BAB II
LANDASAN TEORI

2.1 Kecerdasan Buatan

Kecerdasan buatan merupakan bagian dari ilmu pengetahuan komputer yang khusus ditujukan dalam perancangan otomatisasi tingkah laku cerdas dalam sistem kecerdasan komputer. Bagian utama dari kecerdasan buatan adalah basis pengetahuan (*knowledge base*), yaitu suatu pengertian atau pemahaman tentang wilayah subjek yang diperoleh melalui pembelajaran dan pengalaman (Kristanto, 2003).

Kecerdasan buatan mengembangkan perangkat lunak dan perangkat keras untuk menirukan tindakan manusia. Aktivitas manusia yang ditirukan seperti penalaran, penglihatan, pembelajaran, pemecahan masalah, pemahaman bahasa alami dan sebagainya. Teknologi kecerdasan buatan dapat dipelajari dalam berbagai bidang-bidang seperti robotika (*robotics*), penglihatan komputer (*komputer vision*), pengolahan bahasa alami (*natural language processing*), pengenalan pola (*pattern recognitiion*), sistem syaraf buatan (*artificial neural system*), pengenalan suara (*speech recognition*) dan sistem pakar (*expert system*). (Simarmata, 2010).

2.2 Representasi Pengetahuan

Representasi pengetahuan adalah suatu teknik atau metode untuk merepresentasikan basis pengetahuan yang diperoleh ke dalam suatu skema atau diagram tertentu sehingga dapat diketahui relasi antara suatu data dengan data yang lain sehingga dapat diuji kebenaran penalarannya. Representasi pengetahuan dimaksudkan untuk mengorganisasikan pengetahuan dalam bentuk tertentu (Simarmata, 2010).

Adapun karakteristik dari metode representasi pengetahuan adalah:

1. Harus bisa diprogram dengan bahasa pemrograman dan hasilnya disimpan dalam memori.

2. Dirancang sedemikian rupa sehingga isinya dapat digunakan untuk proses penalaran.
3. Model representasi pengetahuan merupakan sebuah struktur data yang dapat dimanipulasi oleh mesin inferensi dan pencarian untuk aktivitas pencocokan pola.

2.3 *Chatbot*

Chatbot merupakan salah satu program dalam kecerdasan buatan yang dirancang untuk dapat berkomunikasi langsung dengan manusia sebagai penggunanya, yang membedakan *chatbot* dengan sistem pemrosesan bahasa alami (Natural Language Processing System) adalah kesederhanaan algoritma yang di gunakan. Meskipun banyak bots yang dapat menginterpretasikan dan menanggapi input manusia, sebenarnya bots tersebut hanya mengartikan kata kunci dalam input dan membalasnya dengan kata kunci yang paling cocok, atau pola kata-kata yang paling mirip dari data yang telah ada dalam database yang telah dibuat sebelumnya.

Chat dapat diartikan sebagai pembicaraan. *Bot* merupakan sebuah program yang mengandung sejumlah data, jika diberikan masukan maka akan memberikan jawaban. *Chatbot* dapat menjawab pertanyaan dengan membaca tulisan yang diketikkan oleh pengguna melalui *keyboard*.

Pada mulanya, program komputer (*bots*) ini diuji melalui *turing test*, yaitu dengan merahasiakan identitasnya sebagai mesin sehingga dapat membohongi orang yang berbicara dengannya. Jika pengguna tidak dapat mengidentifikasi *bots* sebagai suatu program komputer, maka *chatbot* tersebut dikategorikan sebagai kecerdasan buatan (*artificial intelligence*).

Salah satu *chatbot* yang terkenal adalah Eliza (Dr. Eliza) yang dikembangkan oleh Joseph Weizenbaum di MIT (*Massachusetts Institute of Technology*). Eliza mensimulasikan percakapan antara seorang psikiater dengan pasiennya dalam bahasa Inggris yang alami. *Chatbot* yang pertama adalah Eliza yang dibuat pada tahun 1964 sampai 1966 oleh Professor Joseph Weizenbaum di MIT (*Massachusetts Institute of Technology*), dengan tujuan untuk mempelajari

komunikasi *natural language* antara manusia dengan mesin. Eliza bertindak seolah-olah dia adalah seorang psikolog yang dapat menjawab pertanyaan-pertanyaan dari pasien dengan jawaban yang cukup masuk akal atau menjawabnya dengan pertanyaan balik (Nila, 2013).

