ISSN 2085-4552

Implementasi Algoritma Boyer-Moore pada Aplikasi Kamus Kedokteran Berbasis Android

Kencana Wulan Argakusumah, Seng Hansun
Program Studi Teknik Informatika, Universitas Multimedia Nusantara, Tangerang, Indonesia
kencanawulan2137@yahoo.com, hansun@umn.ac.id

Diterima 23 Oktober 2014
Disetujui 9 Desember 2014

Abstract*—Dictionary is a reference book that contains words and phrases that are usually arranged alphabetically and followed by description of the meaning, usage, or translation. Dictionary has several kinds, one of which is a dictionary of terms. Medical Dictionary is a dictionary of terms that contain medical terms. Medical dictionary identical with the thick book that complicated and slow in its use. To overcome this we need a medical dictionary application. Medical dictionary applications require a search process to support the performance of the application. Search process is needed to shorten the time in the search itself. Searching is done using the Boyer - Moore algorithm is a string search algorithm that has the fastest searching time, because the strings match moves from right to left, so can shorten the time and simplify the use of medical dictionary application. This research was made based on Android using Java programming language with SQLite databases. After testing, 100% of 45 respondents stated that the application of this medical dictionary has the accuracy of the resulting word in the search process.

***Index Terms*—medical dictionary application, search string, Boyer-Moore algorithm, Java, SQLite.**

I. PENDAHULUAN

Kamus menurut daring KBBI (Kamus Besar Bahasa Indonesia) merupakan buku acuan yang memuat kata dan ungkapan, biasanya disusun menurut abjad berikut keterangan dan makna, pemakaian, atau terjemahannya [1]. Selain itu, kamus merupakan buku yang memuat kumpulan istilah atau nama yang disusun menurut abjad beserta dengan penjelasan makna dan pemakaiannya.

Kamus memiliki berbagai macam jenis, sesuai dengan isi yang terkandung di dalamnya. Ada kamus bahasa, kamus istilah, dan ada juga jenis kamus yang menjadi pedoman disiplin ilmu tertentu, misalnya kamus komputer dan kamus kedokteran.

Kamus kedokteran merupakan kamus yang mencakup istilah-istilah kedokteran, termasuk di dalamnya mengenai istilah penyakit, obat-obatan, istilah medis dan peralatan yang biasa digunakan untuk praktik kesehatan dan kedokteran [2]. Kamus kedokteran sangat bermanfaat khususnya bagi mahasiswa kedokteran, namun dengan bentuk kamus kedokteran yang sangat tebal, dirasa kurang efektif dan sulit untuk dibawa kemana-mana.

Dari hasil survei yang dilakukan dengan menyebarkan kuesioner, diperoleh kesimpulan bahwa keberadaan kamus kedokteran yang identik dengan bentuk tebal dapat mempersulit dan memperlambat penggunaan kamus kedokteran itu sendiri sehingga dibutuhkan suatu aplikasi kamus kedokteran yang dapat mempermudah dan mempersingkat waktu dalam penggunaannya.

Proses pencarian dalam suatu aplikasi kamus kedokteran sangatlah penting, namun terkadang proses pencarian itu lambat. Untuk mempercepat dan mempermudah suatu proses pencarian, dibutuhkan suatu algoritma yang dapat memaksimalkan proses pencarian tersebut. Algoritma merupakan urutan langkah-langkah logis pada penyelesaian masalah yang disusun secara sistematis. Masalah dapat berupa apa saja, dengan catatan untuk setiap masalah ada syarat kondisi awal yang harus dipenuhi sebelum menjalankan algoritma [3]. Algoritma untuk pencarian pun sudah semakin berkembang dari hari ke hari. Algoritma pencarian yang dianggap memiliki hasil paling baik dalam praktiknya, yaitu algoritma yang bergerak mencocokkan *string* dari arah kanan ke kiri. Algoritma Boyer-Moore merupakan salah satu contoh algoritma yang menggunakan arah dari kanan ke kiri.

Pada penelitian sebelumnya telah disimpulkan bahwa algoritma Boyer-Moore memiliki waktu pencarian cepat [4]. Algoritma Boyer-Moore telah banyak dikenal oleh masyarakat dan dianggap paling efisien untuk pencarian *string*. Ide di balik algoritma ini adalah dengan memulai pencocokan karakter dari kanan dan bukan dari kiri, maka akan lebih banyak informasi yang didapat [5]. Sebelumnya telah ada yang menggunakan algoritma Boyer-Moore untuk membangun aplikasi permainan *word search puzzle*

[6] dan untuk membangun aplikasi *search engine* sederhana [7]. Sedangkan aplikasi kamus kedokteran, sebelumnya telah berhasil dibangun dengan menggunakan metode *Binary Search* berbasis *Web* [8]. Berdasarkan penelitian yang sudah ada sebelumnya, aplikasi ini menggunakan algoritma Boyer-Moore untuk mencari kata atau istilah yang di-*input* oleh *user*.

