. Akibatnya, teks akan dibaca berkali-kali sebanyak jumlah *pattern*. Salah satu metode yang tergolong tipe *multi pattern* adalah *Aho-Corasick*, yang ditemukan oleh *Alfred V. Aho* dan *Margaret J. Corasick* pada tahun 1975. Metode ini semacam algoritma pencocokan kata atau *string* yang dapat menemukan letak elemen dari kumpulan *string* yang terbatas dan sesuai dengan inputan teks, dan proses pencocokan setiap kata akan dilakukan secara serempak.

Algoritma Aho Corasick String Matching memiliki dua bagian. Bagian pertama adalah membangun sebuah tree yang berasal dari kumpulan kata kunci yang ingin dicari, dan bagian kedua adalah proses pencarian teks pada kumpulan kata kunci menggunakan tree yang telah dibangun sebelumnya (*state machine*). Pencarian kata kunci cukup efisien, karena proses tersebut hanya berpindah melalui state-state pada tree. Jika karakter cocok, maka menggunakan *goto function*, jika tidak cocok maka menggunakan *failure function*.(Soewito, 2006).

Algoritma *aho corasick string matching* dapat diterapkan pada kehidupan sehari-hari, khusunya dalam bidang teknologi. beberapa implementasi dari algoritma ini (Hasib, 2013).

1. *Intrusion detection.*
2. Deteksi *plagiarisme,*
3. BioInformatika,
4. Digital Forensik
5. Text Mining

Kumpulan *pattern* akan disimpan pada sebuah tree yang saling terhubung dengan node yang bersangkutan. Misal, jika terdapat sebuah array yang berisi beberapa kata sebagai berikut: “aa”, “abb”, “abc”, maka akan disusun sebuah tree dengan tahapan seperti yang ada pada gambar berikut:



Gambar 2.3 Tahapan pembuatan tree pada algoritma *Aho-Corasick*

Untuk tree yang paling kanan merupakan hasil dari pembentukan tree berdasarkan kata-kata yang ada. Setelah terbentuk, maka tree siap dibandingkan dengan teks yang sedang dicari.

Untuk menjalankan algoritma *aho-corasick* terdapat tiga langkah yang harus dilakukan, yaitu:

1. *Goto function*, terdiri dari kumpulan kata yang diurutkan dan terdapat dalam state, kemudian masing-masing state saling berhubungan dengan state yang lain membentuk sebuah *trie*. Terkadang akan mengeluarkan *output* jika ditemukan ketika proses sedang berlangsung, dan akan memberikan pesan gagal jika tidak ditemukan pasangan yang cocok. Seperti gambar 2.3.

```
Input: k[] = Kumpulan kata
Output: go, out
begin
        newstate ← 0
        for i ← 1 until k do enter(y_i)
        for all such a that g(0, a)= fail do g(0, a)← 0
end
procedure enter(a_1, a_2, a_3, ..,.., a_m):
begin
   state ← 0; j ← i
   while go(state, a_j) ≠ fail do
      begin
         state ← go(state, a_j)
         j ← j + 1
      end
      for p ← j until  m do
         begin
               newstate ←  newstate + 1
               go(state, a_p ) ← newstate
               state ← newstate
         end
      out(state) ← { a_1, a_2, a_3, ..,.., a_m }
end
```

Keterangan:
go = goto function
out = output function

Gambar 2.4 *pseudo code* dari *goto function*

Pada gambar 2.4 fungsi dari *goto function* yaitu membuat *tree* yang terdiri dari kata-kata yang tersedia dan terdiri dari state yang berisi per

karakter. Output dari fungsi ini adalah *tree* yang telah tersusun, dan nanti akan digunakan dalam proses pencocokan kata.

Berikutnya adalah *failure function* yang akan dieksekusi ketika proses pencocokan karakter tidak sesuai dengan state satu dengan state selanjutnya. Pada gambar 2.5 merupakan gambar dari *pseudocode* dari *failure function.*

2. *Failure function,* fungsi yang akan dijalankan ketika *goto function* memberikan pesan gagal, atau ketika tidak ada karakter yang cocok, maka akan melompat ke state yang cocok.

```
Input : g,  out ← g
Output: f , out
begin
    queue ←  empty
    for each a such that g(0, a) = s ≠ 0 do
        begin
            queue ←  queue ∪ {s}
            f(s) ←  0
        end
    while queue ≠ empty do
        begin
            r akan menjadi state selanjutnya di queue
            queue ←  queue ← {r}
            for each a such that g(r, a) = s ≠ fail do
                begin
                    queue ←  queue ∪ {s}
                    state ← f(r)
                    while g(state, a) = fail do state ← f(state)
                    f (s) ← g(state, a)
                    output(s) ← output(s) ∪ output (f (s))
                    endwhile
                end
            endfor
        end
end
```

Keterangan:
f = *failure function*
out = *output function*
s = karakter yang dicari

Gambar 2.5 *pseudo code* dari *failure function*

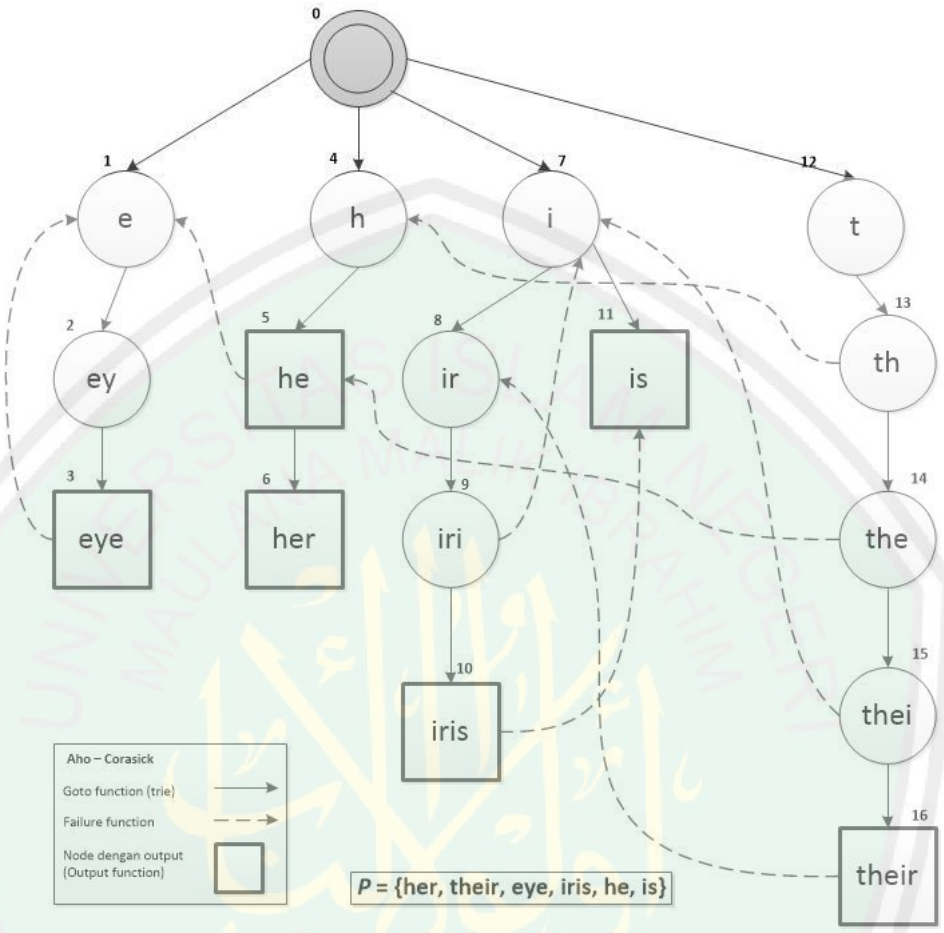
Failure function akan dieksekusi ketika karakter pada state berikutnya tidak sesuai dengan karakter yang sedang dicari. Maka pencocokan akan dikembalikan kepada state awal, kemudian dilanjutkan kepada proses *goto function*, sampai akhir karakter dari kata yang dicari.

Output dari *failure function* berupa state baru yang berupa *output function* atau *goto function*. Ketika akhir dari kata yang dicari, karakternya terdapat pada *leaf* dari *tree*, maka variabel boolean *leaf* akan bernilai *true*.

3. *Output function*, fungsi yang mengeluarkan *output* yang cocok dengan kata yang dicari. Terkadang kosong jika tidak ada karakter yang cocok sehingga akan dilakukan proses *failure function*.

Proses *failure function* bertujuan agar algoritma tidak mencari dan membandingkan karakter itu lagi. Sehingga dapat menghemat waktu dalam proses pencarian.

Pada gambar 2.5 menunjukkan gambaran *tree* atau pohon dari algoritma *aho corasick string matching*, dimulai dari root yang menjadi titik awal ataupun link antara satu state dengan state yang lain, kemudian *failure function* yang ditunjukkan dengan panah yang putus-putus, dan *output function* yang diwakilkan dengan kotak yang berisi output dari kata.



Gambar 2.6 *Aho Corasick String Matching Tree*

2.4 Sistem Operasi Android

Sistem operasi android adalah sebuah sistem operasi untuk perangkat *mobile* berbasis linux yang mencakup sistem operasi, *middleware,* dan aplikasi. (Safaat,2011a). *Platform* android bermula dari sebuah perusahaan yang berkecimpung di dunia *IT & Communication* khususnya bergerak dalam bidang perangkat lunak dengan nama Android Inc. yang kemudian seluruh sahamnya dibeli oleh perusahaan raksasa yaitu Google Inc., setelah itu dibentuklah *Open Handset Alliance,* yaitu konsorium dari 34 perusahaan piranti keras, piranti lunak dan telekomunikasi termasuk di dalamnya yaitu : Google, HTC, Intel, Motorola,

Qualcomm, T – Mobile dan Nvidia. Adapun ikon dari sistem operasi android ini berbentuk robot berwarna hijau, berikut adalah gambar tersebut :

Gambar 2.7 *ikon Android*

Dari perkembangan sistem operasi android ini sekarang menjadi sangat populer karena bersifat *open source* menjadikannya sebagai sistem operasi yang banyak diminati oleh banyak pengguna. Adapun beberapa kelebihan dari sistem operasi android adalah sebagai berikut:

1. *Complete Platform*

   Sistem operasi android adalah sistem operasi yang banyak menyediakan *tools* yang berguna untuk membangun sebuah aplikasi yang kemudian aplikasi tersebut dapat lebih dikembangkan lagi oleh para *developer.*

2. *Open Source Platform*

   Platform Android yang bersifat *open source* menjadikan operasi ini mudah dikembangkan oleh para *developer* karena bersifat terbuka.

3. *Free Platform*

Developer dengan bebas bisa mengembangkan, mendistribusikan dan memperdagangkan sistem operasi android tanpa harus membayar royalti untuk mendapatkan *license*.

2.5 Arsitektur Android

Penggambaran sebuah arsitektur dari sistem operasi Android, jika dilihat secara garis besar adalah sebagai berikut (Safaat,2011b):

1. *Applications dan Widgets*

   Layer yang berhubungan dengan aplikasi yang ada, dimana aplikasi tersebut diunduh, dipasang, serta dijalankan. Sebagai contoh adalah aplikasi SMS (*Short Message Service*), kalender, galeri foto, *email*, kontak, *browser* dan lain sebagainya.

2. *Applications Frameworks*

   Layer dimana para pembuat aplikasi melakukan pengembangan atau pembuatan aplikasi yang akan dijalankan di sistem operasi Android, karena pada layer inilah aplikasi dapat dirancang dan dibuat. Adapun komponen dalam layer *aplications frameworks* adalah sebagai berikut:

   1. *Views*
   2. *Content provider*
   3. *Resources manager*
   4. *Notification manager*
   5. *Activity manager*

3. *Libraries*

   Layer yang menyediakan berbagai fitur – fitur dalam sistem operasi Android berada, biasanya pembuat aplikasi mengakses *libraries* untuk menjalankan aplikasinya.

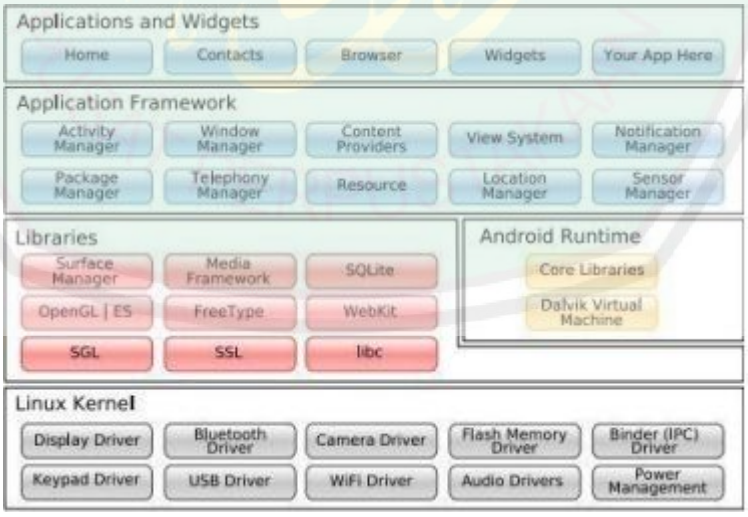
4. *Android Run Times*

   Layer yang membuat aplikasi Android dapat dijalankan dimana dalam prosesnya menggunakan implementasi linux.

