Implementasi *Pattern Matching* pada *Speech Recognition*

Bella Destiana Junaidi
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung 40132, Indonesia
13516070@std.stei.itb.ac.id

***Abstract*—Speech recognition adalah suatu sistem yang memproses suara dari seseorang menjadi teks. Dalam implementasi *speech recognition*, diperlukan algoritma *pattern matching* yang menyesuaikan kata-kata yang diucapkan oleh seseorang menjadi bentuk tulisan. Pada makalah ini akan dijelaskan bagaimana *pattern matching* dapat digunakan pada *speech recognition*.**

***Keywords—Pattern matching; Speech recognition; text*; *speech*; konversi**

I. Pendahuluan

Pada zaman ini, teknologi telah berkembang cukup pesat, dan salah satu penyebabnya adalah kolaborasi berbagai macam bidang ilmu untuk menciptakan suatu teknologi baru. *Speech recognition* merupakan salah satu teknologi hasil kolaborasi bidang-bidang ilmu tersebut, yakni pada teknik elektro, teknik informatika, dan linguistik.

Seperti teknologi pada umumnya, *speech recognition* telah mengalami perkembangan cukup pesat seiring waktu, dari hanya mengenali sejumlah kata hingga dapat mengenali ribuan kata dalam sekejap. Selain itu, *speech recognition* juga telah tersebar terhadap berbagai macam aspek kehidupan, sehingga tidak jarang ditemukan implementasinya dalam kehidupan sehari-hari(contoh utama: Siri pada iPhone). Meskipun begitu, *speech recognition* masih rentan akan kekurangan dalam hal akurasi, yang bisa diakibatkan oleh faktor eksternal(pembicara kurang jelas dalam hal intonasi ataupun diksi) ataupun faktor internal(pencarian kata yang kurang efektif, konversi suara yang kurang akurat).

Salah satu faktor terpenting dalam *speech recognition* adalah pencocokan dari suara ke kata. Implementasi *speech recognition* yang berbasis pencocokan tidak terlepas dari konsep *pattern matching*, yang menjadi salah satu dasar utama dari perkembangan *speech recognition* tersebut. Seseorang dapat mengucapkan berbagai macam kata dan aplikasi *speech recognition* akan menampilkan kata-kata yang telah diucapkan dari hasil pelacakan menggunakan *pattern matching*. Berhubungan dengan persoalan tersebut, makalah ini akan membahas secara teknis implementasi *pattern matching* dalam *speech recognition*.

II. Dasar Teori

A. Pattern Matching

Algoritma *pattern matching* adalah suatu algoritma yang digunakan untuk menemukan suatu *pattern* tertentu dalam teks yang tersedia. Algoritma ini umumnya terbagi menjadi 3 jenis, yakni algoritma *bruteforce*, algoritma Knuth-Morris-Pratt(KMP), dan algoritma Boyer-Moore.

1. Algoritma *bruteforce*

Algoritma *bruteforce* adalah algoritma *pattern matching* yang paling sederhana, dan pencarian *bruteforce* bersifat *left-to-right*, yakni dari kiri ke kanan. Sebuah *pattern* akan dicocokkan huruf per huruf dalam sebuah teks, dan ketika sebuah huruf dalam *pattern* yang ingin dicocokkan tidak sesuai dengan salah satu huruf dalam teks, maka pencarian akan diulang dan dimulai pada huruf teks selanjutnya.

Contoh ilustrasi *pattern matchizng* dengan algoritma *bruteforce*:

```
Teks: nobody noticed him
Pattern: not

      nobody noticed him
s = 0 not
s = 1  not
s = 2   not
s = 3    not
s = 4     not
s = 5      not
s = 6       not
s = 7        not
```

