BAB II

TINJAUAN PUSTAKA

2.1 Kamus

Kamus adalah buku referensi atau rujukan yang menerangkan makna kata-kata dengan kata-kata lain. Kamus memberikan definisi dari setiap masukkan dan dilihat dari sudut pandang linguistik. Kamus 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 untuk suatu kata.

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

2.1.1 Macam-Macam Kamus

Menurut KBBI (Kamus Besar Bahasa Indonesia), terdapat beberapa macam kamus, yaitu:

a. Kamus diakronis: kamus yang melacak asal dan perkembangan kata dalam satu atau beberapa periode sejarah bahasa.
b. Kamus istilah: kamus yang memuat daftar istilah dengan makna konsepnya dari bidang ilmu tertentu.

c. Kamus ekabahasa: kamus yang disusun dengan menggunakan satu bahasa.

d. Kamus dwibahasa: kamus yang disusun dengan menggunakan dua bahasa.

e. Kamus multibahasa: kamus yang terdiri dari tiga bahasa atau lebih.

f. Kamus umum: kamus yang memuat daftar kosakata yang umum dipakai dengan keterangan makna dan penggunaannya dalam satu bahasa.

g. Kamus topografis: kamus yang berisi informasi ensiklopedis tentang tempat atau wilayah tertentu.

h. Kamus visual: kamus bergambar dengan ilustrasi yang menarik dan berwarna.

i. Kamus sinonim: kamus yang memuat daftar kosakata dengan padanannya dalam satu bahasa.

j. Kamus sintagmatis: kamus yang memuat informasi tentang kolokasi, konstruksi, ungkapan tetap, dan frasa.

k. Kamus elektronik: kamus yang dikemas dalam bentuk cakram, disket, atau seperti kalkulator, dapat didistribusikan secara daring melalui jaringan komputer atau internet.

2.2 Kamus Kedokteran

Kamus kedokteran merupakan salah satu contoh dari kamus khusus/kamus istilah, dimana kamus ini hanya memuat kata-kata dari suatu bidang tertentu, yaitu bidang 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 (Isnani, 2010).

2.3 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 (Kumara, 2009). 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 (Chiquita, 2012).

Tabel 2.1 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 2.2 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.

Pseudocode 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 2.1 *Pseudocode* Penghitungan Tabel *bad-character* (Kumara, 2009)

```
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 2.2 *Pseudocode* Penghitungan *Suffix* (Kumara, 2009)

```
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 2.3 *Pseudocode* Penghitungan Tabel *good-suffix* (Kumara, 2009)

Pada gambar 2.3 menunjukkan *pseudocode* dari penghitungan tabel *good-suffix shift.* Dimana tabel *good-suffix shift* ini terdiri dari sejumlah nilai pergeseran yang akan digunakan pada saat terjadi ketidakcocokan antara karakter pada *pattern* dan karakter pada teks. Namun, sebelum membangun tabel *good-suffix shift*, perlu dibangun dahulu tabel *suffix* untuk mengetahui apakah terdapat perulangan akhiran atau tidak.

```
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 2.4 *Pseudocode* Algoritma Boyer-Moore *Search*

2.3.1 Langkah-Langkah Algoritma Boyer-Moore

Dalam penggunaan algoritma Boyer-Moore terdapat beberapa langkah yang harus dilakukan yaitu (Chiquita, 2012):

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.

2.3.2 Occurrence Heuristic (Bad-Character Shift)

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 (Chiquita, 2012):

Contoh *string*: manaman

Panjang: 7 karakter

Tabel 2.3 *Occurrence Heuristic* (Chiquita, 2012)

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*).

2.3.3 Match Heuristic (Good-Suffix Shift)

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 (Chiquita, 2012):

Contoh *string*: manaman

Panjang: 7 karakter

Tabel 2.4 *Match Heuristic* (Chiquita, 2012)

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

2.4 Android

Android merupakan sebuah sistem operasi yang berbasis Linux untuk telepon seluler seperti telepon pintar dan komputer tablet. Android itu sendiri adalah *platform* yang sangat lengkap, baik itu sistem operasinya, aplikasi dan *tool* pengembangan, *market* aplikasi Androidnya, serta didukung tinggi dari komunitas *open source* di dunia, sehingga Android terus berkembang pesat baik dari segi teknologi maupun dari segi jumlah *device* yang ada di dunia.

Menurut Suprianto (2012, h. 11) secara garis besar sistem operasi android terbagi menjadi lima tingkatan:

1. *Linux Kernel*

   Adalah kernel dasar android. Tingkat ini berisi semua *driver* perangkat tingkat rendah untuk komponen-komponen *hardware* perangkat Android.

2. *Libraries*

   Berisi semua kode program yang menyediakan layanan-layanan utama sistem operasi Android.

3. Android *Runtime*

   Kedudukannya setingkat dengan *libraries*, Android *Runtime* menyediakan kumpulan pustaka inti yang dapat diaktifkan oleh pengembang untuk menulis kode aplikasi Android dengan bahasa pemrograman Java.

4. *Application Framework*

   Adalah semacam kumpulan *class built-in* yang tertanam dalam sistem operasi Android, sehingga pengembang dapat memanfaatkannya untuk aplikasi yang sedang dibangun.

5. *Applications*

   Pada tingkat ini akan bekerja, contoh aplikasi ini banyak ditemui, seperti: *Phone*, *Contact*, *Browse* dan lain-lain.

Android memiliki banyak versi seiring dengan kemajuan teknologi, versi terakhir sistem operasi Android yang dirilis, yaitu versi 4.3 dengan nama *Jelly Bean*. Pada tanggal 3 September 2013, Android telah mengumumkan akan ada versi 4.4 yang diberi nama *Kitkat*. Dalam penelitian ini penulis akan melakukan uji coba sistem pada Android versi 4.2.2 (Jelly Bean) dengan minimum spesifikasi Android versi 3.0 (Honeycomb).