Implementasi Algoritma Pattern Matching dalam Aplikasi SoundHound

Timothy Thamrin Andrew H. Sihombing / 13516090
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
13516090@std.stei.itb.ac.id

***Abstrak*—Di dalam dunia pemograman algoritma merupakan hal yang mendasar. Algoritma merupakan cara – cara yang digunakan untuk menyelesaikan permasalahan teretntu dan disusun secara terstruktur dan efektif. Salah satunya adalah algoritma pattern matching. Algoritam ini membantu kita untuk melihat apakah ada kecocokan antara pattern yang kita inginkan dengan sumbernya. Algoritma pattern matching telah menjadi salah satu bagian penting dalam pencarian informasi, analisa di dalam musik salah satunya. String biner dari penggalan lagu yang didapat dari rekaman akan dicocokan untuk mengidentifikasi lagu tersebut dengan mencocokan ke string biner daftar lagu yang ada. Dalam makalah ini akan dibahas lebih lanjut, sehingga pencarian music dapat dilakukan dengan lebih mudan dan efisien.**

Kata kunci—Pattern matching, Boyer Moore, KMP, SoundHound

I. PENDAHULUAN

Musik adalah suara yang disusun demikian rupa sehingga mengandung irama, lagu, dan keharmonisan terutama dari suara yang dihasilkan dari alat-alat yang dapat menghasilkan irama. Musik juga merupakan kumpulan dari nada – nada yang disusun oleh penciptanya sedemikian rupa sehingga menghasilkan suara yang mengandung irama, lagu dan keharmonisan dari alat musik. Musik sudah dikenal sejak 180.000 dan 100.000 tahun yang lalu. Sekarang musik telah tumbuh dan berkembang menjadi berbagai macam dengan berbagai Bahasa. Walaupun musik adalah sejenis fenomena intuisi, untuk mencipta, memperbaiki dan mempersembahkannya adalah suatu bentuk seni. Mendengar musik adalah sejenis hiburan. Musik adalah sebuah fenomena yang sangat unik yang bisa dihasilkan oleh beberapa alat musik.

Keberjalanan musik dalam berkembang bisa dibilang sangat pesat dan sulit untuk dibendung. Setiap tahun sekitar 2500 lagu tercipta yang menandakan bahwa terciptanya kurang lebih 8 buah lagu di setiap harinya. Radio menjadi salah satu media penting dalam penyedia lagu / sarana untuk menerbitkan lagu. Tidak jarang orang – orang yang biasanya mendengarkan radio berusaha untuk mencari tahu judul lagu tersebut. Namun, kerap kali mereka tersendat oleh keterbatasan informasi yang mereka punya. Misalnya hanya mengetahui sepatah lirik atau sebagian nadanya saja. Hal ini yang memicu para developer aplikasi untuk membuat aplikasi yang dapat menganalisa musik.

Analisa musik adalah sesuatu yang terbilang masih baru dan bisa dikatakan mulai memanfaatkan suatu teknik dari ilmu ilmu computer science. Seiring dengan bertambahnya jumlah lagu yang tidak terbendung mengakibatkan semakin dibutuhkannya ‘sesuatu’ untuk membantu manusia menganalisa musik. Nyatanya pun sekarang sudah ada beberapa aplikasi penganalisa musik contohnya adalah Soundhound, Shazam, Midomi dan masih banyak lagi aplikasi lain yang serupa dengan kedua aplikasi tersebut.

Gambar 1. Soundhound, contoh aplikasi pendeteksi lagu – sumber : https://www.theandroidsoul.com/identify-songs-with-soundhound-android-app/

Keberadaan mereka sangat membantu dan memberiknan kemudahan untuk para penikmat musik di seluruh dunia. Sehingga saat mereka sedang tidak sengaja mendengarkan sebuah lagu di café, di radio, di konser, atau di mana pun mereka dapat dengan cepat dan mudah mengetahui lagu apa yang mereka dengar, serta disajikan lirik untuk membantu mereka bernyanyi bersama. Hanya dengan satu kali sentuh dan aplikasi pengenal musik dan anda terkoneksi ke internet anda dapat mencari lagu apapun yang anda inginkan tanpa perlu effort untuk mengingat bagian dari lagu tersebut.