Algoritma untuk melakukan *pattern matching* secara *bruteforce* adalah sebagai berikut:

```
procedure BruteForceSearch(input m, n : integer, input P : array[1..m] of char, input T : array[1..n] of char, output idx : integer)
{ Mencari kecocokan pattern P di dalam teks T. Jika ditemukan P di dalam T, lokasi awal kecocokan disimpan di dalam peubah idx.
Masukan: pattern P yang panjangnya m dan teks T yang panjangnya n. Teks T direpresentasika sebagai string (array of character)
Keluaran: posisi awal kecocokan (idx). Jika P tidak ditemukan, idx = -1}

Deklarasi
 s, j : integer
 ketemu : boolean

Algoritma
s←0
ketemu←false
while (s ≤ n-m) and (not ketemu) do
    j←1
   while (j ≤ m) and (P[j] = T[s+j]) do
        j←j+1
   endwhile
 { j > m or P[j] ≠ T[s+j] }
   if j = m then { kecocokan string ditemukan }
        ketemu←true
   else
         s←s+1 { geser pattern satu karakter ke kanan teks }
   endif
 endfor
{ s > n - m or ketemu }
 if ketemu then
   idx←s+1 { catatan: jika indeks array dimulai dari 0, idx ← s}
 else
   idx←-1
```

Algoritma ini memiliki kompleksitas sebesar O(mn) pada *worst case* dan O(n) pada *best case*, dengan m merupakan jumlah huruf dalam *pattern* dan n merupakan jumlah huruf dalam teks yang dicari.

Best case terjadi ketika sebuah teks tidak pernah sama dengan huruf *pattern* pertama yang dicocokkan, sehingga tidak terjadi pencocokan ulang pada huruf *pattern* berikutnya, contohnya pada string

Contoh *best case* dan *worst case*:

```
Best Case: kompleksitas O(n)
Teks: string with example
Pattern: example

Worst Case: kompleksitas O(mn)
Teks: aaaaaaaaaaaaaaaaaaaab
Pattern: aaaab
```

2. Algoritma Knuth-Morris-Pratt(KMP)

Algoritma Knuth-Morris-Pratt(disingkat KMP) adalah salah satu algoritma *pattern matching* yang dikembangkan untuk mengatasi kekurangan dari *bruteforce* dengan mengembangkan pergeseran yang lebih efisien.

Pada KMP, pergeseran didasarkan pada *prefix* dan *suffix* dari sebuah *pattern*, dengan mencari *prefix*[0..n] yang sama dengan *suffix*[1..n] pada sebuah *pattern.* Ukuran dari *prefix* ini didefinisikan oleh sebuah fungsi pinggiran *b(j)*. Hasil dari perhitungan fungsi ini dituliskan dalam bentuk tabel, dan digunakan sebagai acuan pergeseran pada pencarian yang akan dilakukan.

Algoritma untuk menghitung fungsi pinggiran adalah sebagai berikut:

```
procedure HitungPinggiran(input m : integer, P : array[1..m] of char, output b : array[1..m] of integer)
{ Menghitung nilai b[1..m] untuk pattern P[1..m] }

Deklarasi
k,q : integer

Algoritma
b[1]←0
q←2
k←0
for q←2 to m do
   while ((k > 0) and (P[q] ≠ P[k+1])) do
       k←b[k]
   endwhile
   if P[q]=P[k+1] then
       k←k+1
   endif
b[q]=k
endfor
```

Contoh tabel yang dihasilkan dari algoritma fungsi pinggiran:

Tabel 1. Contoh tabel fungsi pinggiran dengan *pattern* P = “abaaba”

j	0	1	2	3	4	5

P[j]	a	b	a	a	b	a
b(j)	0	0	1	1	2	3

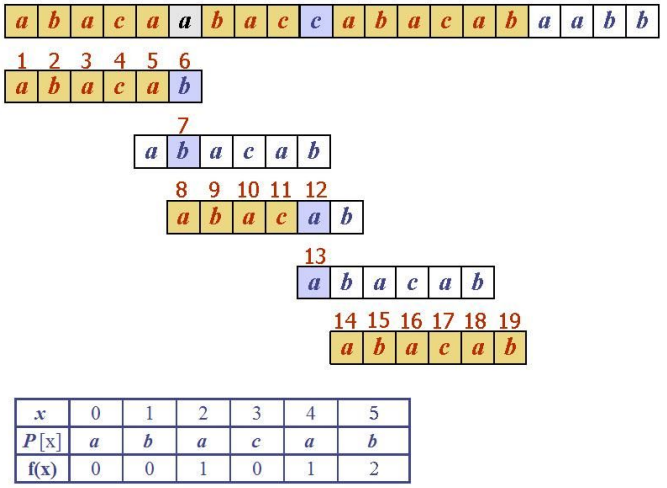
j: indeks *pattern*
P[j]: elemen pada indeks tabel(karakter)
b(j): hasil fungsi pinggiran