5. *Linux Kernel*

   Layer dimana inti *operating system* dari Android itu berada yang berisi file – file sistem yang mengatur sistem *processing, memory, resources, drivers dan sistem operasi Android lainnya.*

Adapun gambaran arsitektur Android adalah sebagai berikut:



Gambar 2.8 *arsitektur pada sistem operasi Android*

BAB III

ANALISA DAN PERANCANGAN

3.1 Analisa Masalah

Langkah-langkah yang perlu ditempuh untuk mengetahui berbagai masalah yang ada pada pemanfaatan teknologi pada kamus, sehingga dengan adanya kamus istilah kedokteran berbasis android ini diharapkan bisa membantu permasalahan yang dihadapi oleh penggunanya. Adapun masalah tersebut adalah sebagai berikut:

1. Kamus dalam bentuk buku memiliki kekurangan karena sifat kertas yang rentan robek dan rusak.
2. Kurang efektif dan efisien dengan menggunakan kamus dalam bentuk buku karena terlalu lama mencari dan menghabiskan banyak ruang.
3. Mengalami kesulitan dalam penambahan kosa-kata ke dalam kamus dalam bentuk buku. (Isnani: 2009)

3.2 Analisa Kebutuhan Algoritma

Dalam subbab ini akan menjelaskan mengenai analisa kebutuhan dari proses perbandingan yang akan diteliti dari masing-masing algoritma. Sebelum melangkah ke proses perbandingan, setiap algoritma memiliki kebutuhan tersendiri untuk proses pencarian yang akan dilakukan.

3.2.1 Algoritma *Aho-Corasick String Matching*

Dalam proses pencarian menggunakan algoritma ini, prosesnya berbeda dengan menggunakan algoritma *interpolation search*. Karena algoritma ini tergolong algoritma *string matching*, tapi tidak menutup kemungkinan untuk

digunakan dalam proses pencarian. Berikut beberapa kebutuhan yang harus tersedia jika menggunakan algoritma ini.

1. Terdapat kumpulan kata yang akan dijadikan sebagai *tree.*
2. Untuk membuat *tree* cukup dilakukan satu kali dari beberapa proses pencarian.
3. Karena tergolong algoritma *string matching*, maka hasil dari pencarian bisa lebih dari satu. Sehingga perlu diberikan tindakan yang berbeda dalam proses menampilkan hasil pencarian.
4. Perhitungan konsumsi waktu tidak dimulai ketika *tree* dibuat.

3.2.2 Algoritma *Interpolation Search*

Untuk proses pencarian menggunakan algoritma *interpolation search*, beberapa kebutuhan yang harus tersedia dalam sistem. Adapun kebutuhan tersebut adalah sebagai berikut:

1. Kumpulan kata yang akan dijadikan referensi pencarian.
2. Menggunakan rumus *interpolation search*, seperti yang telah dijelaskan pada subbab 2.2.
3. Perhitungan konsumsi waktu dilakukan ketika proses pencarian akan dilakukan.

3.3 Analisa Sistem

Dalam pembuatan proyek perangkat lunak diperlukan adanya analisa dan perancangan dari sistem yang akan dibuat, hal ini bertujuan agar proyek yang akan dibuat dapat selesai tepat waktu dan sesuai dengan tujuan awal. Kesalahan

dalam analisa akan berimbas pada tahapan proyek selanjutnya dan mengakibatkan proyek tidak maksimal dan sesuai dengan tujuan pembuatan proyek.

Tahapan analisis ini berisi identifikasi dan evaluasi permasalahan yang ada serta kebutuhan apa saja yang dibutuhkan dalam pembuatan proyek perangkat lunak. Dengan adanya analisa dan perancangan maka akan didapatkan suatu gambaran mengenai kebutuhan apa saja yang dibutuhkan dalam program aplikasi ini. Langkah - langkah dalam penggunaan aplikasi untuk user dari awal hingga akhir adalah sebagai berikut:

1. User menjalankan aplikasi dan masuk ke halaman user.
2. Di dalam halaman user ini terdapat menu pilih algoritma,pencarian, daftar kata, pengaturan, tentang dan exit.
3. Pada menu pilih algoritma, user dapat memilih algoritma yang akan digunakan dalam pencarian data sesuai dengan yang di inginkan.
4. Kemudian user akan masuk ke halaman pencarian dan memasukkan kata yang akan dicari penjelasannya.
5. Hasil dari pencarian akan dikeluarkan pada form output berserta waktu pencariannya sesuai dengan algoritma yang digunakan.
6. Pada menu daftar kata, uer dapat melihat kata apa saja yang ada atau terdaftar dalam kamus kedokteran.
7. Pada menu pengaturan, user dapat mengatur kembali penggunaan algoritma yang akan digunakan pada aplikasi kamus.
8. Kemudian user juga dapat melakukan pengaturan untuk tampilan apakah waktu hasil pencarian ditampilkan atau tidak.

9. Selain itu, pada menu pengaturan ini user dapat login untuk menjadi admin yang dapat menambah, mengedit serta menghapus data pada aplikasi kamus.
10. Pada menu tentang, user dapat mengetahui tentang siapa pembuat aplikasi kamu kedokteran.

3.4 Perancangan Sistem

Pada tahapan perancangan sistem ini akan digambarkan secara garis besar tentang program aplikasi kamus istilah kedokteran berbasis android dengan membandingakan algoritma *Aho Corasick String Matching* dan *Interpolation Search* dalam pengimplementasiannya.

Aplikasi kamus istilah kedokteran berbasis android ini bertujuan untuk membantu pengguna yang ingin cepat dan praktis dalam mencari kosa-kata istilah kedokteran. Dengan aplikasi ini, pengguna disuguhkan pilihan mencari istilah kedokteran dengan menggunakan dua algoritma yang berbeda untuk pembandingnya.

Pengguna dapat menggunakan pilihan pencarian dengan menggunakan algoritma *Aho Corasick String Matching* ataupun algoritma *Interpolation Search* guna membandingkan kedua algoritma tersebut berdasarkan waktu pencarian yang didapatkan. Selain memudahkan penggunanya dalam mencari kosa-kata serta membandingkan dua algoritma berdasarkan waktu pencarian yang didapat, aplikasi ini juga bersifat gratis tanpa harus membayar untuk mendapatkannya.

3.5 Mekanisme Pengujian Algoritma

Pengujian dilakukan terhadap dua algoritma pencarian, yang bertujuan untuk membandingkan konsumsi waktu yang dibutuhkan oleh masing-masing algoritma dalam proses pencarian.

Setiap algoritma akan diujikan terhadap dua buah perangkat keras smartphone yang berbeda spesifikasinya, hal ini dilakukan bertujuan untuk mengetahui pengaruh spesifikasi hardware terhadap perbandingan waktu konsumsi. Pengujian pertama dilakukan terhadap smartphone A yang memiliki spesifikasi lebih tinggi daripada smartphone B, pengujian dibagi menjadi tiga bagian,

i. Pengujian pertama, dilakukan dengan menggunakan kata yang berawalan huruf ‘a’, tapi tidak pada posisi paling awal. Kemudian dilakukan proses pencarian sebanyak sepuluh kali dengan kata yang sama, agar didapatkan rata-rata dari konsumsi waktu yang yang diperlukan.
ii. Selanjutnya, uji coba dilakukan dengan menggunakan kata yang posisi indeksnya berada pada sekitar pertengahan dari kumpulan kata. Kemudian dilakukan pencarian sebanyak sepuluh kali dengan kata yang sama, agar didapatkan rata-rata dari konsumsi waktu yang dibutuhkan.
iii. Kemudian uji coba yang terakhir dilakukan dengan menggunakan kata yang posisi indeksnya berada pada sekitar akhir dari kumpulan kata, tetapi tidak pada posisi paling akhir. Seperti sebelumnya, pencarian

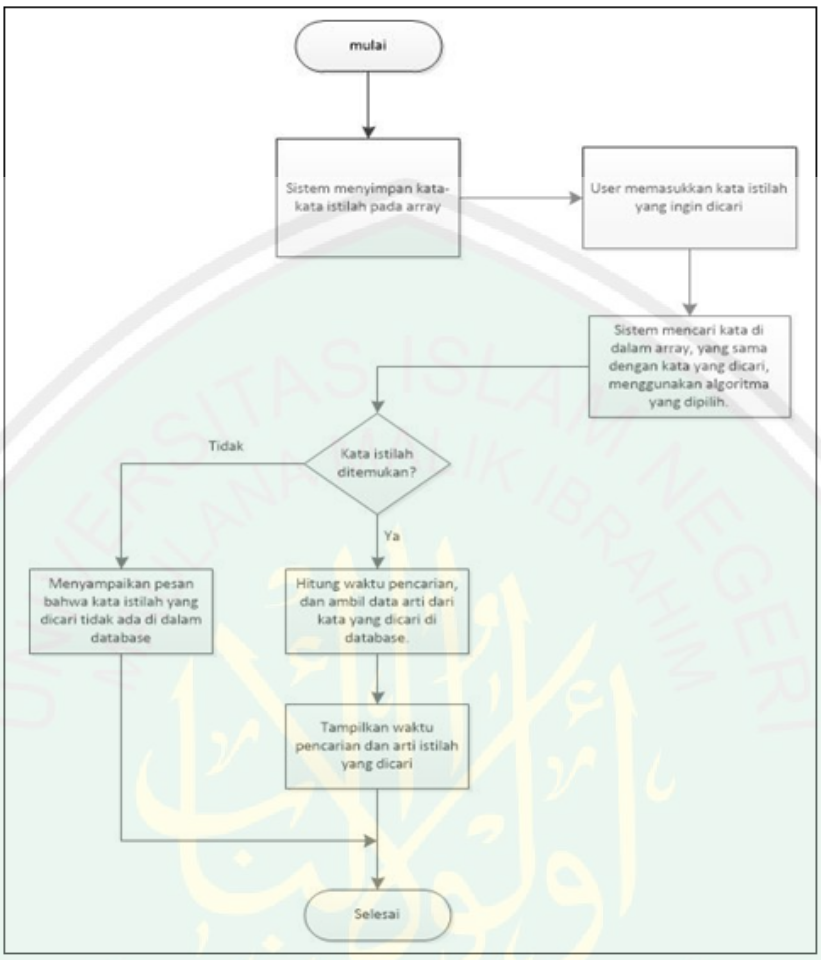
akan dilakukan sebanyak sepuluh kali dengan kata yang sama, agar didapatkan rata-rata dari konsumsi waktu yang dibutuhkan.

Kemudian untuk pengujian terhadap smartphone B, yang memiliki spesifikasi lebih rendah daripada smartphone A, juga dilakukan pengujian yang dibagi menjadi tiga bagian, dan masing-masing pengujian dilakukan proses pencarian sebanyak sepuluh kali. Jadi total uji coba yang dilakukan sebanyak enam kali uji coba.

Kemudian setelah rata-rata dari masing-masing pengujian didapatkan, maka rata-rata tersebut dikumpulkan dan dibagi menjadi dua bagian, disesuaikan dengan perangkat yang digunakan. Setelah itu, pada setiap rata-rata dari masing-masing pengujian dihitung nilai varian agar dapat diketahui selisih rata-rata konsumsi waktu yang dibutuhkan dari proses pencarian.

3.6 Rancangan Algoritma *Aho-Corasick String Matching dan Interpolation Search*

Alur penerapan algoritma *Aho Corasick String Matching* dan *Interpolation Search* tidak berbeda dengan satu dengan yang lain, algoritma akan dijalankan ketika tombol cari ditekan, pada gambar 3.1 menampilkan gambar flowchart dari penerapan algoritma terhadap aplikasi kamus kedokteran.