Eliza adalah pelopor *chatbot,* Eliza dikenal sebagai program *chat* yang memiliki profesi sebagai seorang psikiater. Eliza mensimulasikan percakapan antara seorang psikiater dengan pasiennya menggunakan metode biasa yang dapat mencerminkan perasaan pasien dengan mengajukan pertanyaan-pertanyaan seperti: "*How do you ...*", "*Why do you feel like ...*", "*What do you think about ...*". Program ini akan mencari pola kata-kata tertentu pada *input* yang diberikan oleh pengguna, dan kemudian memberikan output yang sesuai (Ridwan, 2013).

2.3.1 Perkembangan *Chatbot*

Sistem percakapan otomatis kini telah berkembang, dan perusahaan-perusahaan sudah menggunakan sitem-sistem tersebut untuk membantu *call center* memberikan panduan kontak. *Chatbot* pun sudah di implementasikan melalui jejaring sosial, seperti *twitter* dan *Windows Live Messenger*. Portal *online* populer seperti *eBay* dan *PayPal* juga menggunakan agen *virtual multi* bahasa untuk memudahkan penggunanya. Misalnya, *PayPal* menggunakan *chatterbot* Louise untuk menangani *query* dalam bahasa Inggris dan *chatbot* Lea untuk *query* dalam bahasa Perancis. *Chatbot* tersebut Dikembangkan oleh VirtuOz. Selain itu *Chatbot* juga di implementasikan untuk bidang komersial, pendidikan, entertainment dan sektor pelayanan public (Kerly dkk, 2006).

2.3.2 Komponen Utama *Chatbot*

Chatbot terdiri dari dua komponen utama yakni *bot program* dan *brain file*. *Bot program* merupakan program utama pada *chat bot* yang akan mengakses *input* dari pengguna, melakukan *parsing* dan kemudian membawanya ke *brain file (knowledge base)* untuk kemudian diberikan respon. Adapun *bot program* sendiri terdiri dari komponen *scanner* dan *parser* (Rudiyanto, 2005).

Brain file merupakan otak dari *chat bot* itu sendiri yang menentukan bagaimana cara *chat bot* berpikir dan akan memberikan respon. *Brain file* berfungsi sebagaimana tabel informasi (*knowledge base*). Di dalam *brain file* inilah disimpan semua kosakata, kepribadian, dan pengetahuan (*knowledge*) dari *chatbot*. Semakin banyak pengetahuan yang dimiliki *chat bot* maka akan semakin besar ukuran file dari *brain file* tersebut (Rudiyanto, 2005).

2.3.3 *Scanner*

Scanner merupakan salah satu bagian dari kompilator bahasa pada komputer yang bertugas melakukan analisis leksikal. *Scanner* menerima *input* berupa *stream* karakter kemudian memilah program sumber menjadi satuan leksik yang disebut dengan *token*. *Token* ini akan menjadi *input* bagi *parser*. Di dalam aplikasi *chat bot*, yang dimaksud dengan program sumber yang diolah oleh *scanner* adalah berupa kalimat *input* dari pengguna.

2.3.4 *Parser*

Parser atau *syntactic analyzer* pada kompilator bahasa pemrograman berfungsi untuk memeriksa kebenaran kemunculan setiap *token*. Pada *Chatbot system*, fungsi dari *parser* ini agak berbeda karena *token* yang akan diolah, semuanya memiliki tipe yang sama yaitu berupa kata (*word*). Urutan kemunculan *token* akan diolah dengan mengacu pada *brain file* agar didapatkan makna kalimat yang sesungguhnya. Dengan kata lain, tahap analisa semantik terjadi di bagian *brain file*. Kemampuan dari *parser* untuk mengolah *token* dan bekerja sama dengan *brain file* inilah yang paling menentukan tingkat kecerdasan dari sebuah *chatbot*.

2.3.5 *Reasoning*

Reasoning adalah teknik penyelesaian masalah dengan cara mempresentasikan masalah ke dalam basis pengetahuan (*knowledge base*) menggunakan *logic* atau bahasa formal. Pada penerapan *chatbot* proses ini akan dikerahkan bila kata kunci terdapat dalam *knowledge base chatbot* dan dilakukan

untuk mengembalikan respon ke pengguna. Proses ini menunjukkan bahwa masukan yang diberikan oleh pengguna tidak diproses sebagai satuan kata, tetapi sebagai kalimat utuh (Herdi, 2013).