Setelah fungsi pinggiran dihitung, maka dapat dilakukan pencarian berdasarkan teks, dengan algoritma berikut:

```
procedure KMPsearch(input m, n : integer, input P : array[1..m] of char, input T : array[1..n] of char, output idx : integer)
{ Mencari kecocokan pattern P di dalam teks T dengan algoritma Knuth-Morris-Pratt. Jika ditemukan P di dalam T, lokasi awal kecocokan disimpan di dalam peubah idx.
Masukan: pattern P yang panjangnya m dan teks T yang panjangnya n. Teks T direpresentasika sebagai string (array of character)
Keluaran: posisi awal kecocokan (idx). Jika P tidak ditemukan, idx = -1. }

Deklarasi
i, j : integer
ketemu : boolean
b : array[1..m] of integer
procedure HitungPinggiran(input m : integer, P : array[1..m] of char, output b : array[1..m] of integer)
{ Menghitung nilai b[1..m] untuk pattern P[1..m] }
Algoritma
HitungPinggiran(m, P, b)
j←0
i←1
ketemu←false
while (i ≤ n and not ketemu) do
      while((j > 0) and (P[j+1]≠T[i])) do
            j←b[j]
      endwhile
      if P[j+1]=T[i] then
            j←j+1
      endif
      if j = m then
            ketemu←true
      else
            i←i+1
      endif
endwhile
```

```
if ketemu then
idx←i-m+1 { catatan: jika indeks array dimulai dari 0, maka idx←i-m }
else
      idx←-1
endif
```



x	0	1	2	3	4	5
P[x]	a	b	a	c	a	b
f(x)	0	0	1	0	1	2

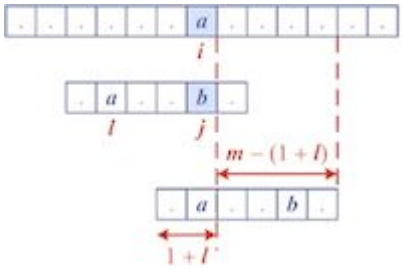
Gambar 1. Ilustrasi pencarian *pattern* berdasarkan KMP, dengan 19 jumlah pencarian

Pada ilustrasi di atas, pencarian pertama dilakukan dan terjadi ketidaksesuaian pada indeks ke-5(huruf ke-6) dalam *pattern*, dan indeks ke-5 memiliki nilai fungsi pinggiran sebesar 2, sehingga *pattern* digeser sehingga huruf ke-2 sejajar dengan huruf teks yang tidak cocok dari pencarian *pattern* sebelumnya.
Karena huruf *pattern* tidak cocok sejak awal dan nilai fungsi pinggirannya 0, dilakukan pergeseran seluruh *pattern* sebanyak 1 huruf. *Pattern* kemudian dicocokan ulang lagi, dan ditemukan ketidak sesuaian dalam indeks ke-4(huruf ke-5), sehingga *pattern* digeser hingga huruf ke-1 sejajar dengan huruf yang tidak sesuai dengan pencocokan sebelumnya, dan dilakukan pergeseran seluruh *pattern* sekali lagi akibat huruf awal *pattern* dan teks tidak sesuai.
Algoritma KMP memiliki kompleksitas algoritma sebesar O(m+n), dengan O(m) merupakan kompleksitas perhitungan fungsi pinggiran dan O(n) merupakan kompleksitas pencarian. Algoritma KMP memiliki keuntungan dalam hal algoritma tersebut tidak pernah perlu untuk mundur dalam pencarian, namun kurang efisien digunakan pada huruf yang memerlukan banyak karakter.