Gambar 3.1 Flowchart rancangan algoritma

3.6.1 Rancangan Implementasi *Aho Corasick String Matching*

Algoritma *Aho Corasick String Matching* diimplementasikan pada sebuah kelas bernama *AhoCorasick.java* yang nanti akan diakses oleh kelas *MainActivity.java.* Pada kelas akan dideklarasikan fungsi *arraylist* yang digunakan untuk menyimpan kata-kata yang nanti akan digunakan pada proses pencarian.

```
ArrayList<Kamus> kata = new ArrayList<Kamus>();
long time_awal, time_akhir;

public ArrayList<Kamus> getKata() {
      return kata;
}

public void setKata(ArrayList<Kamus> istilah) {
      this.kata = istilah;
}
```

Gambar 3.2 *Source code* deklarasi daftar kata

Algoritma *aho corasick* membutuhkan *tree* dalam proses pencariannya. Sebelum proses pencarian dimulai, maka harus dideklarasikan terlebih dahulu *root* dari kelas *Tree* yang memiliki beberapa atribut seperti yang terlihat pada gambar 3.

```
public static Tree createRoot() {
    Tree node = new Tree();
    node.failLink = node;
    node.index = 0;
    node.huruf = '0';
    return node;
}
```

Gambar 3.3 attribut pada fungsi *createRoot()*

Awal pembuatan *tree* dibuat dengan membuat *root* terlebih dahulu. Atribut yang dideklarasikan adalah indeks, huruf, dan *failLink*. Proses pembuatan *root* dimulai dengan membuat objek terlebih dahulu dari kelas *Tree.java* seperti yang terlihat pada gambar 3.4. Atribut yang ada pada kelas *tree* adalah *children* dari kelas objek *arraylist*, variabel *leaf* yang merupakan variable *Boolean* dan diinisialisasi dengan nilai *false*, *parent* dari kelas *Tree* itu sendiri, variable huruf yang merupakan variable *character*, variable *index* dari variabel *integer*, dan variabel *failLink* dari kelas *Tree*.

```
public class Tree {
     ArrayList<Tree> children = new ArrayList<Tree>();
     boolean leaf = false;
     Tree parent;
     char huruf;
     int index;
     Tree failLink;
}
```

Gambar 3.4 Objek kelas *tree*

Setelah proses pembuatan *root*, dimulai proses penambahan kata yang akan dimasukkan dalam *tree* yang tersedia dengan menggunakan fungsi *addString()*.

```
public static void addString(Tree node, String s, ArrayList<Tree>
global) {
    char[] c = s.toCharArray();
    for (int i = 0; i < c.length; i++) {
        Tree newTre = new Tree();
        if (global.size() <= 1) {
            while (i < c.length) {
                newTre = new Tree();
                newTre.huruf = c[i];
                newTre.index = i + 1;
                newTre.parent = node;
                node.children.add(newTre);
                global.add(newTre);
                node = newTre;
                i++;
            }
        } else if (searchChild(node, c[i])) {
                node = searchGetChild(node, c[i]);
        } else {
                int lastInd = global.get(global.size() - 1).index;
                Tree n = new Tree();
                n.index = lastInd + 1;
                n.huruf = c[i];
                n.parent = node;
                node.children.add(n);
                global.add(n);
                node = n;
        }
    }
    node.leaf = true;
}
```

Gambar 3.5 fungsi *addString()*

Pada gambar 3.5 menampilkan fungsi *addString* yang di dalamnya terdapat beberapa parameter, yaitu: kelas *tree*, *String* kata yang akan ditambahkan

pada *tree*, dan *arrayList tree* yang digunakan untuk mengetahui panjang *tree* secara keseluruhan.

Dalam proses penambahan kata, kata tersebut akan diproses per karakter, dalam proses ini terdapat tiga kondisi. Pertama, ketika *tree* masih berupa *root* saja. Kedua, ketika karakter yang akan ditambahkan telah ada pada *tree*. Ketiga, ketika karakter yang akan ditambahkan tidak sama dengan *children tree*.

Pada kondisi kedua terdapat fungsi *searchChild()* yang akan mengembalikan nilai *boolean true* jika ada karakter yang sama, kemudian akan diproses fungsi *searchGetChild()* yang berfungsi untuk menambahkan karakter pada *tree* yang tersedia, dan memberikan nilai boolean true pada indeks.

```
private static boolean searchChild(Tree node, char dicari) {
    boolean ketemu = false;
    for (int i = 0; i < node.children.size(); i++) {
        if (node.children.get(i).huruf == dicari) {
            ketemu = true;
            break;
        }
    }
    return ketemu;
}

private static Tree searchGetChild(Tree node, char dicari) {
    Tree ketemu = null;
    for (int i = 0; i < node.children.size(); i++) {
        if (node.children.get(i).huruf == dicari) {
            ketemu = node.children.get(i);
            break;
        }
    }
    return ketemu;
}
```

Gambar 3.6 *Source code* untuk mencari *children*

Setelah proses penambahan kata untuk membentuk *tree*, proses pencarian dapat dilakukan. Fungsi yang akan digunakan dalam proses pencarian ini terdapat pada fungsi *go()* yang terdapat pada kelas *AhoCorasick*.

```
public static Tree go(Tree node, char ch) {
    Tree ketemu;
    if (searchChild(node, ch)) {
        node = searchGetChild(node, ch);
        ketemu = node;
    } else {
        ketemu = (node.parent == null)? node:
go(failFunction(node), ch);
    }
    return ketemu;
}
```

Gambar 3.7 *Source code* fungsi *go()*.

Pada gambar 3.7 dapat diketahui parameter yang ada pada fungsi *go()* adalah *node* dari kelas *Tree*, dan karakter dari kata yang akan dicari. Fungsi ini akan dijalankan per karakter dari kata yang dicari, dan akan mengembalikan class *tree*, dan *boolean true* jika akhir karakter dari kata terdapat pada *tree* yang dibuat. Jika karakter selajutnya tidak sama dengan karakter pada *tree* yang ada. Maka akan dilakukan *failure function* yang diwakilkan pada fungsi *failFunction()*.

```
public static Tree failFunction(Tree node){
    if(node.failLink == null){
        if (node.parent.parent == null) {
            node.failLink = node.parent;
        } else {
            node.failLink = go(failFunction(node.parent),
node.huruf);
        }
    }
    return node.failLink;
}
```

Gambar 3.8 fungsi *failFunction()*

Fungsi *failFunction()* akan mengembalikan karakter yang tidak sama pada *root* dari *tree*, kemudian fungsi akan mencari karakter yang sama dengan karakter yang sedang dicari. Jika ada karakter yang sama, maka akan dilanjutkan ke karakter tersebut, dan dilanjutkan sampai karakter yang terakhir. Jika tidak ada karakter yang sama, maka fungsi *failFunction()* akan mengembalikan variabel *leaf* dengan nilai *boolean false*. Ketika akhir dari proses perulangan karakter, dan variabel *leaf* pada waktu itu mengembalikan nilai *true*, berarti kata yang dicari terdapat pada database.

Dalam implementasi algoritma *aho corasick string matching* pembuatan *tree* akan dilakukan sekali saja ketika aplikasi pertama kali berjalan. Proses pembuatan *tree* terdapat pada kelas *MainActivity.java* dan diwakilkan pada fungsi *cekStringAho()*. Terdapat kondisi yang akan menjadi pemicu fungsi *cekStringAho()* akan dieksekusi, yaitu ketika panjang array pada variabel kata yang ada pada kelas *AhoCorasick.java* bernilai nol. *Source code* dapat dilihat pada gambar 3.9.

```
private void cekStringAho(){
      db.cekArray(MainActivity.this, 2);
      tree = AhoCorasick.createRoot();
      ArrayList<Tree> global = new ArrayList<Tree>();
      global.add(tree);
      for (int i = 0; i < aho.getKata().size(); i++) {
            AhoCorasick.addString(tree, aho.getKata().get(i)
                        .getIstilah(), global);
      }
}
```

Gambar 3.9 fungsi *cekStringAho()*

Pada kelas *AhoCorasick.java* terdapat fungsi *cekKata()* yang digunakan untuk melakukan proses pencarian, dan diakses oleh kelas *MainActivity.java*

dalam penggunaanya. Fungsi ini akan mengembalikan nilai *boolean* dari variabel *leaf.*

```
public boolean cekKata(Tree pohon, String find, MainActivity
main){
    time_awal = System.nanoTime();
    Tree node = pohon;
    for (char ch : find.toCharArray()) {
        node = go(node, ch);
    }
    time_akhir = System.nanoTime();
    main.setTime_awal(time_awal);
    main.setTime_akhir(time_akhir);
    return node.leaf;
}
```

Gambar 3.10 fungsi *cekKata()*

Gambar 3.10 menampilkan fungsi *cekKata()* yang memiliki parameter 'pohon' dari kelas *Tree*, variabel *find* yang bertipe *String*, dan objek dari kelas *MainActivity.java.* Sebelum pencarian dimulai terdapat variabel time_awal yang berisi waktu saat itu dalam format *nanosecond*, setelah itu akan dilakukan perulangan sepanjang karakter dari kata yang dicari. Setelah selesai maka akan ditentukan variabel time_akhir yang berisi waktu saat itu dalam format *nanosecond* juga. Untuk mendapatkan waktu yang dibutuhkan ketika proses pencarian, perlu dilakukan perhitungan time_akhir dikurangi dengan time_awal, hasil dari proses perhitungan tersebut akan menjadi waktu yang yang dibutuhkan dalam proses pencarian.

3.6.2 Rancangan Implementasi *Interpolation Search*

Implementasi dari algoritma *interpolation search* diterapkan pada kelas *InterpolationSearch.java* yang nanti akan diakses oleh kelas *MainActivity.java* dalam penggunaannya. Pada awal kelas dideklrasikan terlebih dahulu variabel objek 'istilah' dari ke *arraylist* untuk menyimpan kata-kata yang nanti digunakan dalam pencarian.

```
ArrayList<String> istilah = new ArrayList<String>();
public static final int ITEM_NOT_FOUND = -1;
long time_awal, time_akhir;

public ArrayList<String> getIstilah() {
      return istilah;
}

public void addIstilah(String kata) {
      istilah.add(kata);
}
```

Gambar 3.11 *Source code* untuk mendeklarasikan varibel objek

Selain variabel objek 'istilah' terdapat variabel objek lainnya, yaitu variabel objek *ITEM_NOT_FOUND* yang merupakan objek dari kelas *Integer*, dan time_awal dan time_akhir yang merupakan objek dari kelas *Long* dan nanti akan digunakan pada perhitungan waktu yang dibutuhkan dalam proses pencarian.

Fungsi *searchWithInterpolation()* pada kelas *InterpolationSearch.java* berfungsi untuk melakukan proses pencarian dengan menggunakan metode *interpolation search* yang nanti akan mengembalikan nilai -1, jika kata yang dicari tidak ada pada daftar kata yang telah dibuat, dan akan mengembalikan nilai indeks dari kata yang dicari, jika kata yang dicari ada di dalam daftar kata.

```
public int searchWithInterpolation(ArrayList<String> Array, String obj) {

        int low = 0;
        int high = Array.size() - 1;
        int mid;
        int d_pref[] = new int[Array.size()];
        for (int i = 0; i < d_pref.length; i++) {
                d_pref[i] = prefik(Array.get(i));
        }
        int obj_pref = prefik(obj);

        while (obj_pref >= d_pref[low] && obj_pref <= d_pref[high]) {
                mid = low + ((obj_pref - d_pref[low]) * (high - low))
                                / (d_pref[high] - d_pref[low]);

                if (d_pref[mid] == obj_pref) {
                        if (Array.get(mid).equals(obj)) {
                                return mid;
                        } else if (Array.get(mid).compareTo(obj) < 0) {
                                low = mid + 1;
                        } else if (Array.get(mid).compareTo(obj) > 0) {
                                high = mid - 1;
                        }
                } else if (d_pref[mid] < obj_pref) {
                        low = mid + 1;
                } else if (d_pref[mid] > obj_pref) {
                        high = mid - 1;
                }
        }
        if (Array.get(low).compareTo(obj) == 0) {
                return low;
        } else {
                return ITEM_NOT_FOUND;
        }
}
```

Gambar 3.12 fungsi pencarian menggunakan*interpolation search*

Algoritma *interpolation search* akan membandingkan kata yang dicari dengan kata-kata yang ada pada database. Untuk menghindari perbandingan secara keseluruhan, maka perbandingan hanya dilakukan dengan kata-kata yang memiliki awalan kata atau prefiks yang sama dengan kata yang dicari. Setiap prefix dari kata akan memiliki nilai, pada kelas *InterpolationSearch.java* memiliki

fungsi *prefik()* yang berfungsi untuk memberikan nilai pada kata yang akan diproses menggunakan algoritma *interpolation search.*

```
public static int prefik(String kata) {
      ArrayList<String> huruf = new ArrayList<String>();
      huruf.add("0");huruf.add("1");huruf.add("2");
      huruf.add("3");huruf.add("4");huruf.add("5");
      huruf.add("6");huruf.add("7");huruf.add("8");
      huruf.add("9");
      huruf.add("a");huruf.add("b");huruf.add("c");
      huruf.add("d");huruf.add("e");huruf.add("f");
      huruf.add("g");huruf.add("h");huruf.add("i");
      huruf.add("j");huruf.add("k");huruf.add("l");
      huruf.add("m");huruf.add("n");huruf.add("o");
      huruf.add("p");huruf.add("q");huruf.add("r");
      huruf.add("s");huruf.add("t");huruf.add("u");
      huruf.add("v");huruf.add("w");huruf.add("x");
      huruf.add("y");huruf.add("z");

      char text[] = kata.toLowerCase().toCharArray();
      String nilai = String.valueOf(text[0]);
      int value = 0;
      for (int i = 0; i <= huruf.size(); i++) {
            if (nilai.equals(huruf.get(i))) {
                  value = i + 1;
                  break;
```

Gambar 3.13 fungsi *prefik()*

Fungsi *prefik()* seperti yang terlihat pada gambar 4.11 akan mengembalikan nilai karakter prefik dari kata yang dicari. Jadi jika awalan kata tidak terdapat pada daftar karakter awalan yang ada pada fungsi *prefik()* maka algoritma akan mendapatkan *error* atau tidak bekerja.