II. ALGORITMA BOYER-MOORE

Algoritma Boyer-Moore adalah salah satu algoritma untuk mencari suatu *string* di dalam teks, dibuat oleh R.M Boyer dan J.S Moore. Algoritma Boyer-Moore melakukan perbandingan dimulai dari kanan ke kiri, tetapi pergeseran *window* tetap dari kiri ke kanan. Jika terjadi kecocokkan maka dilakukan perbandingan karakter teks dan karakter pola yang sebelumnya, yaitu dengan sama-sama mengurangi indeks teks dan pola masing-masing sebanyak satu [9]. Dengan mengunakan algoritma ini, secara rata-rata proses pencarian akan menjadi lebih cepat jika dibandingkan dengan algoritma lainnya. Alasan melakukan pencocokkan dari kanan (posisi terakhir *string* yang dicari) ditunjukkan dalam contoh berikut [10].

Tabel 1. Contoh Algoritma Boyer-Moore

M	A	K	A	N		T	O	M	A	T
T	O	M	A	T						

Pada contoh di atas, dengan melakukan perbandingan dari posisi paling akhir *string* dapat dilihat bahwa karakter ‘n’ pada *string* “makan” tidak cocok dengan karakter “t” pada *string* “tomat” yang dicari, dan karakter “n” tidak pernah ada dalam *string* “tomat” yang dicari sehingga *string* “tomat” dapat digeser melewati *string* “makan”, sehingga posisinya seperti berikut.

Tabel 2. Contoh Algoritma Boyer-Moore

M	A	K	A	N		T	O	M	A	T
					T	O	M	A	T	

Dalam contoh terlihat bahwa algoritma Boyer-Moore memiliki loncatan karakter yang besar sehingga mempercepat pencarian *string* karena dengan hanya memeriksa sedikit karakter, dapat langsung diketahui bahwa *string* yang dicari tidak ditemukan dan dapat digeser ke posisi berikutnya.

A. Langkah-Langkah Algoritma Boyer-Moore

Dalam penggunaan algoritma Boyer-Moore terdapat beberapa langkah yang harus dilakukan yaitu [10]:

1. Buat tabel pergeseran *string* yang dicari (S) dengan pendekatan *Match Heuristic* (MH) dan *Occurence Heuristic* (OH), untuk menentukan jumlah pergeseran yang akan dilakukan jika mendapat karakter tidak cocok pada proses pencocokkan dengan *string* (T).
2. Jika dalam proses pembandingan terjadi ketidakcocokkan antara pasangan karakter pada S dan karakter pada T, pergeseran dilakukan dengan memilih salah satu nilai pergeseran dari dua tabel analisa *string* yang memiliki nilai pergeseran paling besar.
3. Dua kemungkinan penyelesaian dalam melakukan pergeseran S, jika sebelumnya belum ada karakter yang cocok adalah dengan melihat nilai pergeseran hanya pada tabel *Occurence Heuristic*, jika karakter yang tidak cocok tidak ada pada S, maka pegeseran adalah sebanyak jumlah karakter pada S; dan jika karakter yang tidak cocok ada pada S, maka banyaknya pergeseran bergantung dari nilai pada tabel.
4. Jika karakter pada teks yang sedang dibandingkan cocok dengan karakter pada S, maka posisi karakter pada S dan T diturunkan sebanyak 1 posisi, kemudian dilanjutkan dengan pencocokkan pada posisi tersebut dan seterusnya. Jika kemudian terjadi ketidakcocokkan karakter S dan T, maka dipilih nilai pergeseran terbesar dari dua tabel analisa *pattern,* yaitu nilai dari tabel *Match Heuristic* dan nilai tabel *Occurence Heuristic* dikurangi dengan jumlah karakter yang telah cocok.
5. Jika semua karakter telah cocok, artinya S telah ditemukan di dalam T, selanjutnya geser *pattern* sebesar 1 karakter.
6. Lanjutkan sampai akhir *string* T.

B. Bad-Character Shift (Occurance Heuristic)

Tabel *Occurrence Heuristic* sering disebut juga *Bad-Character Shift*, dimana pergeserannya dilakukan berdasarkan karakter apa yang menyebabkan tidak cocok dan seberapa jauh karakter tersebut dari karakter paling akhir.

Untuk menghitung tabel *Occurrence Heuristic* harus menggunakan langkah-langkah sebagai berikut [10]:

Contoh *string*: manaman

Panjang: 7 karakter

Tabel 3. *Occurrence Heuristic*

Posisi:	1	2	3	4	5	6	7
String:	M	A	N	A	M	A	N
Pergeseran (OH)	2	1	0	1	2	1	0

1. Lakukan pencacahan mulai dari posisi terakhir *string* sampai ke posisi awal, dimulai dengan nilai 0 karena terletak pada jarak terakhir, catat karakter yang sudah ditemukan (dalam contoh ini karakter "n")
2. Mundur ke posisi sebelumnya, nilai pencacah ditambah 1, jika karakter pada posisi ini belum pernah ditemukan, maka nilai pergeserannya adalah sama dengan nilai pencacah (dalam contoh ini, karakter "a" belum pernah ditemukan sehingga nilai pergeserannya adalah sebesar nilai pencacah yaitu 1)
3. Mundur ke posisi sebelumnya, karakter "m" nilai pergeserannya 2
4. Mundur lagi, karakter "a". Karakter "a" sudah pernah ditemukan sebelumnya sehingga nilai pergeserannya sama dengan nilai pergeseran karakter "a" yang sudah ditemukan paling awal yaitu 1.
5. Begitu seterusnya sampai posisi awal *string*.