3. Algoritma Boyer-Moore

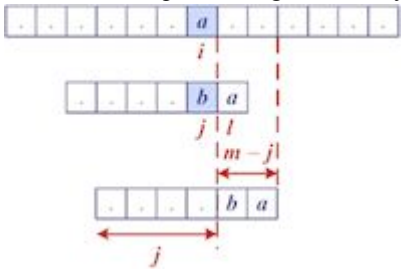
Algoritma Boyer-Moore adalah algoritma *pattern matching* berdasarkan *right-to-left*, yakni pencarian dimulai dari huruf paling belakang pada *pattern*, tidak seperti *bruteforce* atau KMP yang memulai pencarian dari huruf paling depan pada *pattern*.
Pada algoritma Boyer-Moore, pencarian *pattern* dibagi menjadi 3 kasus, yakni:

1. Huruf teks tidak sesuai dengan huruf *pattern*, namun huruf teks tersebut terdapat pada *pattern* pada posisi sebelum huruf teks
Pada kasus ini, dilakukan pergeseran *pattern* ke kanan sehingga huruf *pattern* tersebut sejajar dengan huruf teks yang terdapat dalam pattern.



Gambar 2: Ilustrasi pergeseran kasus pertama

2. Huruf teks tidak sesuai dengan huruf *pattern* dan huruf *pattern* terdapat pada teks tersebut, namun pergeseran ke kanan tidak dimungkinkan karena posisi huruf *pattern* terdapat setelah huruf teks tersebut.
Pada kasus ini, *pattern* digeser sebanyak 1 huruf ke kanan.



Gambar 3: Ilustrasi pergeseran kasus kedua

3. Ketika 2 kasus tersebut tidak berlaku, geser *pattern* sehingga huruf awal *pattern* terdapat pada huruf setelah huruf teks yang tidak sesuai dengan *pattern*.

Untuk mempermudah pergeseran, algoritma Boyer-Moore memiliki sebuah fungsi yang menghitung kemunculan terakhir sebuah kata dalam *pattern*, yang dinotasikan dalam *L(x)*, dengan *x* merupakan huruf dalam *pattern*. Fungsi ini didasarkan oleh alfabet yang disediakan saat pencarian, dengan alfabet yang tidak ada dalam *pattern* diberikan nilai -1.

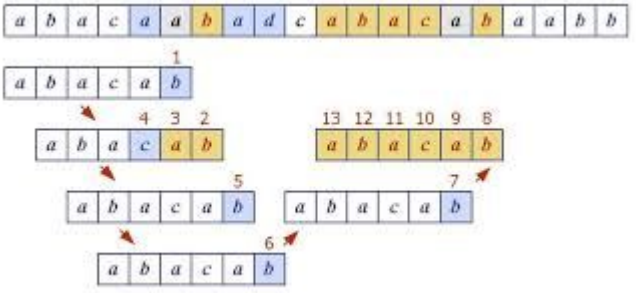
Tabel 2. Contoh tabel kemunculan terakhir pada string “abaac” dengan alfabet {a,b,c,d}
(indeks dimulai dari 0)

x	a	b	c	d
L(x)	3	1	4	-1

Pseudocode dari algoritma Boyer-Moore dituliskan sebagai berikut:

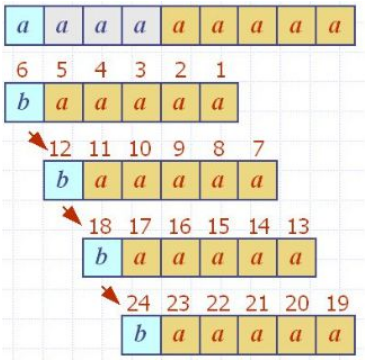
```
BoyerMooreMatch(T,P,A)
  Input:
text T dengan panjang n, pattern P dengan
panjang m, alphabet A
  Output:
index awal sebuah huruf sebuah substring
dalam T yang sama dengan P, -1 jika tidak
ada
Algoritma
//fungsi untuk indeks kemunculan terakhir
  L=lastOccurenceFunction(P,A)
// mulai dari akhir pattern
  i=m-1
  j=m-1
  repeat
    if T[i]=P[j] then
       if j=0 then
          return i
        // substring ditemukan
       else
          i=i-1, j=j-1
       end if
    else
     // pergeseran pattern
       i=i+m-min(j,1+L[T[i]])
       j=m-1
    end if
  until i≥n
  return -1 // pattern tidak ditemukan
```

Contoh implementasi dari algoritma Boyer-Moore:



Gambar 4: Ilustrasi pencarian berdasarkan Boyer-Moore

Algoritma Boyer-Moore memiliki kompleksitas algoritma sebesar O(mn+A) pada *worst case*, dengan m merupakan panjang *pattern*, n merupakan panjang teks, dan A merupakan jumlah alfabet. Seperti pada algoritma *bruteforce*, Boyer-Moore memiliki *worst case* ketika *pattern* memiliki ketidaksesuaian hanya pada akhir pencarian, namun dalam segi waktu, Boyer-Moore jauh lebih cepat daripada *bruteforce* karena pergeseran yang lebih efisien, dengan efisiensi Boyer-Moore lebih terlihat ketika menggunakan banyak variasi karakter dalam teks.

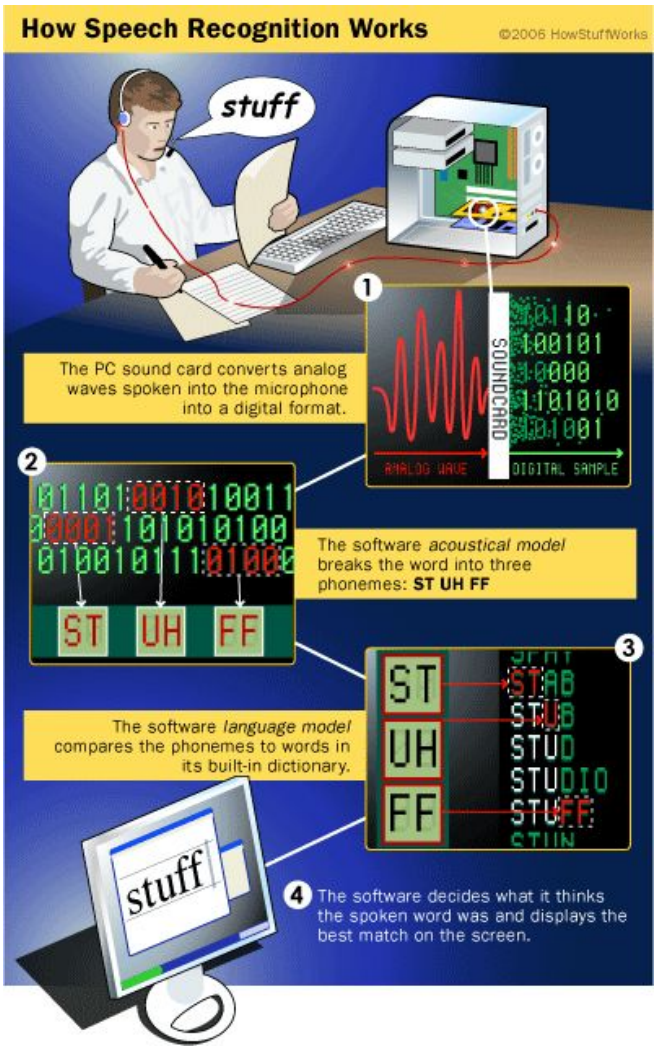


Gambar 5: Ilustrasi *worst case* pada algoritma Boyer-Moore

III. Analisis Persoalan

Sistem *speech recognition* masa kini cenderung memiliki 2 bagian utama, yakni bagian *front-end* dan bagian *back-end*.
Bagian *front-end* adalah bagian yang menangani bagaimana suara dikonversi menjadi sinyal digital oleh *analog-digital converter*, sedangkan bagian *back-end* suatu sistem *speech recognition* adalah bagian terdiri atas 2 bagian, yakni *acoustic model*, *language model*, dan *lexicon*.
Acoustic model adalah bagian yang mencocokkan suara yang dihasilkan dari kata-kata yang diucapkan dengan suku kata yang menyerupai suara tersebut, dan *language model* adalah bagian yang menyesuaikan suku kata dari *acoustic modeling* menjadi kata-kata yang sesuai dengan yang diucapkan, sedangkan *lexicon* berfungsi sebagai kamus dari sistem *speech recognition*, dengan menyediakan kata-kata dan cara pengucapannya berdasarkan suku kata.
Implementasi dari bagian-bagian *back-end* tersebut menggunakan *pattern matching* untuk menyesuaikan agar teks yang muncul sesuai dengan ucapan pengguna sistem *speech recognition* tersebut.