Algoritma *interpolation search* akan dijalankan ketika aplikasi berjalan pertama kali, variabel objek ‘istilah’ akan diisi kata-kata yang akan digunakan dalam proses pencarian. Ketika panjang array dari variabel objek “istilah“ sama dengan nol, maka fungsi *cekArray()* yang ada pada kelas *DatabaseHelper.java*

akan dijalankan untuk menambahkan kata-kata pada variabel objek tersebut. Gambar 3.14 menunjukkan potongan fungsi *cekArray()*.

```
public void cekArray(MainActivity Main, int type) {
      Cursor cur;
      MainActivity main = Main;
      InterpolationSearch inter = main.getInter();
      AhoCorasick aho = main.getAho();

      cur = db.query(tb_data, new String[] { col_istilah,
col_arti }, null, null, null,null, null);
      int row = cur.getCount();
      if (cur.moveToFirst()) {
            do {
            if (type == 1) {
      inter.addIstilah(cur.getString(cur.getColumnIndexOrThro
w(col_istilah)).toLowerCase().trim());
      } else {
            Kamus kamus = new Kamus();
      kamus.setArti(cur.getString(cur.getColumnIndexOrThrow(c
ol_arti)));
            kamus.setIstilah(cur.getString(cur
            .getColumnIndexOrThrow(col_istilah)).toLowerCase(
      ).trim());
            aho.getKata().add(kamus);
            }
      } while (cur.moveToNext());
}
```

3.14 fungsi *cekArray()*

Ketika proses penambahan kata telah dilakukan, maka proses pencarian dapat dilakukan. Pada kelas *MainActivity.java* pencarian akan dilakukan jika nilai dari variabel objek dari *interpolation_search* bernilai *true*, kemudian sistem akan menjalankan fungsi *cari()* dengan parameter kata yang dicari dengan tipe *String* dan objek kelas dari *MaiActivity.java*. fungsi *cari()* dideklarasikan pada kelas *InterpolationSearch.java* yang akan mengembalikan nilai -1 jika kata tidak ditemukan, dan akan mengembalikan indeks dari kata yang dicari jika kata tersebut ada pada *array* "istilah".

```
public void cari(MainActivity Main, String cari) {
      MainActivity main = Main;
      InterpolationSearch is = main.getInter();

      time_awal = System.nanoTime();
      int hasil;
      String find = cari.toLowerCase();
      hasil = is.searchWithInterpolation(istilah, find);
      if (hasil == -1) {
            // Jika kata tidak ditemukan
            main.setITEM_FOUND(-1);
      } else {
            // Jika kata ditemukan pada urutan ke-(hasil+1)
            time_akhir = System.nanoTime();
            main.setTime_awal(time_awal);
            main.setTime_akhir(time_akhir);
            main.setITEM_FOUND(1);
      }
}
```

3.15 fungsi *cari()*

Fungsi *cari()* akan memberikan nilai pada variabel objek *ITEM_FOUND* yang ada pada kelas *MainActivity.java* yang nanti akan diproses untuk pengambilan kata dari database jika variabel *ITEM_FOUND* bernilai satu. Selain memberikan nilai pada variabel objek *ITEM_FOUND*, fungsi *cari()* juga memberikan nilai pada variabel objek time_awal dan time_akhir yang berfungsi untuk mendapatkan waktu yang dibutuhkan dalam proses pencarian.

Perhitungan waktu yang dilakukan tiap-tiap algoritma dilakukan pada kelas interface *Kamus.java*, pada fungsi *setWaktu()* nilai time_akhir yang didapatkan ketika algoritma sedang dijalankan, akan dikurangi dengan time_awal, satuan waktu yang didapatkan dalam bentuk *nanosecond* kemudian dibagi 1000000 agar mendapatkan satuan *millisecond*. Gambar 3.16 menunjukkan potongan dari fungsi *setWaktu()* yang digunakan untuk mendapatkan waktu konsumsi.

```
public void setWaktu(long waktuAwal, long waktuAkhir) {
      float result = (float) (waktuAkhir - waktuAwal) /
1000000
      String res = String.valueOf(result).toString();
      String awal = String.valueOf(waktuAwal);
      String akhir = String.valueOf(waktuAkhir);
      this.waktu = res;
      setWaktuAwal(awal);
      setWaktuAkhir(akhir);
}
```

3.16 fungsi *setWaktu()*

3.7 Rancangan Tampilan Antarmuka

Pada perancangan tampilan ini akan ditampilkan rancangan *interface* dari awal eksekusi program hingga menampilkan hasil output yang dicari. Adapun rancangan tampilan pada aplikasi ini adalah sebagai berikut :

(a) (b)

Gambar 3.17 (a) Tampilan Pilih Algoritma,(b) Tampilan halaman pencarian

Pada gambar 3.17 merupakan desain tampilan *interface* dari aplikasi kamus kedokteran, untuk bagian (a) akan muncul ketika aplikasi pertama kali dijalankan, tampilan ini akan memberikan pilihan algoritma mana yang akan digunakan dalam proses pencarian, kemudian untuk gambar 3.17 bagian (b) merupakan gambar dari desain halaman utama yang akan digunakan untuk mencari kata istilah kedokteran.

(a) (b)

Gambar 3.18 (a) Tampilan hasil pencarian tanpa tab algoritma, (b) Tampilan hasil pencarian dengan tab algoritma

Desain *interface* berikutnya dapat dilihat pada gambar 3.18, untuk gambar (a) merupakan desain gambar dari hasil pencarian istilah beserta makna dari kata

yang dicari, dan juga terdapat waktu yang dibutuhkan dalam proses pencarian, kemudian untuk gambar 3.18 bagian (b) merupakan desain gambar dari hasil pencarian dengan istilah beserta makna dari kata yang dicari, dengan waktu yang dibutuhkan dalam proses pencarian, dan terdapat tab yang mewakili masing-masing algoritma, jika pada awal aplikasi memilih proses pencarian dua sekaligus.

(a) (b)

Gambar 3.19 (a) Desain hasil pencarian dengan algoritma *aho corasick,* (b) Desan tampilan dari arti per kata

Gambar 3.19 bagian (a) merupakan desain tampilan hasil pencarian yang menggunakan algoritma *aho corasick string matching,* jadi hasil kata yang

muncul bisa berjumlah lebih dari satu. Oleh karena itu, tampilan dari hasil pencarian tersebut dibuat seperti daftar kata. Kemudian pada gambar 3.19 bagian (b) merupakan desain tampilan arti dari istilah yang hasilnya didapatkan melalui pencarian menggunakan *aho corasick string matching.*

(a) (b)

Gambar 3.20 (a) Desain tampilan untuk menu pengaturan, (b) Desain tampilan untuk menu "daftar kata"

Pada halaman pengaturan terdapat beberapa fungsi yang terlihat seperti gambar 3.20 bagian (a) seperti menu pengaturan algoritma, dari halaman ini user

dapat mengatur algoritma mana yang ingin digunakan, bisa menggunakan salah satu atau keduanya. Pada halaman pengaturan juga terdapat kolom *Login to* yang berfungsi untuk memberikan hak akses pada user untuk masuk ke halaman manajemen kata. Pada gambar 3.20 bagian (b) merupakan desain tampilan dari menu "daftar kata", yang dapat memberikan informasi mengenai kata apa saja yang terdapat pada aplikasi.

(a) (b)

Gambar 3.21 (a) Tampilan dari halaman manajemen kata, (b) Desain tampilan dari form tambah data.

Desan tampilan dari halaman manajemen kata dapat dilihat pada gambar 3.21 bagian (a). Pada gambar tersebut terdapat menu edit dan hapus untuk per kata. Kemudian pada gambar 3.21 bagian (b) menampilkan form dari proses

tambah kata, dan untuk tombol simpan terdapat pada bagian pojok atas halaman. Untuk menuju halaman tambah data dapat melalui menu yang terdapat pada halaman utama manajemen data.

Desain tampilan dari form edit data dan proses hapus data dapat dilihat pada gambar 3.22 bagian (a) dan bagian (b).

(a) (b)

Gambar 3.22 (a) Tampilan form "Edit data", (b) Desain Tampilan untuk proses "hapus data"

Pada gambar 3.22 bagian (a) merupakan desain tampilan dari halaman edit data, desain tampilannya tidak jauh beda dengan desain tampilan form "tambah

data". Kemudian untuk gambar 3.22 bagian (b) merupakan desain tampilan dari proses menghapus data, tidak ada halaman tambahan untuk menghapus data, hanya menampilkan jendela konfirmasi untuk menghapus.

Kemudian melaui halaman manajemen kata, juga dapat melihat arti dari istilah yang ada. Desain tampilannya dapat dilihat pada gambar 3.23 bagian (a), kemudian pada gambar 3.23 bagian (b) merupakan tampilan dari halaman "Tentang".

(a) (b)

Gambar 3.23 (a) Tampilan untuk melihat arti melalui halaman manajemen kata, (b) Tampilan halaman tentang.

BAB IV

UJI COBA DAN PEMBAHASAN

Pada bab ini membahas tentang uji coba dan pembahasan dari hasil uji coba dari perancangan yang telah dibuat pada bab sebelumnya.

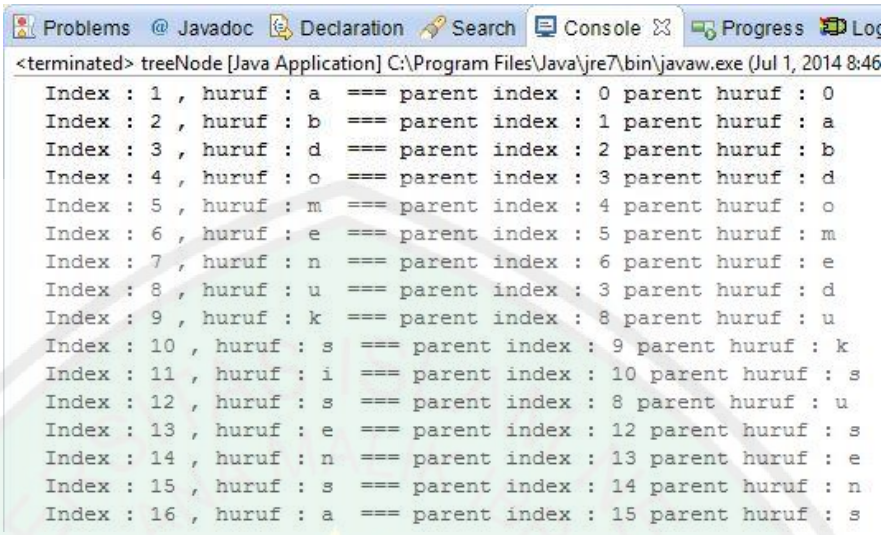
4.1 Proses Alur Pencarian

Pencarian pada aplikasi ini menggunakan dua metode, yaitu *Aho Corasick String Matching* dan *Interpolation Search.* Untuk membuktikan algoritma yang diimplementasikan pada aplikasi sesuai dengan alur dari algoritma, pembuktian akan dilakukan menggunakan platform IDE Eclipse dengan source kode yang sama dengan source kode yag telah diimplementasikan pada aplikasi.

4.1.1 *Aho Corasick String Matching*

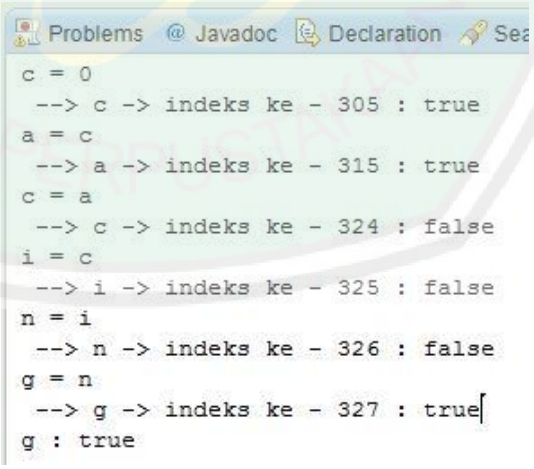
Proses dari *aho corasick* akan mengembalikan nilai *boolean*, jika terdapat kata yang cocok. Selebihnya akan dilakukan dengan fungsi sql. Kata yang akan ditelusuri adalah kata “cacing”, Berikut alur proses pencarian menggunakan *aho corasick string matching.*

1. Menambahkan kata-kata yang ada pada sebuah *tree*, dengan ketentuan per karakter mewakili satu state dan satu indeks, dan menginisialisasi nilai variabel boolean *leaf* sama dengan *false*. Jika karakter yang dimasukkan merupakan karakter terakhir dari suatu kata, maka variabel *leaf* akan bernilai *true*, begitu pun sebaliknya.



Gambar 4.1 indeks *tree* ketika selesai dibuat

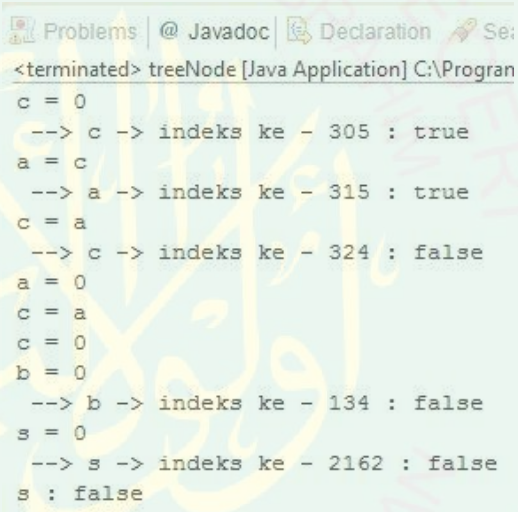
2. Setelah *tree* terbentuk, maka proses pencocokan kata dapat dilakukan. Kata "cacing" akan dilakukan pencocokan satu per satu dimulai dari huruf "c" dan diawali pada parent indeks nol. Perulangan akan dilakukan sepanjang jumlah karakter pada kata "cacing".