Catatan: untuk karakter selain "m, a, n", nilai pergeseran sebesar panjang *string,* yaitu tujuh karakter (sepanjang *pattern*).

C. Good-Suffix Shift (Match Heuristic)

Tabel *Match Heuristic* sering disebut juga *Good-Suffix Shift*, dimana pergeserannya dilakukan berdasarkan posisi ketidakcocokkan karakter yang terjadi. Maksudnya untuk menghitung tabel *Match Heuristic*, perlu diketahui pada posisi keberapa terjadi ketidakcocokkan. Posisi ketidakcocokkan itulah yang akan menentukan besar pergeseran.

Untuk menghitung tabel *Match Heuristic* harus menggunakan langkah-langkah sebagai berikut [10]:

Contoh *string*: manaman

Panjang: 7 karakter

Tabel 4. *Match Heuristic*

Posisi:	1	2	3	4	5	6	7
String:	M	A	N	A	M	A	N
Pergeseran (MH)	4	4	4	4	7	7	1

1. Jika karakter pada posisi 7 bukan "n" maka geser 1 posisi, berlaku untuk semua *pattern* yang dicari.
2. Jika karakter "n" sudah cocok, tetapi karakter sebelum "n" bukan "a", maka geser sebanyak 7 posisi, sehingga posisi *pattern* melewati teks, karena sudah pasti "manambn" bukan "manaman"
3. Jika karakter "an" sudah cocok, tetapi karakter sebelum "an" bukan "m" maka geser sebanyak 7 posisi, sehingga posisi *pattern* melewati teks, karena sudah pasti "manaban" bukan "manaman"
4. Jika karakter "man" sudah cocok, tetapi karakter sebelum "man" bukan "a" maka geser sebanyak 4 posisi, sehingga posisi *pattern* berada atau bersesuaian dengan akhiran "man" yang sudah ditemukan sebelumnya, karena bisa saja akhiran "man" yang sudah ditemukan sebelumnya merupakan awalan dari *pattern* "manaman" yang berikutnya.
5. Jika karakter "aman" sudah cocok, tetapi karakter sebelum "aman" bukan "n" maka geser sebanyak 4 posisi, sehingga posisi *pattern* berada/bersesuaian dengan akhiran "man" yang sudah ditemukan sebelumnya, karena bisa saja akhiran "man" yang sudah ditemukan sebelumnya merupakan awalan dari pattern "manaman" yang berikutnya.

Selanjutnya sama, pergeseran paling mungkin dan aman dalam tabel *Match Heuristic* adalah pergeseran sebanyak 4 posisi.

Berikut ini dijabarkan *pseudocode* dari algoritma Boyer-Moore.

```
procedure preBmBc(
    input P : array[0..n-1] of char,
    input n : integer,
    input/output bmBc : array[0..n-1] of integer
)
Deklarasi:
  i: integer

Algoritma:
  for (i := 0 to ASIZE-1)
     bmBc[i] := m;
  endfor
  for (i := 0 to m - 2)
     bmBc[P[i]] := m - i - 1;
  endfor
```

Gambar 1. *Pseudocode Bad-Character Shift*

```
procedure preSuffixes(
    input P : array[0..n-1] of char,
    input n : integer,
    input/output suff : array[0..n-1] of integer
)
Deklarasi:
  f, g, i: integer

Algoritma:
  suff[n - 1] := n;
  g := n - 1;
  for (i := n - 2 downto 0) {
     if (i > g and (suff[i + n - 1 - f] < i - g))
        suff[i] := suff[i + n - 1 - f];
     else if (i < g)
           g := i;
        endif
     f := i;
       while (g >= 0 and P[g] = P[g + n - 1 - f])
         --g;
       endwhile
     suff[i] = f - g;
     endif
  endfor
```

Gambar 2. *Pseudocode Good-Suffix Shift*

```
procedure preBmGs(
    input P : array[0..n-1] of char,
    input n : integer,
    input/output bmBc : array[0..n-1] of integer
)
Deklarasi:
  i, j: integer
  suff: array [0..RuangAlpabet] of integer

  preSuffixes(x, n, suff);

  for (i := 0 to m-1)
     bmGs[i] := n
  endfor
  j := 0
  for (i := n - 1 downto 0)
      if (suff[i] = i + 1)
         for (j:=j to n - 2 - i)
            if (bmGs[j] = n)
               bmGs[j] := n - 1 - i
             endif
         endfor
       endif
  endfor
  for (i = 0 to n - 2)
     bmGs[n - 1 - suff[i]] := n - 1 - i;
  endfor
```