2.3.6 *Learning*

Learning merupakan pendefinisian aturan tertentu secara otomatis dalam menemukan aturan yang diharapkan bisa berlaku umum untuk data-data yang belum pernah kita ketahui. Pada penerapan *chatbot* proses *learning* dijalankan bila kata kunci pada masukan pengguna tidak terdapat dalam *knowledge base*. Kata kunci yang tidak ditemukan tersebut akan disimpan sebagai *dialog repository* untuk kemudian akan ditanyakan pada *bot program* (Kartika H, 2007).

2.3.7 Prinsip Kerja *Chatbot*

Bot program atau bagian aplikasi menentukan kemampuan dan keterampilan *chat bot* untuk berbicara pada anda atau pada pengguna lainnya, atau dengan kata lain *bot program* berperan sebagai mulut. *Chatbot* tidak tahu apa-apa, tidak punya nama dan tidak punya kepribadian. Untuk itu pengguna harus mengajarinya berbagai hal sehingga ia dapat berbicara dengan baik dan semua pelajaran tersebut dimasukkan ke *brain file* (Kartika H, Astari & Novita. 2007).

Sebagai contoh, pengguna mengeluh bahwa *chatbot* yang akan dirancang berbicara omong kosong dengan mengetik kalimat: "*You are talking nonsense, pinhead*!". Agar *chat bot* dapat memberikan respon terhadap kalimat tersebut, maka dapat ditambahkan baris berikut pada *brain file* dari *chat bot*:

You are talking nonsense ← *trigger line*
I am smarter than you ← *respon bot*

Dengan demikian jika pengguna sekarang *chatting* menggunakan kalimat yang disebutkan di atas maka *bot program* akan masuk ke dalam *brain file* dan kemudian memberikan respon dengan output yang sesuai dengan *trigger line* yaitu *you are talking nonsense* (Rudiyanto, 2005).

Respon yang sama juga dapat muncul jika anda juga menuliskan baris berikut pada *brain file*:

Nonsense ← trigger line

I am smarter than you ← respon bot

Respon yang sama juga akan muncul jika anda menuliskan:

a. *Talking nonsense ← trigger line*

   I am smarter than you ← respon bot

b. *Pinhead ← trigger line*

   I am smarter than you ← respon bot

Namun akan lebih bagus jika *chatbot* diberikan kemampuan untuk merespon kalimat yang beragam misalnya:

c. *You are talking nonsense ← trigger line*

   I am smarter than you ← respon bot

d. *Talking Nonsense ← trigger line*

   Talking is funny, isnt it? ← respon bot

e. *Nonsense ← trigger line*

   Don't be stupid! ← respon bot

f. *Pinhead ← trigger line*

   Please try not to be insulting,buddy ← respon bot

Dengan *brain file* seperti itu maka *chatbot* akan terasa lebih hidup karena dapat memberikan respon dengan berbagai kalimat yang berbeda.

2.4 *Pattern Matching*

Kecerdasan Buatan bekerja dengan menggunakan metode pencocokan pola (*pattern matching*). *Pattern matching* digunakan untuk mencocokan pola pertanyaan yang telah di*input*kan.

2.4.1 Algoritma Boyer-Moore

Algoritma *Boyer Moore* diperkenalkan oleh Bob Boyer dan J.S. Moore pada tahun 1977. Pada Metode ini pencocokan kata dimulai dari karakter terakhir

kata kunci menuju karakter awalnya. Jika terjadi perbedaan antara karakter terakhir kata kunci dengan kata yag dicocokkan, maka karakter-karakter dalam potongan kata yang dicocokkan tadi akan diperiksa satu per satu. Hal ini dimaksudkan untuk mendeteksi apakah ada karakter dalam potongan kata tersebut yang sama dengan karakter yang ada pada kata kunci.