Gambar 4.2 Hasil perulangan dengan kata "cacing" yang bernilai *true*

3. Pada akhir perulangan akan dicek nilai *leaf* bernilai *false* atau *true*. Jika *true* maka kata yang dicari ada yang cocok dengan kata yang ada pada database.

4. Ketika *leaf* adalah *false*, maka fungsi *failure* akan dilakukan. Pencocokan karakter akan dikembalikan pada indeks nol, kemudian dari parent tersebut pencocokan akan dilakukan kembali. Jika tidak ada karakter yang cocok maka karakter tersebut akan dilewati dan dilanjutkan pencocokan karakter berikutnya. Pada akhir karakter dari kata yang dicari jika nilai *leaf* masih bernilai *false* maka kata yang dicari tidak ada yang cocok pada database.



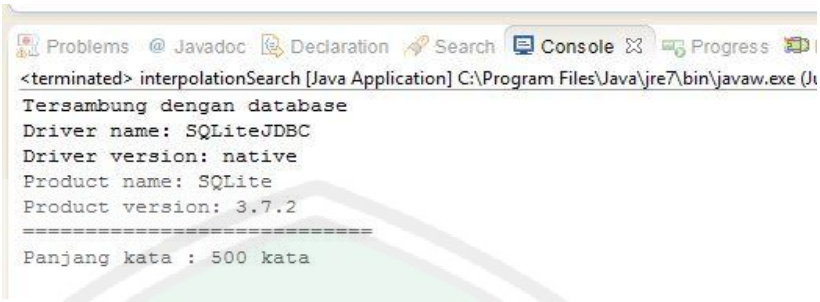
Gambar 4.3 Hasil perulangan dengan kata "cacbs" yang bernilai *false*

5. Ketika variabel *leaf* bernilai *true*, maka kata yang sedang dicari akan diambil artinya dari database menggunakan fungsi sql.

4.1.2 *Interpolation Search*

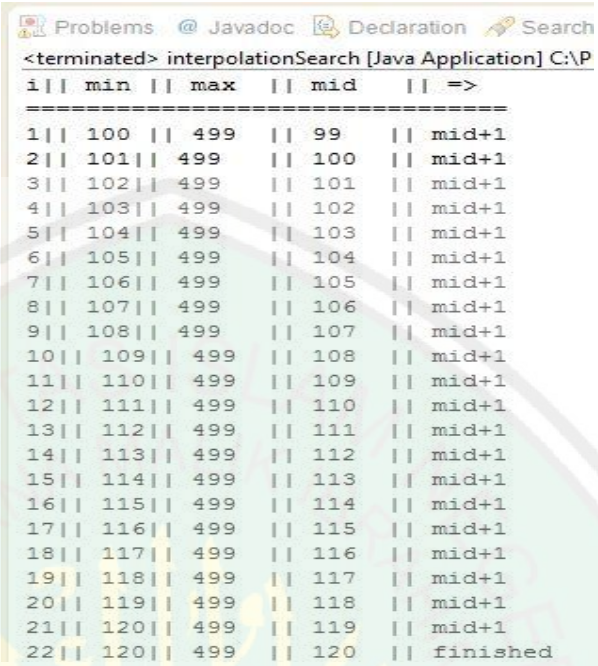
Pencarian yang akan ditelusuri yaitu kata "flu" yang terdapat pada array dengan total jumlah kata terdapat 500 kata.

1. Langkah pertama yang harus dilakukan adalah menyiapkan daftar kumpulan kata yang akan dicari yang diambil dari database kamus.



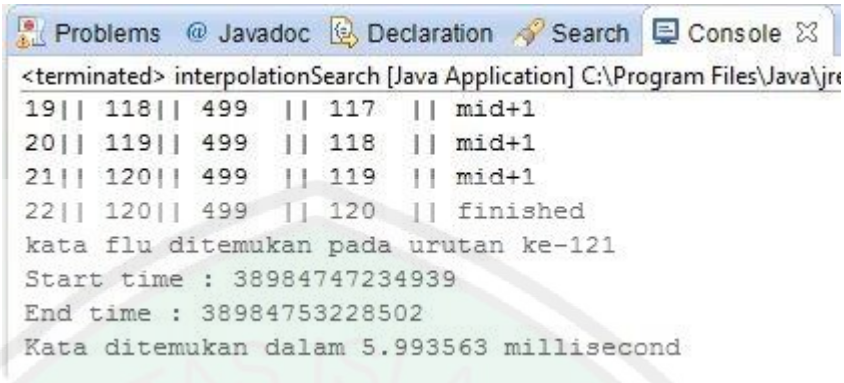
Gambar 4.4 Jumlah kata yang tersedia

2. Mendapatkan waktu saat itu dengan format *nanotime*, sebelum proses algoritma dimulai, dan disimpan dalam variabel sebagai waktu awal.
3. Mendeklarasikan nilai awalan atau prefik dari setiap kata yang ada, jika beberapa kata memiliki awalan yang sama maka nilai dari prefik akan sama.
4. Mendeklarasikan nilai awalan kata yang akan dicari, kata "flu" memiliki awalan huruf "f" maka akan memiliki nilai 16. Nilai tersebut didapatkan dari fungsi *prefik()* yang mengembalikan nilai *integer* dan telah dibahas pada subbab sebelumnya.
5. Mendeklarasikan variabel *low* = 0 (yang merupakan posisi awal dari pencarian), variabel *high* = panjang array dari kumpulan kata (yang akan menjadi posisi paling akhir), dan variabel *mid* yang akan diinisialisasi dengan rumus interpolasi.
6. Melakukan perulangan dengan kondisi nilai prefik kata yang dicari *x* lebih besar daripada nilai prefik *low*, dan *x* kurang dari nilai prefik *high*.



Gambar 4.5 Hasil perulangan untuk mendapatkan nilai *mid*

7. Proses perulangan akan berhenti jika kata yang dicari cocok dengan kata pada perulangan tersebut, pada kasus pencarian kata "flu" perulangan berhenti pada iterasi ke-22, kemudian buat variabel baru untuk mendapatkan waktu saat itu.
8. Setelah mendapatkan waktu akhir dari proses perulangan, maka waktu tersebut dikurangi dengan waktu awal, hingga akan mendapatkan waktu konsumsi yang digunakan dalam proses algoritma.



Gambar 4.6 Hasil pencarian kata "flu" beserta waktu konsumsi yang dibutuhkan

4.2 Proses Uji Coba

Dalam melakukan uji coba terhadap algoritma, terdapat beberapa langkah yang dapat dilakukan, yaitu:

1. Menyiapkan dua buah perangkat keras smartphone, yang memiliki spesifikasi berbeda.
2. Menginstall aplikasi kamus kedokteran yang telah terintegrasi dengan algoritma *aho corasick string matching* dan *interpolation search*.
3. Menyiapkan enam kata yang berbeda posisi (dalam indeks) dengan asumsi posisi yang dipilih adalah posisi awal, tengah, dan akhir.
4. Uji coba pertama dilakukan dengan tiga buah kata yang berbeda posisi, dan diujicobakan pada smartphone A, begitu pula dengan uji coba kedua, dilakukan dengan sisa dari kata dipilih, dan diujicobakan pada smartphone B.
5. Melakukan proses pencarian sebanyak sepuluh kali dengan kata yang sama, berlaku juga dengan kata yang lainnya.

6. Catat waktu yang dibutuhkan ketika proses pencarian oleh masing-masing algoritma pada tabel yang ada. Lakukan kegiatan tersebut pada proses ujicoba yang lain.
7. Setelah selesai, buat table baru yang berisi rata-rata dari setiap percobaan, kemudian hitung nilai varian.
8. Selesai.

4.3 Uji Coba Program

Pengujian perbandingan antara penerapan Algoritma *Aho Corasick String Matching* dengan Algoritma *Interpolation Search* akan dilakukan sebanyak dua kali dengan menggunakan perangkat smartphone yang memiliki spesifikasi yang berbeda, untuk mengetahui pengaruh dari spesifikasi hardware dengan kecepatan proses pencarian kata.

Pengujian dilakukan terhadap pencarian kata istilah kedokteran untuk mendapatkan konsumsi waktu yang dibutuhkan oleh algoritma dalam menemukan kata yang dicari, dan akan dilakukan beberapa kali pengujian untuk menentukan rata-rata waktu konsumsi.

4.3.1 Pengujian Pertama

Pengujian pertama menggunakan perangkat keras smartphone Huawei Ascend dengan spesifikasi sebagai berikut:

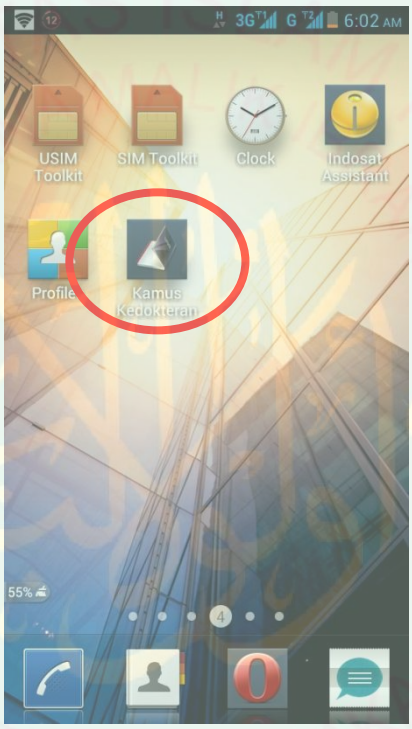
Prosesor : Quad Core 1.2 GHz

RAM : 1 GB

Memory Internal : 4 GB

Resolusi : 540 x 960 pixel

OS : Jelly Bean v4.1.2



Gambar 4.7 Aplikasi Kamus Kedokteran pada perangkat keras

Uji coba pencarian yang pertama dilakukan dengan kata *accoucheur* pada urutan ke-18 dari jumlah total 500 kata.

Tabel 4.1 Uji Ke-I: Pengujian dengan menggunakan kata *accoucheur* (dalam *millisecond*)

No.	*Aho Corasick String Matching*	*Interpolation Search*
1	0.053333	35.158337
2	0.061667	26.628332
3	0.058333	32.111668
4	0.055	25.368332

5	0.051667	19.95
6	0.058333	27.25
7	0.05	34.163334
8	0.085	26.303331
9	0.05	34.78
10	0.05	35.73833
Total	0.573333	297.451664

Untuk pengujian berikutnya akan dilakukan dengan menggunakan kata *makrofag* yang berada pada urutan ke-250.

Tabel 4.2 Uji Ke-II: Pengujian dengan menggunakan kata *makrofag* (dalam *millisecond*)

No.	***Aho Corasick String Matching***	***Interpolation Search***
1	0.068333	59.74167
2	0.05	60.02667
3	0.058333	28.43
4	0.06	41.39
5	0.06	32.68
6	0.061667	20.968332
7	0.071667	20.796667
8	0.06	50.338337
9	0.061667	26.023333
10	0.116667	28.853333
Total	0.668334	369.248342

Kemudian pengujian selanjutnya akan dilakukan dengan menggunakan kata *zonipetal* yang terdapat pada urutan ke-490.

Tabel 4.3 Uji Ke-III: Pengujian dengan menggunakan kata *zonipetal* (dalam *millisecond*)

No.	*Aho Corasick String Matching*	*Interpolation Search*
1	0.08	31.0
2	0.07	34.575
3	0.066666	75.745
4	0.066667	32.38
5	2.41	21.111668
6	0.068333	66.333336
7	0.068333	33.998333
8	0.066667	40.861668
9	0.068334	37.551666
10	0.07	29.528334
Total	3.035	403.085005

4.3.2 Pengujian Kedua

Kemudian untuk pengujian kedua akan dilakukan dengan menggunakan perangkat smartphone Sony Xperia Miro dengan spesifikasi sebagai berikut:

Processor : 800 Mhz Cortex-A5

RAM : 512 MB

Memori Internal : 4 GB

Resolusi : 320 x 480 pixels

OS : Ice Cream Sandwich v4.0

Uji coba pencarian yang keempat dilakukan dengan kata "bakteri" pada urutan ke-23 dari jumlah total 500 kata.

Tabel 4.4 Uji ke-IV: Pengujian dengan menggunakan kata "bakteri" (dalam *millisecond*)

No.	*Aho Corasick String Matching*	*Interpolation Search*
1	0.063333	91.91334
2	0.065	87.97666
3	0.068333	94.26666
4	0.056666	92.275
5	0.06	93.04666
6	0.185	21.98
7	0.058333	89.07166
8	0.061666	21.308332
9	0.056667	21.746666
10	0.066667	22.198332
Total	0.741665	635.78331

Kemudian uji coba kelima dengan menggunakan kata "oksigen" yang terdapat pada urutan ke-293 dari jumlah total 500 kata.

Tabel 4.5 Uji ke-V: Pengujian dengan menggunakan kata "oksigen"(dalam *millisecond*)

No.	*Aho Corasick String Matching*	*Interpolation Search*
1	0.093333	21.435001
2	0.098333	20.965
3	0.081667	22.008333
4	0.088334	21.906668
5	0.088334	21.618332
6	0.095	21.891668
7	0.098333	22.083332
8	0.086667	24.746668
9	0.085	21.723331
10	0.095	24.888332
Total	0.910001	223.266665

Untuk uji coba yang keenam akan menggunakan kata *yaws* yang terdapat pada urutan ke-477 dari total 500 kata.