Gambar 3. *Pseudocode* Perhitungan *Suffix*

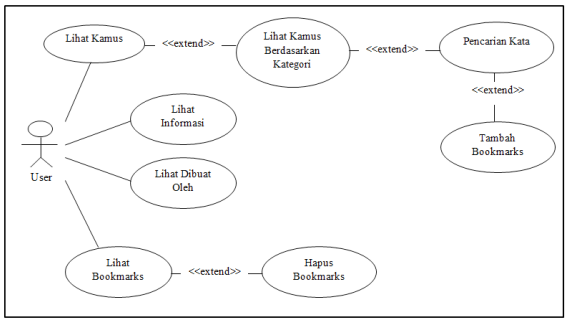
```
procedure BoyerMooreSearch(
    input m, n : integer,
    input P : array[0..n-1] of char,
    input T : array[0..m-1] of char,
    output ketemu : array[0..m-1] of boolean
)

Deklarasi:
i, j, shift, bmBcShift, bmGsShift: integer
BmBc : array[0..255] of interger
BmGs : array[0..n-1] of interger

Algoritma:
preBmBc(n, P, BmBc)
preBmGs(n, P, BmGs)
i:=0
while (i<= m-n) do
    j:=n-1
    while (j >=0 n and T[i+j] = P[j]) do
       j:=j-1
    endwhile
    if(j < 0) then
       ketemu[i]:=true;
       shift := bmGs[0]
    else
       bmBcShift:= BmBc[chartoint(T[i+j])]-n+j+1
       bmGsShift:= BmGs[j]
       shift:= max(bmBcShift, bmGsShift)
    endif
    i:= i+shift
endwhile
```

Gambar 4. *Pseudocode* Pencarian *Boyer-Moore*

III. PERANCANGAN APLIKASI

A. *Use Case Diagram*

Aplikasi Kamus Kedokteran ini dapat diakses oleh *user* melalui *gadget* Android-nya. Aktivitas yang dapat dilakukan oleh *user,* yaitu memilih menu Kamus, menu *Bookmarks*, menu Informasi, menu Dibuat Oleh, dan menu Keluar. Di dalam menu kamus terdapat empat menu yang sudah dikategorikan yaitu menu Kedokteran Umum, menu Kedokteran Gigi, menu Anatomi, dan menu Penyakit.



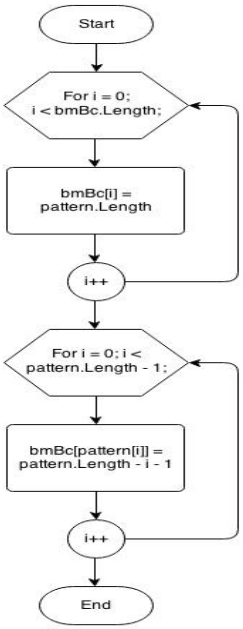
Gambar 5. *Use Case Diagram* Aplikasi Kamus

B. *Flowchart*

Dalam aplikasi Kamus Kedokteran ini memiliki beberapa *flowchart,* dimana masing-masing *flowchart* merepresentasikan alur kerja tiap bagian dari suatu sistem.

B.1 Flowchart Perhitungan Tabel *Bad Character Shift*

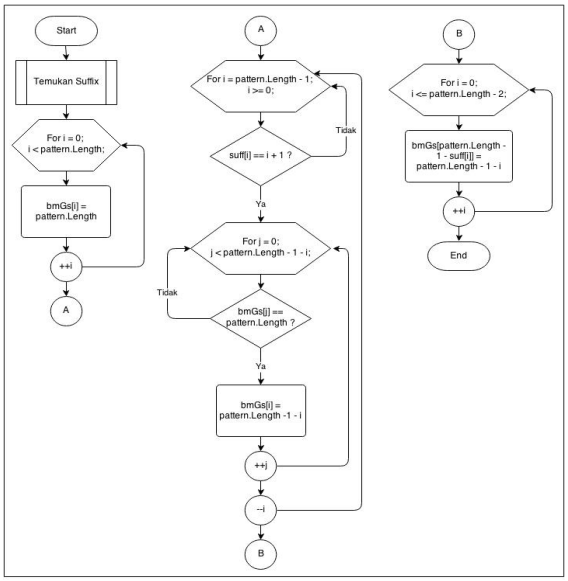
Flowchart ini menggambarkan alur kerja yang terjadi pada saat perhitungan tabel *bad-character shift*. Tabel *bad-character shift* berisi nilai-nilai pergeseran yang dapat dilakukan untuk setiap karakter yang terdapat di alfabet, nomor, dan tanda baca yang umum digunakan. Setiap karakter yang ada di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