Langkah-langkah dalam *speech recognition* adalah sebagai berikut:
1. Pengguna mengucapkan sebuah perkataan ke dalam suatu sistem yang memiliki *speech recognition*(contoh: komputer, iPhone, dsb.)
2. Sistem akan melakukan penerjemahan signal audio yang diterima dari ucapan pengguna(yang merupakan signal analog) menjadi signal digital.
3. Pencocokkan dilakukan oleh *acoustic model* pada signal digital tersebut, sehingga dihasilkan beberapa suku kata yang sesuai.
4. Pencocokkan suku kata oleh *language model* sehingga membentuk suatu ucapan.



Gambar 6: Ilustrasi tahapan *speech recognition*

A. Acoustic Model

Acoustic model adalah salah satu bagian hasil implementasi *pattern matching* pada sistem *speech recognition* yang berhubungan dengan suara hasil ucapan dan suku kata yang dimiliki.
Pada *acoustic model*, sinyal *digital* yang dihasilkan dari konversi suara analog dari pembicara berbentuk sebuah *string* berbasis bit, dengan ukuran bervariasi berdasarkan jenis *A/D converter* yang digunakan(umumnya 16-bit), dan sistem telah memiliki berbagai macam suku kata dengan *pattern* bit tertentu.
Pencocokan dilakukan terhadap seluruh suku kata yang tersedia dalam sistem tersebut. Ketika suatu *pattern* yang menandakan sebuah suku kata ditemukan, maka sinyal akan langsung dikonversi menjadi suku kata tersebut dan pencarian dalam teks dilanjutkan pada titik setelah itu, dan jika tidak ditemukan, maka *pattern* suku kata akan diganti dengan *pattern* suku kata lain.

Pseudocode algoritma pencocokkan *acoustic model* dapat dirumuskan sebagai berikut:

```
BitConverter(L,T)
//melakukan konversi string bit audio ke suku kata

Deklarasi
//tipe bentukan suku kata dan kodenya
type kode_suku struct {
// suku kata
   suku: string
//kode suku
   kode: string }
Input:
//array kode suku kata
L: array of kode_suku
//teks yang dicari
T: string

Output:
//suku kata yang ditemukan
words: array of string

Fungsi:
//algoritma pattern matching, output bool
search(pattern,text)

Algoritma
//teks sementara untuk pencarian
newtext = T
words = []
foreach kode_suku sukukata in L
//jika sukukata ada pada teks, tambahkan ke sukuata yang ditemukan
    if(search(sukukata.kode,T))
       add sukukata.nama to words
//hapus sukukata yang telah ditemukan dalam
       newtext = newtext - sukukata.kode
   endif
endfor
//suku kata ditemukan
if newtext is not empty
       return words
else
//suku kata tidak ditemukan
       return null
```

Contoh:
Sebuah sistem *sound recognition* memiliki diksi suku kata sebagai berikut:

Tabel 3: Tabel kode suku kata *speech recognizer*

Suku kata	***Kode***
ma	01101101 01100001
ra	01110010 01100001
sa	01110011 01100001
kan	01101011 01100001 01101110

Dan terdapat hasil dari *converter* sebagai berikut:
01101101 01100001 01101011 01100001 01101110

Hasil pencocokkan:

Pattern 1 - suku kata ma
Teks: **01101101 01100001** 01101011 01100001 01101110
Pattern: **01101101 01100001**
Hasil: Ditemukan
Teks saat ini: 01101011 01100001 01101110
Kata teks: ma

Pattern 2 - suku kata ra
Teks: 01101011 01100001 01101110
Pattern: 01110010 01100001
Hasil: Tidak ditemukan
Teks saat ini: 01101011 01100001 01101110
Kata teks: ma

Pattern 3 - suku kata sa
Teks: 01101011 01100001 01101110
Pattern: 01110011 01100001
Hasil: Tidak ditemukan
Teks saat ini: 01101011 01100001 01101110
Kata teks: ma

Pattern 4 - suku kata kan
Teks: **01101011 01100001 01101110**
Pattern: **01101011 01100001 01101110**
Hasil: Ditemukan
Teks saat ini: -
Kata teks: ma, kan

Dari hasil di atas, didapatkan kesimpulan bahwa *input* sinyal digital 01101101 01100001 01101011 01100001 01101110 menghasilkan suku kata ma dan kan.