Tabel 4.6 Uji ke-VI: Pengujian dengan menggunakan kata "Yaws"(dalam *millisecond*)

No.	*Aho Corasick String Matching*	*Interpolation Search*
1	0.088333	23.528334
2	0.078334	20.5005
3	0.098333	20.645
4	0.076666	20.318333
5	0.101667	21.045
6	0.09	21.215
7	0.08	21.315
8	0.88334	37.673336
9	0.176667	24.383331
10	0.08	23.11
Total	1.75334	233.733834

Selanjutnya hasil rata-rata dari masing-masing pengujian, dengan mengambil total data konsumsi waktu, kemudian dibagi dengan jumlah pengujian pada tiap-tiap pengujian.

Tabel 4.7 Tabel rata-rata dari uji coba dengan perangkat smartphone yang pertama

Uji Ke -	$\bar{X}$ *Aho Corasick String Matching*	Varian	$\bar{X}$ *Interpolation Search*	Varian
I	0.0573333	0.0001112346	29.7451664	28.7310241284
II	0.0668334	0.0003397878	36.9248342	227.0070001597
III	0.3035	0.547835463	40.3085005	294.2082890943
Total	**0.16721739**		**35.6595003**	

Tabel 4.8 Tabel rata-rata dari uji coba dengan perangkat smartphone yang kedua

Uji Ke -	$\bar{X}$ *Aho Corasick String Matching*	Varian	$\bar{X}$ *Interpolation Search*	Varian
IV	0.0741665	0.0015328754	63.578331	1295.6758056047
V	0.0910001	0.0000334537	22.3266665	1.8288513595
VI	0.175334	0.0627558707	23.3733834	27.6855133061
Total	**0.1135002**		**36.426127**	

4.4 Hasil Uji Coba

Setelah melakukan uji coba pada subbab sebelumnya, dapat dilihat pada tabel 4.7 dan tabel 4.8 merupakan kumpulan rata-rata dari masing-masing uji coba yang total keseluruhan ada enam kali percobaan. Jika dilihat lebih teliti, pada uji coba ke-I dan uji coba ke-IV, rata-rata konsumsi waktu yang didapatkan adalah rata-rata yang paling rendah dari rata-rata uji coba yang lainnya. Hal ini menunjukkan bahwa dengan menggunakan algoritma *aho corasick string matching*, posisi indeks kata yang dicari cukup berpengaruh dalam mengambil konsumsi waktu ketika proses pencarian.

Kemudian untuk proses pencarian menggunakan algoritma *interpolation search*, posisi indeks kata terhadap kumpulan array kata, tidak terlalu berpengaruh terhadap konsumsi waktu. Hal ini dapat dilihat pada uji coba ke-IV, yang menggunakan kata yang memiliki indeks kecil, tetapi konsumsi waktu yang dibutuhkan lebih besar dari yang lainnya.

Pada kolom varian pada masing-masing uji coba dapat dilihat, untuk algoritma *aho corasick string matching* rata-rata untuk nilai variannya mendekati angka nol, hal ini berarti selisih konsumsi waktu dari setiap pencarian tidak jauh

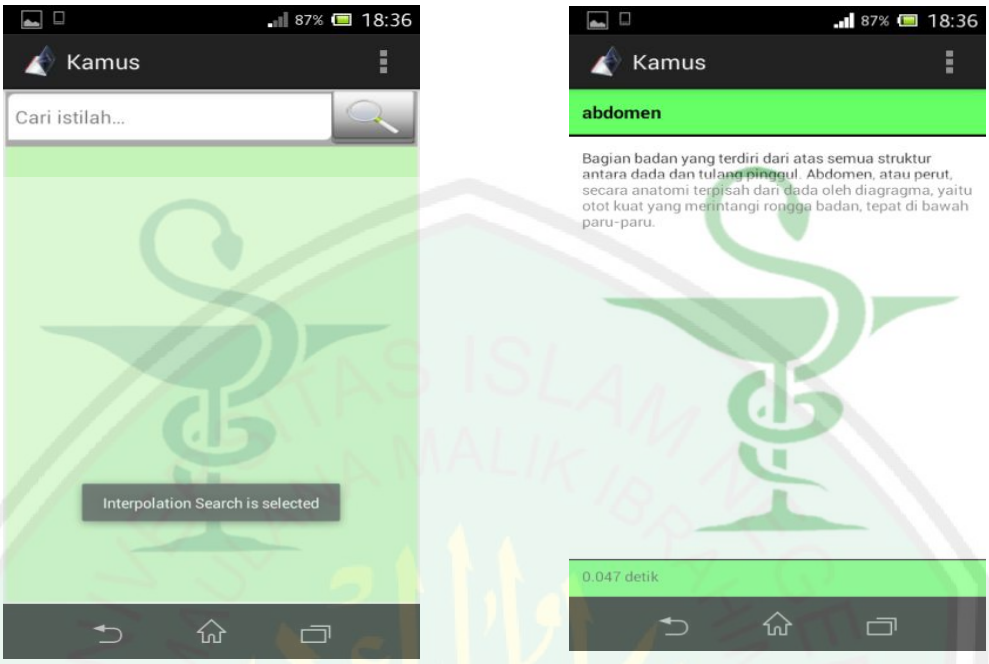
berbeda, berbeda dengan nilai varian dari pencarian yang menggunakan algoritma *interpolation search*, yang memiliki nilai cukup besar, berarti konsumsi waktu yang dibutuhkan dari setiap pencarian memiliki selisih yang besar.

4.5 Implementasi Aplikasi

Berikut ini hasil implementasi aplikasi Kamus Kedokteran:

Gambar 4.8 Halaman Pilih Algoritma

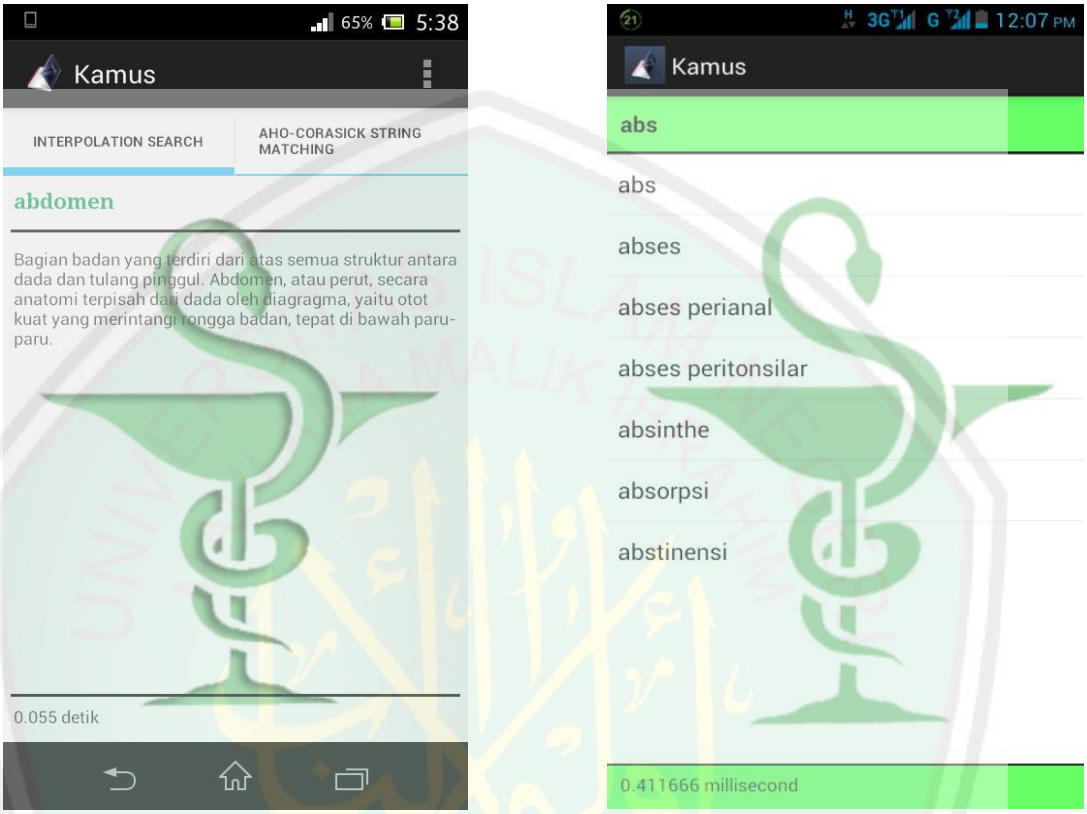
Dalam halaman awal terdapat pilihan algoritma seperti pada perancangan sebelumnya. Pilihan algoritma ini digunakan untuk memilih algoritma apa yang akan dipakai dalam pencarian kata istilah kedokteran dalam kamus kedokteran. Setelah algoritma telah dipilih, halaman berikutnya akan ditampilkan halaman input kata seperti gambar 4.8.



(a) (b)

Gambar 4.9 (a) Halaman Input Kata, (b) Halaman Arti kata tanpa tab algoritma

Pada halaman input kata seperti gambar 4.9 bagian (a), user memasukkan kata yang akan dicari dan menekan tombol search. Setelah tombol search ditekan, halaman berikutnya yang akan ditampilkan adalah halaman arti kata tanpa tab algoritma seperti gambar 4.9 bagian (b), dan dengan tab algoritma seperti gambar 4.10 pada bagian (a).

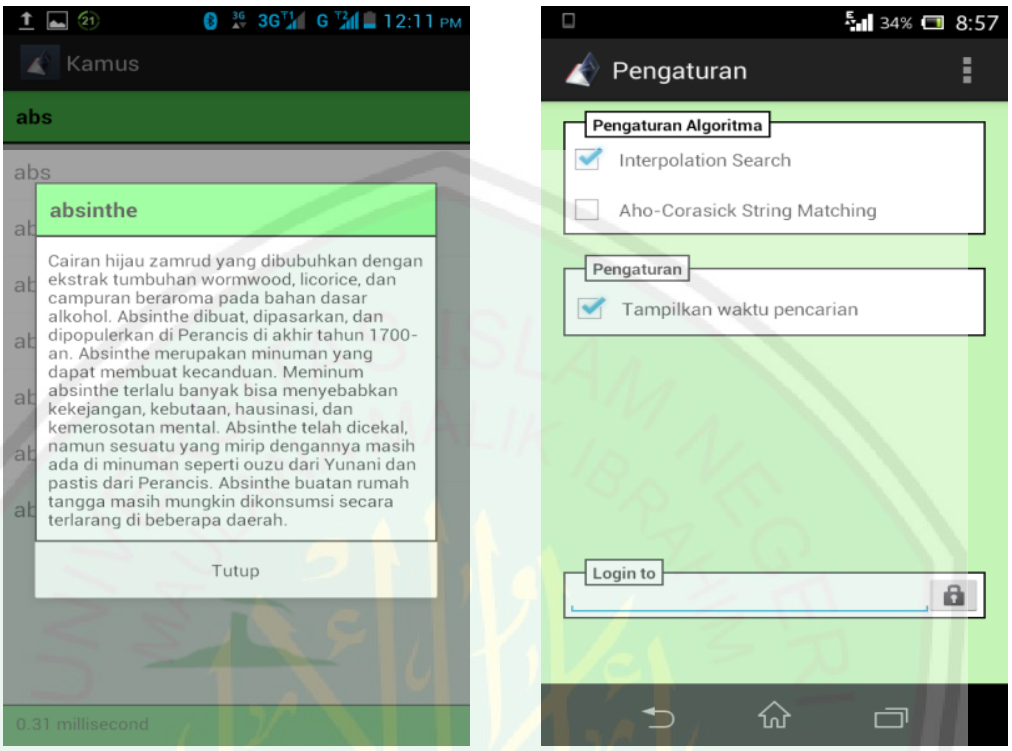


(a) (b)

Gambar 4.10 (a) Halaman arti dengan tab algoritma, (b) Halaman daftar kata menggunakan algoritma *aho corasick*

Selain itu, jika dipilih algoritma *Aho Corasick String Matching* maka akan ditampilkan juga halaman list kata hasil pencarian menggunakan algoritma *aho corasick string matching*, sebagaimana gambar 4.10 pada bagian (b).

Setelah memilih kata yang akan dilihat artinya, maka akan ditampilkan halaman arti list kata hasil pencarian menggunakan *aho corasick string matching* akan tampak seperti pada gambar 4.11 bagian (a).