Gambar 6. *Flowchart* Proses Perhitungan Tabel *Bad-Character Shift*

B.2 Flowchart Perhitungan Tabel *Good-Suffix Shift*

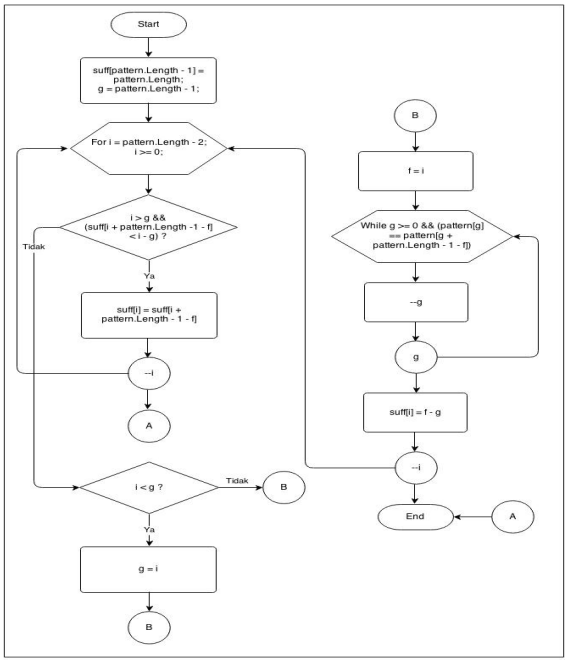
Flowchart ini menggambarkan alur kerja yang terjadi pada saat perhitungan tabel *good-suffix shift*. Tabel *good-suffix shift* berisi nilai yang dapat digunakan untuk pergeseran ketika ketidakcocokan ditemukan berdasarkan karakter pada posisi keberapa yang menyebabkan ketidakcocokan. Untuk menentukan nilai dari tabel *good-suffix shift*, perlu menghitung terlebih dahulu tabel *suffix* yang fungsinya memberi tanda jika terdapat perulangan akhiran. Bila tabel *suffix* sudah dibuat, tabel *good-suffix shift* dapat dihitung. Nilai dari tiap karakter yang ada di dalam *pattern* bergantung pada apakah adanya perulangan akhiran (*suffix*) atau tidak. Semakin banyak perulangan yang terjadi, semakin kecil nilai pergeseran.



Gambar 7. *Flowchart* Proses Perhitungan Tabel *Good-Suffix Shift*

B.3 Flowchart Perhitungan Tabel *Suffix*

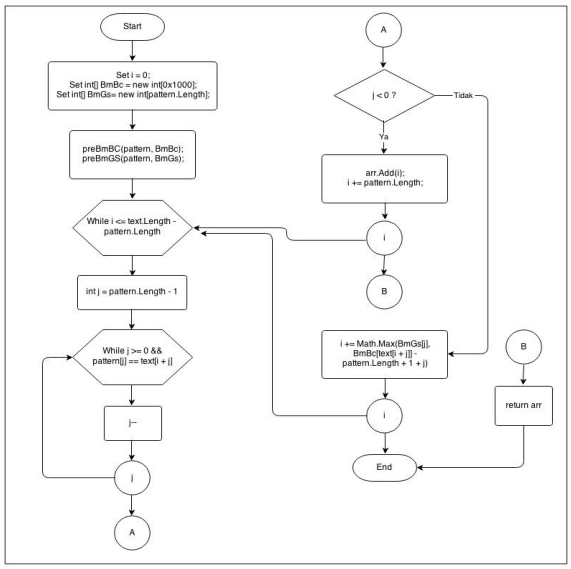
Flowchart ini menggambarkan alur yang terjadi pada saat perhitungan tabel *suffix* yang berguna untuk menghitung tabel *good-suffix shift* nantinya. Tabel ini berisi nilai-nilai dari tiap karakter yang ada di *pattern* yang menunjukkan adanya perulangan akhiran (*suffix*) atau tidak dan dimana letak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



Gambar 8. *Flowchart* Proses Perhitungan Tabel *Suffix*

B.4 Flowchart Fase Pencarian Algoritma Boyer-Moore

Flowchart ini menggambarkan alur pada proses pencarian algoritma Boyer-Moore, dimana proses awal yang dilakukan yaitu penempatan *window* dari *pattern* di *text* yang tersedia. Proses pencarian dimulai dari karakter paling kanan *pattern*. Setiap karakter akan dibandingkan satu per satu. Jika terjadi ketidakcocokan, maka akan dicek nilai pergeseran yang mungkin dilakukan dengan melihat ke tabel *bad-character shift* dan *good-suffix shift*. Nilai terbesar yang didapat di antara kedua tabel tersebut akan diambil dan pergeseran *window* akan dilakukan sesuai dengan nilai tersebut.



Gambar 9. *Flowchart* Fase Pencarian Algoritma Boyer-Moore

IV. IMPLEMENTASI DAN UJI COBA

Tampilan awal yang akan muncul apabila aplikasi pertama kali dijalankan. Tampilan ini berisi logo aplikasi Kamus Kedokteran. Tampilan ini akan otomatis hilang dan diganti dengan halaman menu apabila waktu telah mencapai 5 detik.

