IMPLEMENTASI ALGORITMA PENCOCOKAN STRING DALAM APLIKASI PENGENAL MUSIK

Melvin Fonda / 13512085
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
13512085@std.stei.itb.ac.id

ABSTRAK

Algoritma pattern matching telah menjadi suatu alat yang sangat penting dalam analisa musik dan pencarian informasi. Berbagai cara telah dicari dalam beberapa tahun belakangan dan menghasilkan beberapa algoritma yang efisien. Dalam makalah ini akan dibahas tentang algoritma pencocokan string dalam aplikasi pengenal musik. Sehingga pencarian musik dapat dilakukan dengan mudah dan efisien.

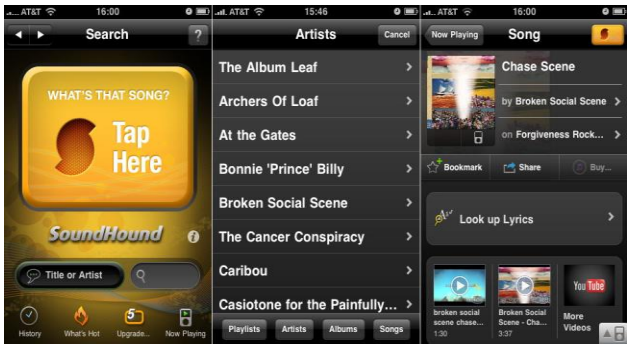
I. PENDAHULUAN

Musik merupakan kumpulan dari nada-nada yang disusun sedemkian rupa sehingga mengandung irama, lagu, dan keharmonisan terutama suara yang dihasilkan dari alat – alat yang menghasilkan irama. Musik sudah dikenal sejak lama yaitu sejak 180.000 hingga 100.000 tahun yang lalu. Dan sekarang , sudah banyak sekali berbagai macam musik dan bahasa yang berkembang dan tumbuh di dunia.

Dalam perjalanannya, perkembangan musik tidak dapat lagi dibendung lagi jumlahnya. Setiap tahun kira-kira 2500 buah lagu diciptakan yang berarti bahwa setiap harinya dapat tercipta kurang lebih 8 buah lagu berbeda di seluruh penjuru dunia. Yang menjadi salah satu permasalahan yang sering dialami para penikmat musik di dunia ini salah satunya adalah terkadang kita ingin mengetahui judul lagu yang hanya "numpang lewat" di radio tetapi kita hanya mengetahui nada dari lagu tersebut. Bermula dari hal ini, para developer aplikasi berlomba-lomba membuat suatu aplikasi untuk menganalisa musik.

Analisa musik merupakan sesuatu yang masih baru dan mulai memanfaatkan suatu teknik dari ilmu *computer science*. Dengan seiring bertambah banyaknya lagu-lagu yang ada di dunia, makin dibutuhkannya sebuah aplikasi yang dapat digunakan untuk menganalisa dan mencari musik. Teknik yang digunakan tidak hanya harus tepat , tatapi juga harus se-efisien mungkin.

Beberapa aplikasi penganalisa musik contohnya adalah Soundhound, Shazam, Midomi dan masih banyak lagi aplikasi lain yang serupa dengan kedua aplikasi tersebut.



Gambar 1. Contoh aplikasi pendeteksi lagu Soundhound

Aplikasi – aplikasi ini dapat memberikan suatu sensasi yang baru untuk para penikmat musik di seluruh dunia.Anda dapat langsung mengenali sebuah lagu yang menarik perhatian anda pada saat anda duduk dan bersantai di sebuah kafe, atau ketika anda sedang mendengarkan musik yang menyentuh hati anda di sebuah radio yang anda putar saat menyetir pulang dari kantor. Hanya dengan satu kali sentuh dan aplikasi pengenal musik dan anda terkoneksi ke internet anda dapat mencari lagu apapun yang anda inginkan tanpa perlu mengingat-ingat liriknya.