B. Language model

Tahap selanjutnya dalam *speech recognition* setelah *acoustic model* adalah *language model*, yang memproses suku kata yang telah ditemukan menjadi sebuah kata atau kalimat yang ditampilkan dalam hal tersebut.
Pada *language model*, tersedia sebuah kamus(*lexicon*) yang

digunakan untuk mencocokkan suku-suku kata yang telah tersedia. Kata-kata dalam kamus tersebut menjadi teks yang akan dicari *pattern*-nya, dan suku kata yang ditemukan dari *acoustic model* menjadi *pattern* yang akan dicari, dengan jumlah *pattern* yang akan dicari sesuai dengan jumlah *pattern* yang dihasilkan sebelumnya.. Jika semua suku kata berhasil ditemukan dan membentuk suatu kata atau kalimat, maka kata atau kalimat yang dihasilkan akan ditampilkan pada layar.

Pseudocode algoritma pencocokkan *language model* dapat dirumuskan sebagai berikut:

```
TextConverter(D,S)
//melakukan konversi suku kata menjadi sebuah kata

Deklarasi
//tipe bentukan kata kamus
type dictwords struct {
// nama kata
   nama: string
//suku kata cara pelafalan
   lafal: string }

Input:
//kamus kata
D: array of dictwords
//suku kata yang ditemukan
S: array of string

Output:
//kata yang ditampilkan
text: string

Fungsi:
//fungsi pattern matching, output bool
search(pattern,text)
//fungsi pemecah kata, output array of string
tokenize(text)


Algoritma
//kata yang ditampilkan
text = []
//iterasi pencarian pada tiap kata di kamus
foreach dictwords dict in D
//suku kata yang ditemukan pada kata kamus
  sukukata_dict = []
//kata sementara untuk pemeriksaan
  temp = S, temp2 = dict.lafal
      foreach var sukukata in temp
//periksa apakah sukukata ada pada pelafalan
        if search(sukukata, temp2) then
//tambah suku kata yang ditemukan
          add sukukata to sukukata_dict
//hapus suku kata yang telah diperiksa untuk mencegah pemeriksaan dobel
          temp2 = temp2-sukukata
        else
//jika suku kata tak sesuai, keluar loop
          break
        endif
      endfor
//periksa kesamaan suku kata yang ditemukan
  if sukukata_dict = tokenize(dict.name)
//tambahkan teks yang ditemukan
      text = text+dict.name
//hapus suku kata yang telah dipakai
      S = S - sukukata_dict
  endif
endfor
return text
```

Contoh:
Melanjutkan contoh di atas, didapatkan 2 suku kata yang menjadi *pattern* yang akan dicari, yakni ma dan kan. Disediakan kamus sebagai berikut:

Tabel 4: Tabel kamus *speech recognizer*

Kata	***Suku kata***
pelan	pe-lan
lama	la-ma
makanan	ma-ka-nan
makan	ma-kan

Hasil pencocokkan:

Kata 1 - pelan
Teks: pe-lan
Pattern 1: ma
Pattern 2: kan
Suku kata sesuai: tidak ada
Hasil: tidak ada

Kata 2 - lama
Teks: la-ma
Pattern 1: ma
Pattern 2: kan

Suku kata sesuai: ma
Hasil: tidak ada(karena hanya 1 suku kata ditemukan)

Kata 3 - makanan
Teks: ma-ka-nan
Pattern 1: ma
Pattern 2: kan
Suku kata sesuai: ma, kan
Hasil: tidak ada(karena jumlah suku kata tidak sesuai)

Kata 4 - makan
Teks: ma-kan
Pattern 1: ma
Pattern 2: kan
Suku kata sesuai: ma, kan
Hasil: makan

Dari hasil di atas, dapat disimpulkan bahwa kata yang akan ditampilkan pada layar melalui pemrosesan suku kata sebelumnya adalah kata "makan".