Algoritma Boyer Moore termasuk algoritma *string matching* yang paling efisien dibandingkan algoritma-algoritma *string matching* lainnya. Karena sifatnya yang efisien, banyak dikembangkan algoritma *string matching* dengan bertumpu pada konsep algoritma *Boyer Moore*, beberapa di antaranya adalah algoritma *Turbo* BM dan algoritma *Quick Search.*

Algoritma *Boyer Moore* menggunakan metode pencocokan *string* dari kanan ke kiri yaitu men-*scan* karakter *pattern* dari kanan ke kiri dimulai dari karakter paling kanan. Algoritma *Boyer Moore* menggunakan dua fungsi *shift* yaitu *good-suffix shift* dan *bad-character shift* untuk mengambil langkah berikutnya setelah terjadi ketidakcocokan antara karakter *pattern* dan karakter teks yang dicocokkan (Dwi Purwoko, 2006)**.**

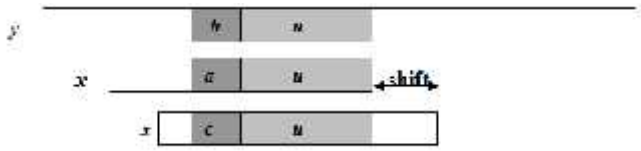
2.4.2 Deskripsi kerja algoritma Boyer Moore

Untuk menjelaskan konsep dari *good-suffix shift* dan *bad-character shift* diperlukan contoh kasus, seperti kasus ketidakcocokan ditengah pencocokan karakter pada teks dan *pattern.* Karakter *pattern* x[i]=a tidak cocok dengan karakter teks $y[i+j]=b$ saat pencocokan pada posisi j. Maka x[i+l .. m-1]= y[$i+j$+1 .. $j+m$-1]=u dan $x[i] \neq y[i+j]$.

2.4.3 Good-suffix Shift

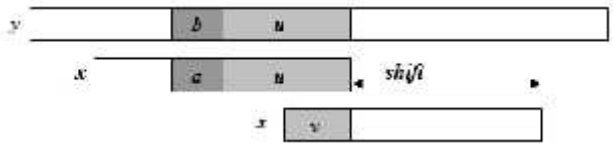
Konsep dari fungsi *good-suffix shift* adalah sebagai berikut:

1. *Good-suffix shift* adalah pergeseran yang dibutuhkan dari $x[i]=a$ ke karakter lain yang letaknya lebih kiri dari $x[i]$ dan terletak di sebelah kiri segmen u. Kasus ini ditunjukkan pada Gambar berikut.



Gambar 2.1 *Good-suffix shift, u* terjadi lagi didahului karakter *c* berbeda dari *a*

2. Jika tidak ada segmen yang sama dengan *u*, maka dicari *u* yang merupakan *suffiks* terpanjang *u*. Kasus ini ditunjukkan pada Gambar berikut.



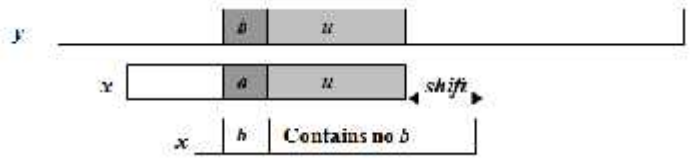
Gambar 2.2 *Good-suffix shift,* hanya *suffix* dari *u* yang terjadi lagi di *pattern x*

2.4.4 *Bad-character shift*

Berdasarkan contoh kasus di atas, *bad-character* adalah karakter pada teks yaitu *y* [*i+j*] yang tidak cocok dengan karakter pada *pattern.*

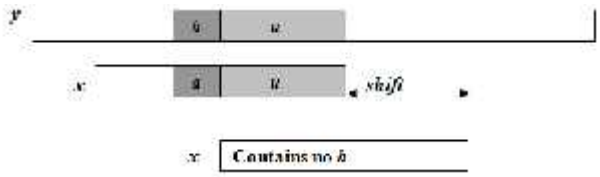
Konsep dari fungsi *bad-character shift* adalah sebagai berikut:

1. Jika *bad-character y*[*i+j*] terdapat pada *pattern* di posisi terkanan *k* yang lebih kiri dari *x*[*i*] maka *pattern* digeser ke kanan sejauh *i-k.* Kasus ini ditunjukkan pada Gambar berikut.