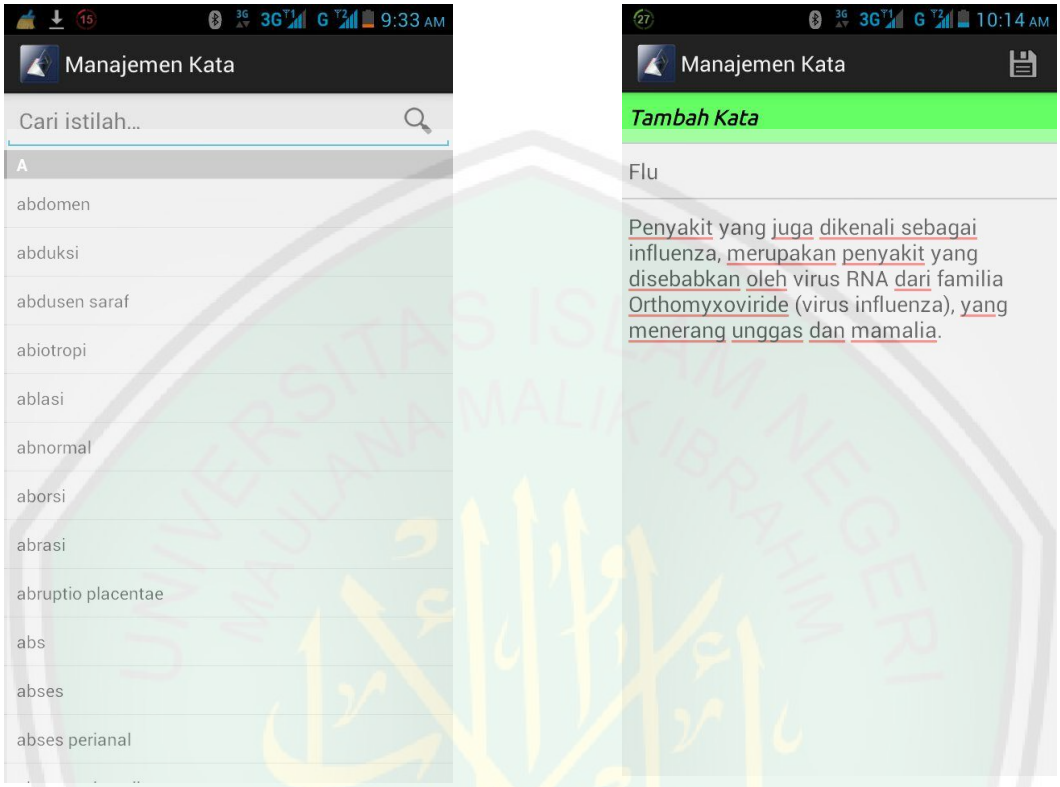


(a) (b)

Gambar 4.11 (a) Halaman arti daftar kata menggunakan *Aho Corasick*,(b) Halaman Pengaturan

Di dalam kamus ini juga terdapat menu pengaturan yang dapat digunakan oleh user untuk mengatur algoritma, serta form inputan password untuk masuk ke halaman manajemen kata. Menu pengaturan ditampilkan seperti gambar 4.11 pada bagian (b).

Pada menu pengaturan terdapat *login to* yang berfungsi untuk membawa user masuk kedalam halaman manajemen kata. Setelah user berhasil login, maka akan ditampilkan halaman manajemen kata sebagai berikut:

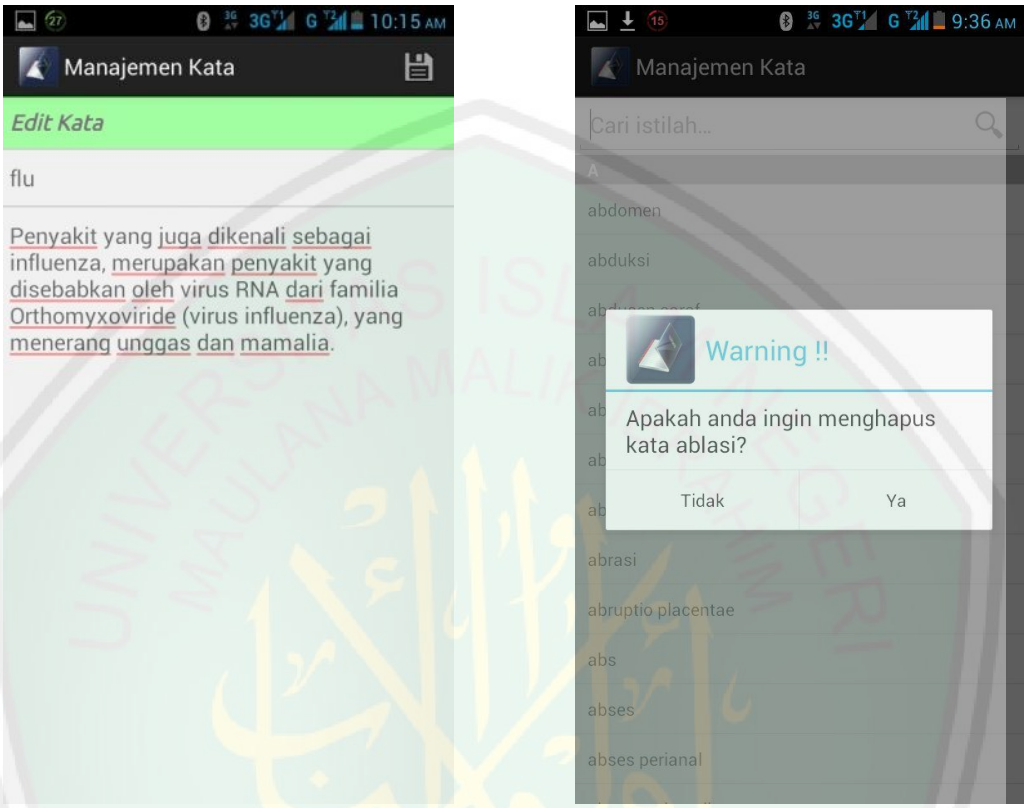


(a) (b)

Gambar 4.12 (a) Halaman manajemen kata, (b) Halaman tambah kata

Pada halaman manajemen kata ini, terdapat beberapa aksi yang dapat digunakan oleh user. Di antaranya adalah ada aksi tambah kata, edit serta hapus. Pada halaman tambah kata user dapat menambahkan kata pada aplikasi kamus. Halaman tambah kata akan dapat dilihat seperti gambar 4.12 bagian (b).

Selain halaman tambah kata, terdapat halaman edit yang dapat digunakan oleh user untuk memperbaiki kata serta penjelasan yang dirasa kurang pas pada aplikasi kamus kedokteran. Halaman edit kata pada halaman manajemen kata akan terlihat seperti gambar 4.13 bagian (a).



(a) (b)

Gambar 4.13 (a) Halaman edit kata, (b) Tampilan untuk menhapus kata

Aksi terakhir pada halaman manajemen kata adalah hapus. Pada halaman hapus ini, user dapat menghapus kata pada aplikasi kamus kedokteran. Halaman hapus kata ditampilkan seperti gambar 4.13 pada bagian (b).

Selain dapat menambah, mengedit serta menghapus kata, user juga dapat melihat list arti kata pada halaman manajemen kata dengan menekan salah satu kata yang ada pada halaman manajemen kata. Halaman arti list kata pada halaman manajemen kata dapat dilihat seperti gambar 4.14.

Gambar 4.14 Halaman arti daftar kata pada manajeme kata

4.6 Integrasi Studi Perbandingan Algoritma Pencarian Dengan Islam

Penelitian ini menitikberatkan pada proses perbandingan antara algoritma *Aho-Corasick String Matching* dan *Interpolatin Search.* Pada penelitian ini juga dilakukan proses pengamatan terhadap masing-masing algoritma, dengan tujuan mendapatkan hasil sesuai dengan tujuan penelitian ini.

Dalam islam proses pengamatan terhadap sesuatu telah dibenarkan dan pernah dilakukan oleh nabi Ibrahim AS, pada waktu itu beliau mencari kebenaran mengenai tuhanNya, hal ini telah tertuang pada ayat al-qur'an surat al-an'am ayat ke 74-78, berikut kutipan ayat tersebut.

۞ وَإِذْ قَالَ إِبْرَٰهِيمُ لِأَبِيهِ ءَازَرَ أَتَتَّخِذُ أَصْنَامًا ءَالِهَةً ۖ إِنِّىٓ أَرَىٰكَ وَقَوْمَكَ فِى ضَلَٰلٍ
مُّبِينٍ ﴿٧٤﴾ وَكَذَٰلِكَ نُرِىٓ إِبْرَٰهِيمَ مَلَكُوتَ ٱلسَّمَٰوَٰتِ وَٱلْأَرْضِ وَلِيَكُونَ مِنَ
ٱلْمُوقِنِينَ ﴿٧٥﴾ فَلَمَّا جَنَّ عَلَيْهِ ٱلَّيْلُ رَءَا كَوْكَبًا ۖ قَالَ هَٰذَا رَبِّى ۖ فَلَمَّآ أَفَلَ قَالَ لَآ
أُحِبُّ ٱلْءَافِلِينَ ﴿٧٦﴾ فَلَمَّا رَءَا ٱلْقَمَرَ بَازِغًا قَالَ هَٰذَا رَبِّى ۖ فَلَمَّآ أَفَلَ قَالَ لَئِن لَّمْ
يَهْدِنِى رَبِّى لَأَكُونَنَّ مِنَ ٱلْقَوْمِ ٱلضَّآلِّينَ ﴿٧٧﴾ فَلَمَّا رَءَا ٱلشَّمْسَ بَازِغَةً قَالَ
هَٰذَا رَبِّى هَٰذَآ أَكْبَرُ ۖ فَلَمَّآ أَفَلَتْ قَالَ يَٰقَوْمِ إِنِّى بَرِىٓءٌ مِّمَّا تُشْرِكُونَ ﴿٧٨﴾

Artinya:

"Dan (ingatlah) di waktu Ibrahim berkata kepada bapaknya, Aazar, "Pantaskah kamu menjadikan berhala-berhala sebagai tuhan-tuhan? Sesungguhnya aku melihat kamu dan kaummu dalam kesesatan yang nyata(74) Dan demikianlah Kami perlihatkan kepada Ibrahim tanda-tanda keagungan (Kami yang terdapat) di langit dan bumi dan (Kami memperlihatkannya) agar dia termasuk orang yang yakin(75) Ketika malam telah gelap, dia melihat sebuah bintang (lalu) dia berkata: "Inilah Tuhanku", tetapi tatkala bintang itu tenggelam dia berkata: "Saya tidak suka kepada yang tenggelam(76) Kemudian tatkala dia melihat bulan terbit dia berkata: "Inilah Tuhanku". Tetapi setelah bulan itu terbenam, dia berkata: "Sesungguhnya jika Tuhanku tidak memberi petunjuk kepadaku, pastilah aku termasuk orang yang sesat(77) Kemudian tatkala ia melihat matahari terbit, dia berkata: "Inilah Tuhanku, ini yang lebih besar". Maka tatkala matahari itu terbenam, dia berkata: "Hai kaumku, sesungguhnya aku berlepas diri dari apa yang kamu persekutukan(78)"

Pada kitab tafsir al-qur'an Al-aisaar, menerangkan bahwa ketika *nabi Ibrahim mencari kebenaran mengenai tuhannya dimulai ketika beliau melihat bintang yang bersinar, kemudian setelah bintang tersebut tenggelam, beliau melihat bulan yang sinarnya lebih terang daripada bintang, dan berkata "inilah tuhanku", namun ketika bulang tersebut tenggelam, beliau berkata "Jika Rabbku tidak memberikan petunjuk, maka pastilah aku termasuk orang-orang yang sesat". Kemudian beliau melihat matahari terbit, yang sinarnya lebih terang dari keduanya (bintang dan bulan), namun ketika matahari tenggelam dengan*

datangnya malam, beliau berkata "Sesungguhnya aku berlepas diri dari apa yang kamu persekutukan".

Pengamatan yang dilakukan oleh nabi Ibrahim, sama halnya dengan melakukan perbandingan antara bintang, bulan, dan matahari, kemudian nabi Ibrahim mendapatkan suatu kesimpulan dari proses pengamatan tersebut. Tidak jauh berbeda dengan proses studi perbandingan algoritma pencarian antara *aho corasick string matching* dengan *interpolation search* yang bertujuan untuk mendapatkan kesimpulan tentang algoritma mana yang paling baik dalam proses pencarian.

BAB V

PENUTUP

5.1 Kesimpulan

Dari hasil implementasi dan uji coba yang telah peneliti lakukan dapat diambil kesimpulan

1. Rata-rata konsumsi waktu yang dibutuhkan algoritma *Aho Corasick String Matching* lebih sedikit daripada algoritma *Interpolation Search.*
2. Beda waktu yang dibutuhkan oleh masing-masing algoritma dapat dilihat pada selisih total rata-rata pada setiap algoritma yaitu 35.495778561 milliseconds (pengujian pertama) dan 36.3126268 milliseconds (pengujian kedua).

5.2 Saran

Tentunya masih banyak kekurangan dalam penelitian studi perbandingan yang diimplementasikan pada aplikasi kamus kedokteran ini. Oleh karena itu, penulis menyarankan beberapa hal untuk bahan pengembangan selanjutnya, diantaranya:

1. Melakukan perbandingan lebih jauh lagi, tidak hanya berdasarkan konsumsi waktu saja, tapi bisa dengan konsumsi memori yang dibutuhkan.
2. Dapat menambahkan algoritma yang lain jika diperlukan, dan tidak hanya terbatas pada dua algoritma saja.

3. Mengembangkan aplikasi kamus kedokteran dengan lebih baik, dengan menambahkan gambar pada kata yang mungkin memiliki penjelasan dengan menyertakan gambarnya.

DAFTAR PUSTAKA

Al Barry, M. Dahlan Yacub dan Partanto, Pius A. 2000. *Kamus Ilmiah Populer*. Surabaya: Arkola.