A. Implementasi

Gambar 10. Tampilan Aplikasi Kamus Kedokteran

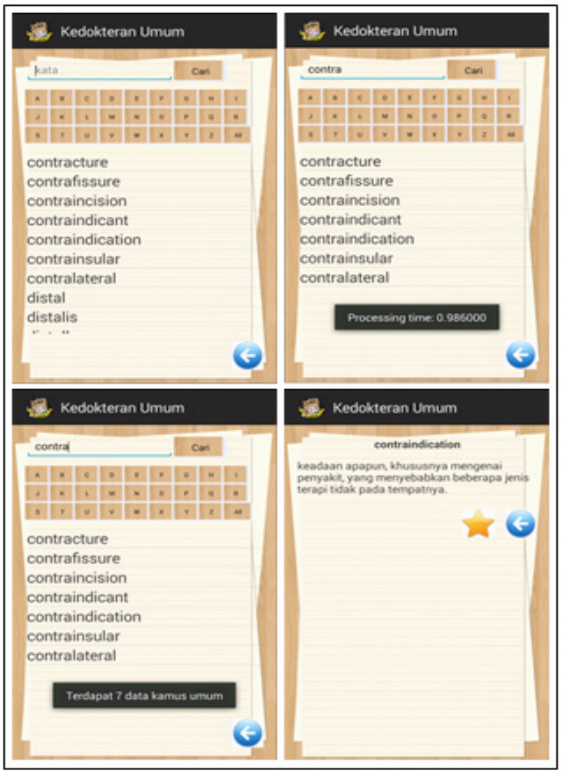
Halaman menu Kedokteran Umum menampilkan daftar kata atau istilah dari kamus kedokteran umum yang direpresentasikan ke dalam sebuah *listview*. Halaman ini akan meminta *input* dari *user* yang kemudian akan dijadikan parameter untuk proses pencarian. Tombol cari adalah tombol yang digunakan untuk memulai proses pencarian. Apabila kata yang dicari berhasil ditemukan, maka halaman ini akan langsung menampilkan hasil pencarian. Tombol kembali adalah tombol yang digunakan untuk kembali ke halaman sebelumnya.

Apabila pada proses pencarian berhasil ditemukan, akan mengembalikan suatu nilai yang merupakan posisi dari suatu kata yang dicari, kemudian akan mengambil data tersebut dari *database*, lalu menampilkan hasil pencarian ke dalam sebuah *listview* beserta dengan *toast* yang berisi informasi mengenai waktu pencarian dan jumlah data yang ditemukan dari proses pencarian.

Apabila *user* memilih salah satu *item* dari *listview* kedokteran umum, akan berpindah ke halaman baru yang berisi detail dari *item* tersebut.

Halaman detail kedokteran umum memiliki dua buah *text field* yang berisi detail dari kedokteran umum. Halaman ini juga memiliki tombol *bookmark* dan tombol kembali. Tombol *bookmark* berfungsi untuk menyimpan kata ke dalam daftar penyimpanan. Kata yang sudah terdapat di dalam *database*

MASTER_BOOKMARK tidak dapat ditambahkan kembali. Untuk mencegah adanya duplikasi data, maka pada saat proses pencarian detail dilakukan juga proses pengecekan kata dengan *database* pada tabel MASTER_BOOKMARK

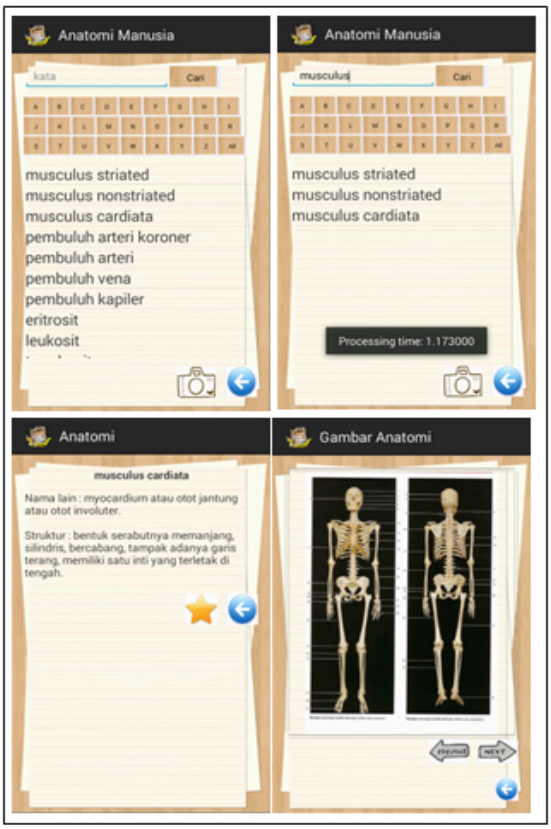


Gambar 11. Tampilan Menu Kedokteran Umum

Halaman menu Anatomi menampilkan daftar kata atau istilah dari kamus anatomi yang direpresentasikan ke dalam sebuah *listview.* Halaman ini akan meminta *input* dari *user* yang kemudian akan dijadikan parameter untuk proses pencarian. Tombol cari adalah tombol yang digunakan untuk memulai proses pencarian. Apabila kata yang dicari berhasil ditemukan, maka halaman ini akan langsung menampilkan hasil pencarian. Tombol gambar anatomi berfungsi berpindah ke halaman gambar anatomi yang akan menampilkan gambar dari anatomi manusia. Tombol kembali adalah tombol yang digunakan untuk kembali ke halaman sebelumnya.