Gambar 2.3 *Bad-character shift,* b terdapat di *pattern* x

2. Jika *bad-character y*[*i+j*] tidak ada pada *pattern* sama sekali, maka *pattern* digeser ke kanan sejauh *i*. Kasus ini dit tunjukkan pada Gambar berikut.



Gambar 2.4 *Bad-character shift,* b tidak ada di pattern x

3. Jika *bad-character y*[*i+j*] terdapat pada *pattern* di posisi terkanan k yang lebih kanan dari *x*[*i*] maka *pattern* seharusnya digeser sejauh *i-k* yang hasilnya negatif *(pattern* digeser kembali ke kiri). Maka bila kasus ini terjadi. akan diabaikan.

Pada kasus ketidakcocokan di atas, algoritma akan membandingkan langkah yang diambil oleh fungsi *good-suffix shift* dan *bad-character shift* di mana langkah yang paling besar yang akan digunakan.

2.4.5 Cara Kerja Algoritma *Boyer-Moore*

Cara kerja dari algoritma *Boyer Moore* adalah sebagai berikut:

1. Menjalankan prosedur preBmBc dan preBmGs untuk mendapatkan inisialisasi.
   a. Menjalankan prosedur preBmBc. Fungsi dari prosedur ini adalah untuk menentukan berapa besar pergeseran yang dibutuhkan untuk mencapai karakter tertentu pada *pattern* dari karakter *pattern* terakhir/terkanan. Hasil dari prosedur preBmBc disimpan pada tabel BmBc.
   b. Menjalankan prosedur preBmGs. Sebelum menjalankan isi prosedur ini, prosedur *suffix* dijalankan terlebih dulu pada *pattern*. Fungsi dari prosedur *suffix* adalah memeriksa kecocokan sejumlah karakter yang dimulai dari karakter terakhir/terkanan dengan sejumlah karakter yang dimulai dari setiap karakter yang lebih kiri dari karakter terkanan tadi. Hasil dari prosedur *suffix* disimpan pada tabel *suff*. Jadi *suff*[i] mencatat panjang dari *suffix* yang cocok dengan segmen dari *pattern* yang diakhiri karakter ke-i.
   c. Dengan prosedur preBmGs, dapat diketahui berapa banyak langkah pada *pattern* dari sebeuah segmen ke segmen lain yang sama yang letaknya lebih kiri dengan karakter di sebelah kiri segmen yang berbeda. Prosedur preBmGs menggunakan tabel *suff* untuk mengetahui semua pasangan segmen yang sama. Contoh pada Gambar 2.1, yaitu berapa langkah yang dibutuhkan dari *au(u* = segmen, *a* =

karakter di sebelah kiri *u*) ke *cu* yang mempunyai segmen *u* pada *pattern* dengan karakter di sebelah kiri segmen yaitu *c* berbeda dari *a* dan terletak lebih kiri dari *au*. Hasil dari prosedur preBmGs disimpan pada tabel BmGs.

2. Dilakukan proses pencarian *string* dengan menggunakan hasil dari prosedur preBmBc dan preBmGs yaitu tabel BmBc dan BmGs.

Berikut ini diberikan contoh untuk menjelaskan proses inisialisasi dari algoritma *Boyer Moore* dengan *pattern* **gcagagag** yang akan dicari pada *string* **gcatcgcagagagtatacagtacg**.

1. Dengan prosedur preBmBc, didapatkan jumlah pergeseran pada *pattern* yang dibutuhkan untuk mencapai karakter a,c,g,t dari posisi terkanan. Berdasarkan contoh diketahui untuk mencapai masing-masing karakter tadi dibutuhkan pergeseran sebanyak 1, 6, 2 dan 8.
2. Dengan prosedur preBmGs, dijalankan prosedur *suffix* terlebih dulu. Dengan prosedur *suffix* akan diketahui:

   suff[0] = 1, 1 karakter g posisi 7 cocok dengan 1 karakter g posisi 0.

   suff[1] = 0, karakter g posisi 7 tidak cocok dengan karakter c posisi 1.

   suff[2] = 0, karakter g posisi 7 tidak cocok dengan karakter a posisi 2.

   suff[3] = 2, 2 karakter dimulai dari karakter g posisi 7 cocok dengan 2 karakter dimulai dari karakter g posisi 3, yang artinya karakter a,g posisi 6,7 cocok dengan karakter a,g posisi 2,3.