Al-Jazairy, Syaikh Abu Bakar Jabir. 2007. *Tafsir Al-Qur'an Al-Aisaar (Jilid 2)*. Jatinegara – Jakarta Timur: Darus Sunnah Press.

Aho, Alfred V. and Margaret J. Corasick. 1975, *Efficient String Matching: An Aid to Bibliographic Search.* Bell Laboratories.

Azmi, Yan. 2009. Perangkat Lunak Aplikasi. Yogyakarta: Penerbit Andi.

Chapra, Steven C. 2012. *Applied Numerical Methods With Matlab For Engineers And Scientists: Third Edition*, USA: Mc Graw Hill.

Crous, Carl. 2006. *Dictionary Matching Automata: The Aho-Corasick Algorithm.* BScHons

Gonnet, Gaston H, Rogers, Lawrence D, dan Goerge, J Alan. 1979. *An Algorithmic and Complexity Analysis of Interpolation Search*.University of Waterloo: Canada

Hasib, Saima. Motwani, Mahak. Saxena, Amit. 2013. *Importance of Aho-Corasick String Matching Algorithm in Real World Applications*. Truba Institute of Engineering and Information Technology Bhopa: India.

Isnani, Martina Husnul. 2010. *Aplikasi Online Kamus Kedokteran Dengan Menggunakan Metode Binary Search.* Jurusan Teknik Informatika. Fakultas Sains dan Teknologi. UIN MAulana Malik Ibrahim Malang.

Kadir, Abdul.2005, *Dasar Pemoraman Java 2.* Yogyakarta: Penerbit Andi.

Kilpelainen, Pekka. 2005. *Set Matching and Aho-Corasick Algorithm.* University Kuopio. Department of Computer Science.

Kukreja, Kartik. 2013. *Beating Binary Search: The Interpolation Search.* http://kartikkukreja.wordpress.com/2013/08/17/beating-binary-search-the-interpolation-search/, (diakses pada tanggal 02 Juni 2014)

Ngoen, Thompson Subsada. 2011, *Algoritma Dann Struktur Data Pengurutan Dan Pencarian.* Jakarta: Mitra Wacana Media.

Purwanto, Budi Eko. 2008. *Perancangan Dan Analisa Algoritma.* Yogyakarta. Graha Ilmu.

Safaat, Nazaruddin. 2011. *Pemograman Aplikasi Mobile Smartphone dan Tablet PC Berbasis Android.* Bandung: Informatika.

Sholiq. 2006. *Analisis dan Perancangan Berorientasi Obyek.* Bandung : CV. Muara Indah.

Sholiq.2006. *Pemodelan Sistem Informasi Berorientasi Obyek dengan UML.* Yogyakarta: Graha Ilmu.

Tim Dokter WebMD. 2008. *KAMUS KEDOKTERAN WEBSTER'S NEW WORLD.* Jakarta Barat: PT. Indeks Permata Puri Media.

Tim Penyusun Kamus Pusat Pembinaan dan Pengembangan Bahasa. 1989. *Kamus Besar Bahasa Indonesia Edisi Kedua.* Balai Pustaka.

Umar, Dr. Ahmad Mukhtar. 1998, *Shina'atul Ma'ajim Al-Hadist.* Cairo:'Alam Al-Kutub

Yuswanto. 2009. *Algoritma dam Pemrograman Dengan Visual Basic.Net,* Jakarta: Cerdas Pustaka Publisher.

http://en.wikipedia.org/wiki/Aho-Corasick_string_matching_algorithm (diakses pada tanggal 13 Februari 2014).

http://blog.ivank.net/aho-corasick-algorithm-in-as3.html, (diakses pada tanggal 10 Maret 2014)

LAMPIRAN

1	abdomen
2	abduksi
3	abdusen saraf
4	abiotropi
5	ablasi
6	abnormal
7	aborsi
8	abrasi
9	abruptio placentae
10	abs
11	abses
12	abses perianal
13	abses peritonsilar
14	absinthe
15	absorpsi
16	abstinensi
17	acanthosis nigricans
18	accoucheur
19	acetaminophen
20	ACOG
21	babinski refleks
22	bag of waters
23	baker,kista
24	bakteremia
25	bakteri
26	bakterial
27	bakteriofag
28	bakteriosidal
29	bakteriostastis
30	bakterium
31	balanitis
32	balon,angiplasti
33	banjir,sarana perlengkapan
34	bantu,alat
35	barbiturat
36	bariatrik,dokter
37	barium enema
38	barium sulfat
39	barium,larutan
40	bola-dan-soket,sendi
41	C 1
42	C1 sampai C7
43	CA 125
44	Ca19-9
45	CABG
46	cacing cambuk
47	cacing pita
48	cacing pita babi
49	CAD
50	caduceus
51	caffey disease
52	campak
53	campak atipikal
54	cancer survivor
55	candida albicans
56	cannabis
57	cardioulmonary resuscitation
58	carpal tunnel
59	carrier tes
60	cast
61	dada
62	dada,rasa sakit
63	dada,sinar-X
64	Daily Prayer of a Physicia
65	dakrokista
66	daktil
67	dander
68	darah
69	debilitas
70	debride
71	defibrilasi
72	defibrilator
73	deglutisi
74	dehidrasi
75	dehisce
76	dejavu
77	dekstrokardia
78	dekstroposisi
79	dekstrosa
80	delirium
81	E.coli
82	ear tag
83	Ebola virus
84	echovirus
85	ectopia cordis
86	edema
87	edema periorbital
88	edentulous
89	efasmen
90	efedrin
91	eferen
92	efusi
93	efusi perikardum
94	ehrlichiosis
95	ejakulasi
96	ekimosis
97	eklamsia
98	ekolalia
99	ekomotis
100	ekopraksia
101	factor VIII
102	fagosit
103	fakoemulsifikasi
104	falang
105	false negative
106	false positif
107	familial
108	faring
109	faringitis
110	farmakogenetik
111	farmakolog
112	farmakologi
113	farmakope
114	fasia
115	fasikulasi
116	fava bean
117	fecalith
118	feedback

LAMPIRAN

119	femur
120	fenestrasi
121	flu
122	G6PD
123	gait
124	galaktosa
125	galium
126	gamet
127	ganglion
128	gastrektomi
129	gastrik
130	gastritis
131	gastroenteritis
132	gastroenterolog
133	gastroskop
134	gatal
135	gen
136	genital
137	genitalia
138	genom
139	genu
140	gigantisme
141	ginekolog
142	H and H
143	hairball
144	halitosis
145	halusinogen
146	hamartoma
147	hammer toe
148	hamstring
149	hantavirus
150	haploid
151	haplotipe
152	havrix
153	heberden's disease
154	heksadaktili
155	hemangioma
156	hemangioma kavernosa
157	hematolog
158	hematologi

159	hematoma
160	hematoma intraserebral
161	hematrosis
162	iatrapistik
163	iatrogenik
164	iatromelia
165	iatromisia
166	idiopatik
167	iktiosis vulgaris
168	ilium
169	immune
170	impaksi
171	imunisasi
172	imunitas
173	imunodefisien
174	imunokompeten
175	infan
176	infark
177	infertil
178	infiltrasi
179	inflamasi
180	inheritance
181	insersi
182	jamais vu
183	jangkauan
184	jangkauan gerakan
185	jantung artifisial
186	jaringan
187	jaundice
188	jaundice hemolitik
189	jaundice hepatoselular
190	jaundice obstruktif
191	JC virus
192	jejunostomi
193	jejunum
194	jerawat
195	jet lag
196	jinak
197	jock itch
198	jukstaartikular

199	jukstapiloris
200	jukstaspinal
201	juvenil
202	kafein
203	kakeksia
204	kalamin
205	kalium
206	kalkaneus
207	kalkulus
208	kalor
209	kalori
210	kalsifikasi
211	kalsinosis
212	kalsium
213	kanker
214	kantung Douglas
215	kapiler
216	kapitasi
217	karantina
218	karbunkel
219	kardiak
220	kardiolog
221	kardiomiopati
222	labia
223	labia mayora
224	labia minora
225	labia oral
226	labia vagina
227	labil
228	labirin
229	labirintitis
230	laboratorium
231	Lactobacillus
232	lakrimal
233	lakrimasi
234	laksatif
235	laktase
236	lakuna
237	lama hidup
238	lambung

LAMPIRAN

239	lamela
240	lamina
241	laminaria
242	magnetic resonance imaging
243	magnetisme
244	major hystocompatibility complex
245	makanan
246	makanan fungsional
247	makroangiopati
248	makrobiota
249	makrobiotik
250	makrofag
251	makroglobulinemia
252	makroglosia
253	makrognatia
254	makrosefali
255	makrositik
256	makroskopis
257	makrosomia
258	maksila
259	makula
260	makula lutea
261	makular
262	nadir
263	narkotik
264	nasal
265	nasofaring
266	nasogastris
267	natrium
268	natriuresis
269	natriuretik
270	naturopati
271	nausea
272	nebulisasi
273	nebuliser
274	nefrektomi
275	nefritis
276	nefrolit

277	nefrologi
278	nefron
279	nekrosis
280	neonatal
281	neonatus
282	obesitas
283	obesitas ginekoid
284	obstetri
285	obtunded
286	occupational therapist
287	oftalmia
288	oftalmik
289	oftalmolog
290	oftalmologi
291	oftalmoskop
292	oklud
293	oksigen
294	oksigenasi
295	oksimetolone
296	oksimetri
297	oksiput
298	oksitosin
299	okuloplasti
300	okulta
301	olekranon
302	p53
303	pacemaker
304	pachyonychia congenita
305	paha
306	paliatum
307	palpabel
308	palpasi
309	palpebra
310	palpitasi
311	panasea
312	pandemik
313	pandikulasi
314	panik
315	panikulitis
316	papiloma

317	papula
318	parasit
319	parasomnia
320	parenteral
321	paronikia
322	q fever
323	QRS complex
324	quack
325	quackery
326	quickening
327	quiescent
328	quinacrine
329	quinidine
330	quinine
331	quotidian
332	quotient
333	rabdomiolisis
334	rabdomiosarkoma
335	rabun dekat
336	racemose
337	racun
338	rad
339	radial
340	radiasi
341	radikulopati
342	radioaktif
343	radiograf
344	radiografi
345	radioimmunoassay
346	radioinsensitif
347	radioisotop
348	radiolog
349	radiolog intervensi
350	radiologi
351	radiologis
352	radiolusen
353	random
354	rangkaian
355	saccade
356	saccharide

LAMPIRAN

357	saccharin
358	saccharolytic
359	saccharometabolism
360	Saccharomyces
361	saccharopine
362	saccharopinuria
363	sacciform
364	saccular
365	saccule
366	sacculocochlear
367	sacculotomy
368	sacralization
369	sacroiliac
370	sadism
371	salicylic acid
372	salicylism
373	salicyluric acid
374	sallicymlamide
375	tabanid
376	Tabanus
377	tabescent
378	tablature
379	tablet
380	taboparesis
381	tachography
382	tachyarrhythmia
383	tachydysrhythmia
384	tachygastria
385	tachykinin
386	tachyphylaxis
387	tachypnea
388	tachysterol
389	tacrine
390	tactometer
391	talc
392	tambour
393	tamoxifen
394	tannic acid
395	ubiquinol
396	ubiquinone
397	ulceration
398	ulcerogenic
399	ulectomy
400	ulerthema
401	ulnar
402	ulorrhagia
403	ultasonic
404	ultracentrifugation
405	ultradian
406	ultrafiltration
407	ultramicroscope
408	ultrasonic
409	ultrasound
410	ultrastructur
411	uncipressure
412	undecylenic acid
413	undersensing
414	union
415	VAC
416	vaccinal
417	vaccination
418	vacuolation
419	vacuole
420	vagina
421	vaginismus
422	vagolytic
423	vagomimetic
424	vagotonia
425	valine
426	valproic acid
427	valvulopplasty
428	valvulotome
429	vancomcycin
430	vanillism
431	vanillymandelic acid
432	varicellavirus
433	varicoblepharon
434	varicocele
435	waist
436	walker
437	ward
438	warfarin
439	wash
440	waters
441	waxing
442	wean
443	weber
444	wen
445	wheal
446	wheeze
447	whitehead
448	WHO
449	whoop
450	winking
451	withdrawal
452	wohlfahrtia
453	wrist
454	wristdrop
455	xanthelasme
456	xanthic
457	xanthine
458	xanthinuria
459	xanthochromia
460	xanthochromic
461	xanthocyanopsia
462	xanthoderma
463	xanthomatosis
464	xanthopsia
465	xanthosis
466	xanthurenic acid
467	xenegenous
468	xenogeneic
469	xenogenesis
470	xenograft
471	xenoparasite
472	xenophobia
473	xenophonia
474	xenophthalmia
475	yabapox
476	yatapoxvirus

477	yaws
478	yellow
479	yoke
480	yolk
481	zarontin
482	zaroxolyn
483	zero
484	zidovudine
485	zigzaplasty
486	zoacanthosis
487	zoanthropy
488	zonesthesia
489	zonifugal
490	zonipetal
491	zonule
492	zonulitis
493	zonulolysis
494	zoodermic
495	zoografting
496	zooid
497	zoolagnia
498	zoology
499	zoomastigophorea
500	zoonosis