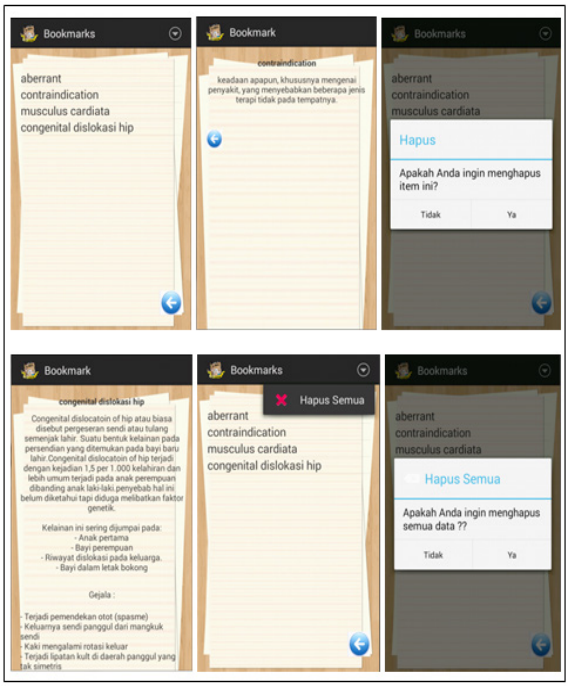
Apabila *user* memilih salah satu *item* dari *listview* anatomi, akan berpindah ke halaman baru yang berisi detail dari *item* tersebut.

Halaman detail anatomi memiliki dua buah *text field* yang berisi detail dari anatomi. Halaman ini juga memiliki tombol *bookmark* dan tombol kembali. Tombol *bookmark* berfungsi untuk menyimpan kata ke dalam daftar penyimpanan. Kata yang sudah terdapat di dalam *database* MASTER_BOOKMARK tidak dapat ditambahkan kembali. Untuk mencegah adanya duplikasi data, maka pada saat proses pencarian detail diakukan juga proses pengecekan kata dengan *database* pada tabel MASTER_BOOKMARK.



Gambar 12. Tampilan Menu Anatomi

Halaman menu *Bookmark* didesain untuk menampilkan daftar kata yang telah disimpan sebelumnya. Jika salah satu *item* pada *bookmark* ditekan, maka akan muncul halaman baru yang berisi detail dari *item* tersebut. Pada halaman *bookmark* ini, *user* dapat menghapus *bookmark* baik satu persatu, maupun secara keseluruhan sekaligus. Pada saat melakukan penghapusan, akan muncul kotak *dialog* dimana *user* diminta untuk konfirmasi apakah yakin penghapusan akan dilakukan. Pada kanan atas halaman ini terdapat menu *dropdown* yang berisi tombol untuk menghapus semua data dalam sekali proses. Jika ditekan, maka akan muncul kotak *dialog* sebagai konfirmasi bahwa semua data yang telah disimpan akan dihapus dari *database.*



Gambar 13. Tampilan Menu *Bookmarks*

B. Uji Coba

Dalam uji coba disiapkan delapan kata, responden diminta untuk melakukan pencarian kata tersebut sesuai dengan masing-masing kategori dan melengkapi kolom "ditemukan berapa data" dan kolom "waktu" pada tabel. Uji coba ini dilakukan untuk menilai performa aplikasi ini khususnya pada proses pencarian, apakah sudah berjalan dengan baik dalam melakukan pencarian.

Total responden pada penelitian aplikasi Kamus Kedokteran ini berjumlah 45 orang, dengan alasan karena jumlah responden minimum yang mendekati kurva normal yaitu 30 responden, namun semakin banyak responden akan menghasilkan statistik yang lebih akurat. Rata-rata responden berusia 20 – 35 tahun dengan profesi dokter, perawat, bidan, dan mahasiswa kedokteran.

Tabel 5. Tabel Profesi Responden

No.	Profesi	Jumlah	Persentase
1.	Dokter	5	11,11%
2.	Perawat	28	62,22%
3.	Bidan	2	4,44%
4.	Mahasiswa/i Kedokteran	10	22,22%

Berdasarkan hasil uji coba, didapatkan jumlah waktu pencarian yang diperoleh dari 45 responden dan rata-rata waktu pencarian dari 45 responden untuk tiap istilah dalam masing-masing kategori.