   suff[4] = 0, karakter g posisi 7 tidak cocok dengan karakter a posisi 4.

   suff[5] = 4, 4 karakter dimulai dari karakter g posisi 7 cocok dengan 4 karakter dimulai dari karakter 5, artinya karakter a,g,a,g posisi 4,5,6,7 cocok dengan karakter a,g,a,g posisi 2,3,4,5.

   suff[6] = 0,karakter g posisi 7 tidak cocok dengan karakter a posisi 6.

   suff[7] = 8, 8 karakter g,c,a,g,a,g,a,g posisi 0,1,2,3,4,5,6,7 cocok dengan 8 karakter g,c,a,g,a,g,a,g posisi 0,1,2,3,4,5,6,7.

3. Dengan prosedur BmGs akan didapatkan:

0	1	2	3	4	5	6	7
g	c	a	g	a	g	a	g

bmGs[0]= 7, karakter ke-0 g adalah karakter sebelah kiri segmen cagagag.Tidak ada segmen cagagag lain dengan karakter sebelah kiri bukan g maka digeser 7 langkah.

bmGs[1]= 7, karakter ke-1 c adalah karakter sebelah kiri segmen agagag. Tidak ada segmen agagag lain dengan karakter sebelah kiri bukan c maka digeser 7 langkah.

bmGs[2]= 7, karakter ke-2 a adalah karakter sebelah kiri segmen gagag. Tidak ada segmen gagag lain dengan karakter sebelah kiri bukan a maka digeser 7 langkah.

bmGs[3]= 2. karakter ke-3 g adalah karakter sebelah kiri segmen agag. Karena ada segmen agag posisi 2,3,4,5 dengan karakter sebelah kiri bukan g yaitu c posisi 1 maka digeser 2 langkah.

bmGs[4]= 7, karakter ke-4 a adalah karakter sebelah kiri segmen gag. Karena tidak ada seamen gag lain dengan karakter sebelah kiri bukan a maka digeser 7 langkah.

bmGs[5]= 4. karakter ke-5 g adalah karakter sebelah kiri seamen ag. Karena ada segmen ag posisi 2,3 dengan karakter sebelah kiri bukan g yaitu c posisi 1 maka digeser 4 langkah.

bmGs[6]= 7, karakter ke-6 a adalah karakter sebelah kiri segmen yaitu a posisi 7. Karena tidak ada segmen g dengan karakter sebelah kirinya bukan a maka digeser 7 langkah.

bmGs[7]= 1, karakter ke-7 g adalah karakter sebelah kiri segmen dan karena segmen tidak ada maka digeser 1 langkah.

2.4.6 Prosedur Algoritma *Boyer Moore*

Berikut ini adalah prosedure yang ada dalam 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
```

Algoritma 2.1 Algoritma *Boyer Moore* Prosedur preBmBc

```
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
```

Algoritma 2.2 Algoritma *Boyer Moore* Prosedur *suffix*

```
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
```

Algoritma 2.3 Algoritma *Boyer Moore* prosedur preBmGs