Lagu – lagu yang sudah dipastikan memiliki lirik dan nada yang unik (meski ada beberapa lagu yang memiliki nada yang sama namun liriknya berbeda) dapat dikenali oleh pengenal musik dari berbagai Bahasa, negara, dan pencipta. 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 Pattern Matching*

Algoritma pencarian string atau sering disebut juga algoritma pencocokan string yaitu algoritma untuk melakukan pencarian semua kemunculan string pendek dan dan panjang, untuk string pendek yang disebut pattern dan string yang lebih panjang yang disebut teks.

string pendek = pattern[0..n-1]

string panjang = teks[0..m-1]

Algoritma pencarian string ini dapat juga diklasifikasikan menjadi 3 bagian menurut arah pencariannya ,berikut ini adalah algoritma yang termasuk dalam algoritma ini. Dari arah yang paling alami yaitu dari kiri ke kanan, yang merupakan arah untuk membaca, algoritma yang termasuk kategori ini adalah:

- Algoritma Brute Force. Anda bisa membaca lebih detail tentang algoritma tersebut pada judul artikel ini "Algoritma Brute Force"
- Algoritma dari Morris dan Pratt, yang kemudian dikembangkan oleh Knuth, Morris, dan Pratt

Kategori kedua yaitu dari arah kanan ke kiri, arah yang biasanya menghasilkan hasil terbaik secara praktikal, contohnya adalah:

- Algoritma dari Boyer dan Moore, yang kemudian banyak dikembangkan, menjadi Algoritma turbo Boyer-Moore, Algoritma tuned Boyer-Moore, dan Algoritma Zhu-Takaoka;

Dan kategori terakhir yaitu adalah dari arah yang ditentukan secara spesifik oleh algoritma tersebut, arah ini menghasilkan hasil terbaik secara teoritis, algoritma yang termasuk kategori ini adalah Algoritma Colussi dan Algoritma Crochemore-Perrin".

1) Brute Force

Algoritma brute force menggunakan metode pemeriksaan setiap karakter pada teks mulai dari posisi pertama hingga posisi ke (n – m), jika terdapat kecocokan dengan awal pola yang dicari, maka dilakukan pergeseran pada pola satu karakter ke kanan. Kompleksitas waktu pada algoritma brute force secara umum adalah sebesar O(nm), berikut bentuk implementasi algoritma brute force pada bahasa java:

```
public static int brute(String text,String pattern) {
        int n = text.length(); // n is length of text
        int m = pattern.length(); // m is length of
        pattern
        int j;
        for(int i=0; i <= (n-m); i++) {
                j = 0;
                while ((j < m) && (text.charAt(i+j)==
        pattern.charAt(j)) ) { j++; }
                if (j == m)
                        return i; // match at i
                } return -1; // no match
        } // end of brute()
```

Algoritma brute force seringkali disebut algoritma naif karena algoritma ini mengecek segala kemungkinan secara traversal tanpa memedulikan efisiensi dari algoritma tersebut.

Contoh penggunaan :
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
```

Pada contoh di atas berhenti pada langkah ke 7 karena sudah menemukan kata yang cocok pada langkah ke 7

Teks: 1001010100101111010101010001
Pattern: 001011