Pengenal musik ini juga dapat mengenali berbagai jenis lagu dari berbagai jenis negara dengan bermacam – macam bahasanya. Bagaimana bisa ? Hal ini menjadi mungkin dilakukan dengan pemanfaatan pattern matching dari nada lagu (yang seharusnya unik) dari setiap lagu. Sehingga hanya dengan mendengungkan nada – nada yang kita tangkap dari lagu yang ingin kita cari, kita dapat langsung menemukan sebuah lagu dengan judul, nama penyanyi , album, lirik dari lagu tersebut, beserta seluruh deskripsi dari lagu tesebut.

II. TEORI DASAR

A. ALGORITMA STRING MATCHING

String mathcing adalah algoritma yang dipakai untuk mencocokan suatu teks terhadap teks lain yang diiinginkan. Beberapa Algoritma yang populer untuk memecahkan persoalan string match adalah :

1. BRUTE FORCE

Algoritma brute force merupakan cara yang paling sederhana untuk memecahkan masalah string match. Cara kerja algoritma ini dengan mencoba setiap posisi pattern lalu dilanjutkan dengan mencocokan tiap tiap karakter dalam teks tersebut hingga ditemukan sebuah posisi yang dan teks yang cocok dengan teks yang diinginkan.

Berikut adalah psedocode dari algoritma brute force :

```
PencocokanStringBruteForce(T[0..n-1] , P[0..m-1)

//input larik T[0..n-1] dengan ukuran n karakter merepresentasikan teks dan larikP[0..m-1] dengan ukuran m karakter merepresentasikan pattern
//Output : indeks dari karkter pertama dalam teks yang dimulai dengan substring yang bersesuaian atau -1 jika tidak ditemukan pattern dalam teks

Kamus
Integer i , j ;

Algoritma
For i == 0 tp n-m do
    j = 0
    While j < m and P[j] = T[i+j] do
        j = j + 1
     if j == m
        return i
return -1
```

Contoh pengaplikasian algoritma brute force pada string matching :

Gambar 2. Algoritma pencocokan string menggunakan brute force

2. ALGORITMA BOYER – MOORE

Algoritma pencarian string, dipublikasikan oleh Robert S. Boyer, dan J. Strother Moore pada 1977. Algoritma ini dianggap sebagai salah satu algoritma yang paling efisien pada aplikasi umum.Tidak seperti algoritma pencarian string yang ditemukan sebelumnya, algoritma Boyer-Moore mulai mencocokkan karakter dari sebelah kanan pattern. Ide di balik algoritma ini adalah bahwa dengan memulai pencocokan karakter dari kanan, dan bukan dari kiri, maka akan lebih banyak informasi yang didapat.

Cara kerja algoritma Boyer – Moore dapat dimisalkan dengan usaha pencocokan yang terjadi pada teks [i..i + n – 1], dan anggap ketidakcocokan pertama pattern terjadi di antara teks [i+j] dan pattern j, dengan 0<n Berarti, teks[i+j+ 1..i + n-1] = pattern [j+1..n-1] dan a = teks[i+j] tidak sama dengan b = pattern[j]. Jika u adalah akhiran dari pattern sebelum b dan u adalah sebuah awalan dari pattern, maka penggeseran-penggeseran yang mungkin adalah:

1. Penggeseran good-suffix yang terdiri dari menyejajarkan potongan teks[i+j+1..i+n-1] = pattern [j+1..n-1] dengan kemunculannya paling kanan di pattern yang didahului oleh karakter yang berbeda dengan pattern[j]. Jika tidak ada potongan seperti itu, maka algoritma akan menyejajarkan akhiran v dari teks[i+j+1 .. i+n-1] dengan awalan dari pattern yang sama.
2. Penggeseran bad-character yang terdiri dari menyejajarkan teks[i+j] dengan kemunculan paling kanan karakter tersebut di pattern. Bila karakter tersebut tidak ada di pattern, maka pattern akan disejajarkan dengan teks[i+n+1].