Tabel 6. Tabel Hasil Waktu Uji Coba

No.	Istilah	Jumlah Waktu	Rata-Rata
1.	Contra	0.137	0.00304
2.	Distal	0.068	0.00151
3.	Cement	0.122	0.00271
4.	Dyspha	0.073	0.00162
5.	Pembuluh	0.092	0.00204
6.	Musculus	0.080	0.00180
7.	Congenital	0.093	0.00207
8.	Fibro	0.066	0.00147
	Rata-Rata:		**0.00203**

Pada tahap uji coba, terdapat empat parameter yang telah dinilai oleh para responden yang mencoba aplikasi kamus kedokteran ini. Hasil dari empat parameter tersebut dipaparkan pada tabel 6.

Tabel 7. Tabel Uji Coba Aplikasi Kamus Kedokteran

No.	Parameter	Ya	Tidak	Persentase
1.	Kemudahan dalam menggunakan aplikasi Kamus Kedokteran	44	1	97,8%
2.	Keakuratan kata dengan hasil dari proses pencarian	45	0	100%
3.	Keindahan desain aplikasi Kamus Kedokteran	43	2	95,6%
4.	Memberikan manfaat	45	0	100%

V. SIMPULAN DAN SARAN

A. Simpulan

Berdasarkan hasil implementasi dan uji coba yang dilakukan sebelumnya, simpulan dari penelitian ini adalah implementasi algoritma Boyer-Moore dalam pencarian kata pada aplikasi Kamus Kedokteran berhasil dilakukan. Simpulan ini dirumuskan berdasarkan hasil uji coba, dimana pencarian kata dengan algoritma Boyer-Moore berhasil dilakukan dengan persentase sebesar 100% dari responden menyatakan bahwa proses pencarian istilah pada aplikasi Kamus Kedokteran dapat memberikan penjelasan (hasil) yang sesuai dengan yang diharapkan.

B. Saran

Berdasarkan penelitian yang telah dilakukan, terdapat beberapa saran yang dapat digunakan sebagai

masukan untuk penelitian selanjutnya.

1. Data yang disimpan dalam *database* ditambah perbendaharaan katanya pada masing-masing kategori.
2. Dapat ditambahkan fitur gambar 3D, agar lebih menarik dan jelas dalam pembelajaran mengenai istilah-istilah kedokteran.
3. Dapat menambahkan fitur gambar pada kategori penyakit agar lebih menarik dan jelas dalam pendeskripsiannya.
4. Dapat menggunakan algoritma pencarian *string* lainnya, seperti algoritma Reverse Colussi.

DAFTAR PUSTAKA

[1] http://bahasa.kemdiknas.go.id/kbbi/index.php

[2] Isnani, Martina Husnul. 2010. *Aplikasi Online Kamus Kedokteran Dengan Menggunakan Metode Binary Search*. Dalam http://lib.uin-malang.ac.id/?mod=th_detail&id=05550012 yang diakses tanggal 8 Februari 2014.

[3] Indra, Anisa. 2013. "Pengertian Algoritma". Dalam http://www.varia.web.id/2013/05/pengertian-algoritma.html yang diakses tanggal 8 Februari 2014.

[4] Sagita, Vina. 2012. Dalam jurnal dengan judul *Studi Perbandingan Implementasi Algoritma Boyer-Moore, Turbo Boyer Moore, dan Tuned Boyer-Moore Dalam Pencarian String*. Tangerang: Universitas Multimedia Nusantara.

[5] Fauzy, Mufthy. 2011. "Sekilas Tentang Ilmu Komputer & Informatika". Dalam http://mufthyrocket.blogspot.com/2011/11/sekilas-tentang-ilmu-komputer.html yang diakses tanggal 8 Februari 2014.

[6] Gunawan, Steven Kristanto. 2013. Dalam jurnal dengan judul *Implementasi Algoritma Boyer-Moore pada Permainan Word Search Puzzle.* Yogyakarta: Universitas Kristen Duta Wacana.

[7] Soleh, Moch. Yusup. 2011. Dalam jurnal dengan judul *Implementasi Algoritma KMP dan Boyer-Moore dalam Aplikasi Search Engine Sederhana*. Bandung: Institus Teknologi Bandung.

[8] Isnani, Martina Husnul. 2010. *Aplikasi Online Kamus Kedokteran Dengan Menggunakan Metode Binary Search*. Dalam http://lib.uin-malang.ac.id/?mod=th_detail&id=05550012 yang diakses tanggal 8 Februari 2014.

[9] Kumara, Gozali Harda. 2009. Dalam jurnal dengan judul *Visualisasi Beberapa Algoritma Pencocokan String Dengan Java*. Bandung: Institut Teknologi Bandung.

[10] Chiquita, Christabella. 2012. Dalam jurnal dengan judul *Penerapan Algoritma Boyer-Moore-Dynamic Programming untuk Layanan Auto-Complete dan Auto Correct*. Bandung: Institut Teknologi Bandung.