```
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;
    endif
    bmBcShift:= BmBc[chartoint(T[i+j])]-n+j+1
    bmGsShift:= BmGs[j]
    shift:= max(bmBcShift, bmGsShift)
    i:= i+shift
```

Algoritma 2.4 Algoritma *Boyer Moore* Prosedur BM

2.5 *Unified Modelling Language* (UML)

Unified Modelling Language (UML) adalah sebuah bahasa yang telah menjadi standar dalam industri untuk visualisasi, merancang dan mendokumentasikan sistem piranti lunak. UML menawarkan sebuah standar untuk merancang model sebuah sistem (Dharwiyanti dan Wahono, 2006).

Untuk merancang sebuah model, UML memiliki beberapa diagram antara lain: *usecase diagram, class diagram, activity diagram, sequence diagram.*

2.5.1 *Usecase Diagram*

Usecase diagram merupakan sebuah gambaran fungsionalitas sebuah sistem. Sebuah *usecase* merepresentasikan interaksi antara aktor dengan sistem. *Usecase* merupakan sebuah pekerjaan tertentu, misalnya login ke sistem, *create* sebuah daftar belanja, dan sebagainya. Seorang/sebuah aktor adalah sebuah entitas manusia atau mesin yang berinteraksi dengan sistem untuk melakukan pekerjaan-pekerjaan tertentu.

Dalam sebuah sistem *usecase diagram* akan sangat membantu dalam hal menyusun *requirment*, mengkomunikasikan rancangan dengan klien dan merancang *test case* untuk semua fitur yang ada pada sistem.

2.5.2 *Class Diagram*

Class adalah sebuah spesifikasi yang jika diinstansiasi akan menghasilkan sebuah objek danmerupakan inti dari pengembangan dan desain berorientasi objek. *Class* menggambarkan keadaan (atribut/properti) suatu sistem, sekaligus menawarkan layanan untuk memanipulasi keadaan tersebut (metoda/fungsi).

Class diagram menggambarkan struktur dan deskripsi *class, package* dan objek beserta hubungan satu sama lain seperti *containment*, pewarisan, asosiasi, dan lain-lain. *Class* memiliki tiga area pokok yaitu, nama, *stereotype* ,atribut dan metoda.

2.5.3 *Sequence Diagram*

Sequence diagram menggambarkan interaksi antar objek di dalam dan di sekitar sistem (termasuk pengguna, *display*, dan sebagainya) berupa *message* yang digambarkan terhadap waktu. *Sequence diagram* terdiri atar dimensi vertikal (waktu) dan dimensi horizontal (objek-objek yang terkait). *Sequence diagram* biasa digunakan untuk menggambarkan skenario atau rangkaian langkah-langkah yang dilakukan sebagai *respons* dari sebuah *event* untuk menghasilkan *output*

tertentu. Diawali dari apa yang memicu aktivitas tersebut, proses dan perubahan apa saja yang terjadi secara internal dan *output* apa yang dihasilkan.

2.5.4 *Activity Diagram*

Activity diagrams menggambarkan berbagai alir aktivitas dalam sistem yang sedang dirancang,bagaimana masing-masing alir berawal, *decision* yang mungkin terjadi, dan bagaimana mereka berakhir. *Activity diagram* juga dapat menggambarkan proses paralel yang mungkin terjadi pada beberapa eksekusi.

Activity diagram merupakan *state diagram* khusus, di mana sebagian besar *state* adalah *action* dan sebagian besar transisi di-*trigger* oleh selesainya *state* sebelumnya (*internal processing*). Oleh karena itu *activity diagram* tidak menggambarkan *behaviour internal* sebuah sistem (dan interaksi antar subsistem) secara eksak, tetapi lebih menggambarkan proses-proses dan jalur-jalur aktivitas dari level atas secara umum.

2.6 Program Berbasis *Web*

Pemrograman berbasis *web*, faktor yang menentukan kinerja aplikasi adalah kecepatan akses *database* dan kecepatan akses jaringan. untuk menjalankan sistem berbasi *web* dibutuhkan engine tertentu yakni *web server*.

2.6.1 *Apache*

Apache merupakan *web server* yang paling sering digunakan sebagai *server* internet dibandingkan *web server* lainnya. *Web server* merupakan *server* internet yang mampu melayani koneksi transfer di dalam protokol *HTTP* (*Hypertext Transfer Protocol*) saat ini *web server* merupakan inti dari *server-server* internet selain *e-mail server*, *ftp server*, dan *news server*.

2.6.2 *PHP*

PHP merupakan singkatan dari *PHP : Hypertext Preprocessor*, PHP adalah bahasa scripting server-side, artinya di jalankan di server, kemudian outputnya dikirim ke client (browser), PHP digunakan untuk membuat aplikasi

web, PHP mendukung banyak database (MySQL, Informix, Oracle, Sybase, Solid, PostgreSQL, Generic ODBC, dll.).

2.6.3 *MySQL*

Bahasa *PHP* mempunyai kelebihan yaitu kompabilitasnya dengan berbagai macam jenis *database*, dukungan dengan berbagai macam jenis sistem operasi. *PHP* lebih cocok dan umum digunakan jika di gabungkan dengan *database MySQL*.

Bahasa SQL pada umumnya informasi tersimpan dalam tabel-tabel yang secara logika merupakan struktur dua dimensi terdiri dari baris (*row* atau *record*) dan kolom (*column* atau *field*). Sedangkan dalam sebuah *database* dapat terdiri dari beberapa *table*.