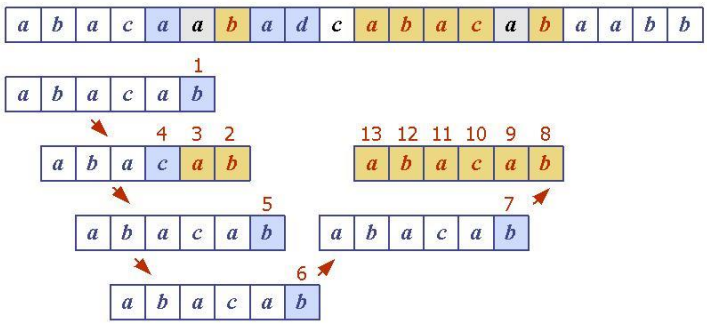
Secara sistematis, langkah-langkah yang dilakukan algoritma Boyer-Moore pada saat mencocokkan string adalah:

1. Algoritma Boyer-Moore mulai mencocokkan pattern pada awal teks.
2. Dari kanan ke kiri, algoritma ini akan mencocokkan karakter per karakter pattern dengan karakter di teks yang bersesuaian, sampai salah satu kondisi berikut dipenuhi:
   - Karakter di pattern dan di teks yang dibandingkan tidak cocok (mismatch).
   - Semua karakter di pattern cocok. Kemudian algoritma akan memberitahukan penemuan di posisi ini.
3. Algoritma kemudian menggeser pattern dengan memaksimalkan nilai penggeseran good-suffix dan penggeseran bad-character, lalu mengulangi langkah 2 sampai pattern berada di ujung teks.

Adapun pseudocode dari algoritma Boyer-Moore adalah sebagai berikut :

```
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
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
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
//Dan berikut adalah pseudocode
algoritma Boyer-Moore pada fase
pencarian:
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
```

Contoh pengaplikasian algoritma Boyer-Moore pada string matching :



Gambar 3. Algoritma pencocokan string dengan Boyer-Moore

3. Algoritma Knuth-Morris-Pratt

Algoritma Knuth Morris Pratt (KMP) dikembangkan oleh D. E. Knuth, bersama dengan J. H. Morris dan V. R. Pratt. Untuk pencarian string dengan algoritma Brute Force , setiap kali ditemukan teks yang tidak cocok, maka pattern akan digeser satu karakter ke kanan. Berbeda dengan algoritma Brute Force, informasi yang digunakan masih dipelihara untuk melakukan jumlah pergeseran. Algoritma menggunakan informasi tersebut untuk membuat pergeseran yang lebih jauh, dan optimal Perbandingan antara algoritma *brute force* dengan algoritma KMP ditunjukkan dengan perpindahan posisi pattern terhadap posisi teks. Dimana dalam hal pencocokan string, pattern yang dicocokkan berawal dari kiri teks.

Kompleksitas algoritma pencocokan string dihitung dari jumlah operasi perbandingan yang dilakukan. Kompleksitas waktu terbaik dari algoritma brute force adalah $O(n)$. Kasus terbaik terjadi jika pada operasi perbandingan, setiap huruf pattern yang dicocokkan dengan awal dari teks adalah sama. Kompleksitas waktu terburuk dari *brute force* adalah $O(mn)$. Jika dibandingkan dengan algoritma brute force, maka algoritma KMP mempunyai kompleksitas algoritma $O(m+n)$.

Berikut adalah prosedur perhitungan fungsi pinggiran KMP :

```
procedure HitungPinggiran(input m :
integer, P : string)
( Menghitung nilai pinggiran b[1..m]
untuk pattern P[1..m] )

Kamus :
  k, q : integer

Algoritma :
  b[] = array[1..m] of integer
  b[0] = 0
  q = 1
  k = 0

  for q ← 1 to m-1 do
    while ((k > 0) and (P[q] ≠ P[k]))
  do
      k ← b[k]
    endwhile

    if P[q] = P[k] then
      k ← k+1
    endif

    b[q] ← k
  endfor
```

Prosedur algoritma KMP :

```
function KMPSearch(input m : integer,
n : integer, P : string, T string) →
boolean
( Mengembalikan true jika pattern P
ditemukan dalam teks T )

Kamus :
  i, j : integer
  ketemu : boolean
  procedure HitungPinggiran(input m :
  integer, P : string)
  ( Menghitung nilai pinggiran b[1..m]
  untuk pattern P[1..m] )

  Algoritma :
    HitungPinggiran(m, P)
    i ← 0
    j ← 0
    ketemu = false

    while ((j > 0) and (P[j] ≠ T[i]))
  do
      j ← b[j-1]
    endwhile

    if P[j] = T[i] then
      j ← j+1
    endif

    if j = m then
      ketemu ← true
    else
      i ← i+1
    endif

    return ketemu
```