Efektivitas *pattern matching* dalam *speech recognition*, baik dalam segi kecepatan maupun akurasi, didasarkan pada jenis algoritma yang digunakan pada sistemnya.
Algoritma *bruteforce* cenderung perlu dihindari karena merupakan algoritma yang pelan meskipun bersifat sederhana, dan algoritma KMP dan Boyer-Moore dapat diimplementasikan tergantung pada jenis perbandingan. Ketika perkataan yang memiliki variasi huruf yang besar(contoh: alfabet), Boyer-Moore dapat diaplikasikan, namun ketika perkataan yang akan diubah tidak memiliki banyak variasi huruf(contoh: *binary*), KMP menjadi lebih efektif untuk diterapkan.

C. Implementasi lainnya

Selain 2 contoh utama di atas, implementasi lain *pattern matching* dalam *speech recognition* adalah pada *call center*. Ketika seseorang menghubungi sebuah *call center*, maka akan muncul pesan rekaman yang menjelaskan instruksi mengenai apa yang dapat dilakukan. Jika seseorang melakukan *input* lisan sesuai dengan instruksi, maka aksi sesuai dengan penjelasan *call center* tersebut akan dipanggil. Aksi-aksi berdasarkan nomor tersebut telah dikumpulkan dalam sebuah lokasi tertentu bernama *domain*.

Dalam implementasi sesungguhnya, kebanyakan *speech recognition* menggabungkan berbagai macam ilmu lainnya untuk menghasilkan pencarian yang lebih efisien, namun *pattern matching* tetap menjadi dasar utama dari ilmu-ilmu pencarian tersebut.

IV. Kesimpulan

Berdasarkan analisis di atas, didapatkan kesimpulan bahwa *pattern matching* pada *speech recognition* digunakan pada 2 tahapan, yakni konversi *digital audio* oleh *acoustic model* dan konversi suku kata oleh *language model*. *Pattern* yang dicocokkan dalam tahap *acoustic modeling* adalah *string* dalam bentuk bit, karena *digital audio* umumnya tersusun berdasarkan bit, dan *pattern* pada tahap *language modeling* adalah *string* dalam bentuk alfabet, yang dikonversi dari tahap *acoustic modeling,* dan karena sifat alfabet yang berbeda pada 2 pencarian tersebut, penerapan algoritma *pattern matching* disesuaikan berdasarkan jumlah alfabet yang ada.

Ucapan Terima Kasih

Melalui makalah ini, penulis mengucapkan terima kasih terhadap Tuhan Yang Maha Esa sehingga makalah ini dapat diselesaikan tepat pada waktunya, terhadap orangtua dan keluarga yang telah mendukung studi penulis di Teknik Informatika hingga saat ini, dan terhadap dosen-dosen mata kuliah Strategi Algoritma, yakni Bapak Rinaldi Munir, Ibu Masayu Leylia Khodra, dan Ibu Nur Ulfa Maulidevi, yang memberikan tugas makalah ini untuk mengajarkan mahasiswa bagaimana cara menerapkan ilmu yang telah dipelajari untuk mengatasi persoalan kehidupan sehari-hari.

Referensi

[1] Munir, Rinaldi. 2003. *Diktat Strategi Algoritma*. Bandung: Departemen Teknik Informatika.

[2] Munir, Rinaldi (2006). *Pattern Matching*.

[3] http://www.cis.uoguelph.ca/~xli/courses/cis2520/c16.pdf
diakses pada 11 Mei 2018 pukul 17.40

[4] http://www-igm.univ-mlv.fr/~lecroq/string/node14.html
diakses pada 11 Mei 2018 pukul 17.42

[5] https://electronics.howstuffworks.com/gadgets/high-tech-gadgets/speech-recognition1.htm
diakses pada 11 Mei 2018 pukul 17.44

[6] https://searchcrm.techtarget.com/definition/speech-recognition
diakses pada 11 Mei 2018 pukul 19.39

[7] https://msdn.microsoft.com/en-us/library/hh378337(v=office.14).aspx
diakses pada 11 Mei 2018 pukul 20.27

[8] http://www.explainthatstuff.com/voicerecognition.html
diakses pada 11 Mei 2018 pukul 20.29

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 11 Mei 2018

Bella Destiana Junaidi
13516070