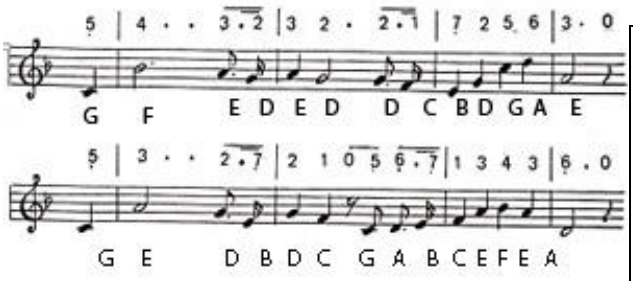
III. ANALISIS & PEMBAHASAN

Secara umum, musik dapat direpresentasikan dengan nada – nada yang dimainkan pada waktu tertentu dan *pitch* tertentu. Atribut lain yang ada dalam musik juga dapat dimasukkan. Subjek tentang representasi musik sendiri sudah menjadi perdebatan selama 20 tahun terakhir. Biasanya parameter dari sebuah lagu meliputi tinggi nada, durasi, volume suara, warna nada, notasi, dan fitur fitur lain.

Dalam makalah ini akan saya bahas lebih dalam tentang pencocokan string melalui notasi dari sebuah lagu. Misalkan nada nada berikut ini yang di asumsi kan telah ditangkap dan nadanya disimpan dalam notasi huruf. Maka akan kita cari apakah bersesuaian dengan lagu tertentu.

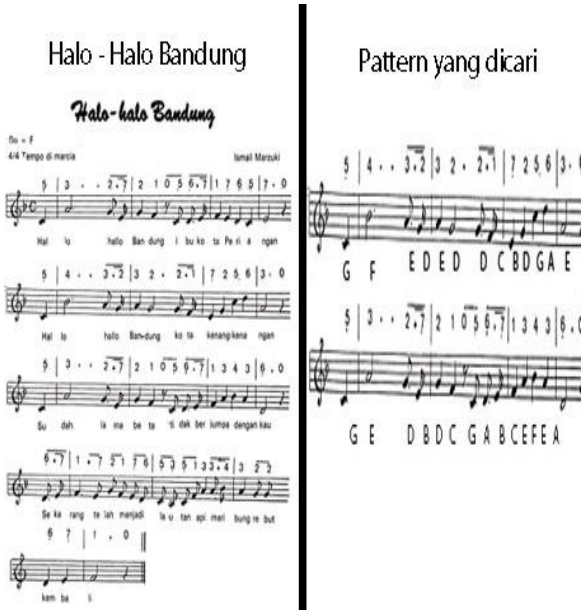
Langkah – langkahnya adalah sebagai berikut :

1. Jadikan nada nada di bawah sebagai pattern.



Pattern
GFEDEDDCBDGAGEDBDCGABCEFEA

2. Dari setiap lagu yang ada di dalam database dilakukan pencarian. Pencarian dilakukan dengan merubah pattern yang didengar melalui microphone dari handphone menjadi sebuah "fingerprint" yang kemudia akan dibandingkan dengan database pusat untuk pencocokan.
3. Jika sudah ditemukan lagu yang akan dicocokan dengan pattern maka dilakukan pencocokan pattern. Salah satu lagu yang ditemukan adalah "Halo Halo Bandung".



4. Pattern ini akan dicocokkan menggunakan algoritma Boyer - Moore

Ternyata setelah dilakukan pencocokan string, pattern yang dicari bersesuaian dengan lagu "Halo – Halo Bandung" Sehingga aplikasi akan langsung memunculkan deskripsi lengkap lagu "Halo – Halo Bandung".

Algoritma Boyer-Moore dapat ditampilkan dengan pseudocode sebagai berikut:

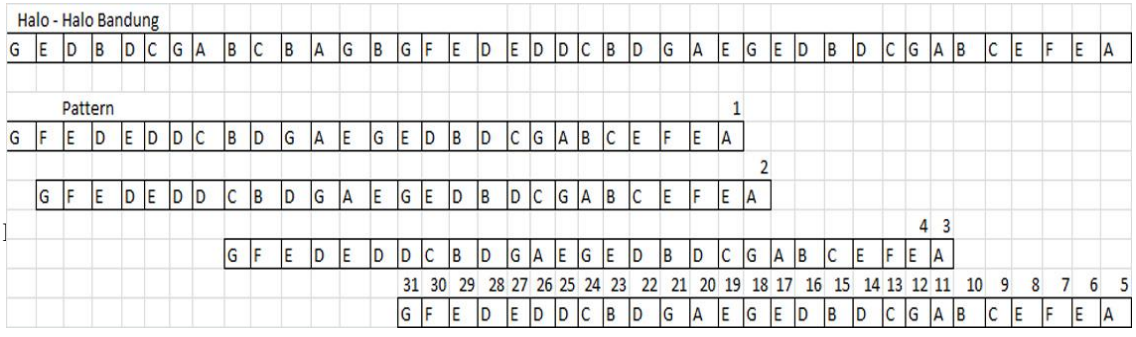
Algoritma PencarianLagu(L, P) -> String
Input: String T (lagu yang akan dicocokan) dan P (pattern) dengan m jumlah karakter
Output: Menghasilkan lagu yang dicocokan jika ditemukan, menampilkan pesan "Tidak Ditemukan" jika lagu tidak ditemukan

```
i <-- m - 1
j <-- m - 1

repeat

if P[j] = T[i] then

        if j = 0 then

                return S {Lagu yang dicari}

        else {cek karakter selanjutnya}

                i <-- i - 1
                j <-- j - 1

else { P[j] <> T[i] pindahkan pattern}

        i <-- i + m - j - 1 {buat i kembali menjadi test
        yang paling awal}
        i <-- i + max(j - last(T[i]), match(j))
        {Pindahkan P relatif terhadap T}
        j <-- m-1

until i > n - 1

return "Tidak Ditemukan"
```

IV. KESIMPULAN

Dalam masa teknologi seperti sekarang ini, pemanfaatan pattern matching sangat dibutuhkan. Berbagai developer IT berlomba – lomba untuk membuat sebuah aplikasi dapat memuaskan penggunanya. Aplikasi pencarian lagu adalah salah satu contohnya. Riset tentang string matching untuk pencocokan lagu pun terus dilakukan agar dapat ditemukannya sebuah cara untuk lebih mengefisiensi pencarian serta dapat meningkatkan keakuratan pencarian.

Penulis berharap dalam perkembangannya, string matching tidak hanya dimanfaatkan dalam bidang musik tetapi juga di bidang lain yang belum "terjamah".



V. ACKNOWLEDGEMENT

Ucapan terima kasih sebesar-besarnya saya tujukan kepada dosen Informatika saya Ibu Dr. Masayu Leylia Khodra dan Bapak Dr. Rinaldi Munir atas bimbingan di dalam maupun luar kelas. Berbagai referensi saya dapatkan dari kuliah ataupun kegiatan di luar kuliah sehingga saya dapat menyelesaikan makalah ini. Saya juga berterima kasih kepada teman – teman dan pihak lain yang telah membantu saya dalam proses pembuatan makalah ini.

REFERENSI

[1] Diktat Kuliah IF3051 Strategi Algoritma. Ir. Rinaldi Munir, M. T.

[2] http://www.tonytorero.com/2011/04/19/battle-of-the-mobile-song-id-apps-shazam-vs-soundhound/. Tanggal akses : 18/05/14

[3] **(Inggris)**Lecroq, Thierry Charras, Christian. 2001. Handbook of Exact String Matching Algorithm. ISBN 0-9543006-4-5

[4] **(Inggris)**Boyer, Robert Moore, J. 1977. A Fast String Searching Algorithm. Comm. ACM 20: 762–772 doi:[1]

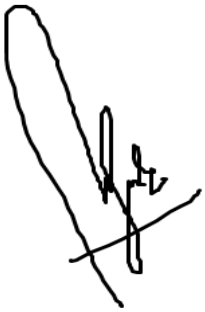
[5] http://www.stoimen.com/blog/2012/03/27/computer-algorithms-brute-force-string-matching/

[6] http://anthon-23510311.blogspot.com/2011/09/tentang-router.html

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, 18 Mei 2014



Melvin Fonda /13512085