```
      1001010100101111010101010001

s=0 001011
s=1  001011
s=2   001011
s=3    001011
s=4     001011
s=5      001011
s=6       001011
s=7        001011
s=8         001011
```

Pada contoh di atas berhenti pada langkah ke 8 karena sudah menemukan kata yang cocok pada langkah ke 8.

2) Knuth-Morris-Pratt

Algortima knuth-Morris-Pratt dikembang secara terpisah oleh Donald E-Knuth pada tahun 1967 James H. Morris bersama Vaughan R.Pratt pada tahun 1966, namun keduanya mempublikasikannya pada tahun 1977.

Jika kita lihat kembali algoritma straightforward (brute force) lebih mendalam, maka dapat kita ketahui bahwa mengingat beberapa perbandingan yang dilakukan sebelumnya kita dapat meningkatkan besar pergeseran yang dilakukan. Hal ini akan menghemat perbandingan, yang selanjutnya akan meningkatkan kecepatan pencarian string.

Dalam algoritma Knuth-Morris-Pratt kita dapat memutuskan keberhasilan dan kegagalan setiap karakter yang dibandingkan. Algoritma KMP membangun sebuah mesin automata yang status-statusnya adalah status dari string yang

kita cari. Dan setiap status memiliki fungsi berhasil dan gagal. Berhasil artinya status mendekati akan bergerak mendekati ke status akhir dan gagal apabila status status menjadi semakin menjauh dengan status terakhir.
Secara sitematis langkah-langkah yang dilakukan algoritma Knuth-Morris-Pratt pada saat mencocokkan sebuah string adalah sebagai berikut :

1. Algoritma Knuth-Morris-Pratt mulai mencocokkan pattern pada awal teks.
2. Dari kiri ke kanan, algoritma ini akan mencocokkkan string karakter per karakter pattern denga teks yang bersesuaian, sampai salah satu kondisi terpenuhi :
    a. Karakter di pattern dan teks yang dibandingkan tidak cocok (mismatch).
    b. Semua di pattern cocok, kemudian algoritma akan memberitahukan penemuan di posisi ini.
3. Algoritma ini menggeser pattern pada table next, lalu mengulangi langkah 2 sampai pattern berada pada ujung teks.

Berikut adalah algoritma Knuth-Morris-Pratt pada fase pra pencarian dalam python :

```
def kmp(strings, find_string):
    n = len(strings)
    m = len(find_string)
    j = 0
    k = 0
    fail = kmp_fail(find_string)
    while j < n:
        if strings[j].lower() ==
find_string[k].lower():
            if k == m - 1:
                return j - m + 1
            j += 1
            k += 1
        elif k > 0:
            k = fail[k-1]
        else:
            j += 1
    return -1


def kmp_fail(find_string):
    n = len(find_string)
    fail = [0] * n
    j = 1
    k = 0
    while j < n:
        if find_string[j].lower() ==
find_string[k].lower():
            fail[j] = k + 1
            j += 1
            k += 1
        elif k > 0:
            k = fail[k - 1]
        else:
            j += 1
    return fail
```

j	0	1	2	3	4	5	6	7	8	9
P [j]	a	b	a	b	a	b	a	b	c	a
k	-	0	1	2	3	4	5	6	7	8
b[k]	-	0	0	1	2	3	4	5	6	0

Gambar 2. Border Function pada KMP sumber : http://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2017-2018/Pencocokan-String-(2018).pdf

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).

3) Boyer Moore

Boyer-Moore String Matching merupakan sebuah cara untuk mencocokan sebuah String dari kanan ke kiri. Sebuah teks dicocokan dengan pattern tertentu untuk menentukan apakah dalam teks yang dicocokan terdapat pattern tersebut.

Perbedaan pencocokan String Boyer-Moore dengan pencocokan String secara Brute Force adalah pada Algoritma Boyer-Moore tidak semua String dicocokan seperti pada cara Brute Force. Ketika Ada text dan pattern yang terjadi Mismatch, maka pattern akan mencocokan text meloncat menurut nilai pada tabel delta atau sebanyak jumlah karakter yang telah dicocokan. Hal ini tergantung nilai maksimum yang terdapat dari keduanya.

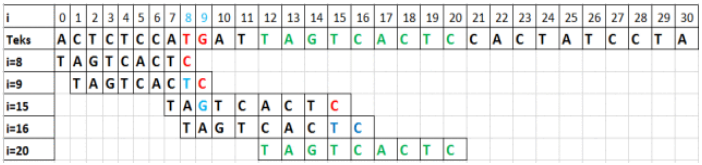
Diketahui
Teks: ACTCTCCATGATTAGTCACTCTCCACTATCCTA
Pattern: TAGTCACTC

1. Mencari Tabel Delta atau Occurrence Heuristic Table

kemunculan pada indeks	A	T	G	C
0				0
1		1		
2				2
3	3			
4				4
5		5		
6			6	
7	7			
8	8			

Tabel Delta				
Karakter Pattern	A	T	G	C
Kemunculan paling awal	3	1	6	0

Tabel delta didapat dari kemunculan pertama sebuah pattern yang dihitung dari kanan. perhitungan dimulai dari angka nol. Berikut adalah Tabel pencocokan String dengan menggunakan Algoritma Boyer-Moore



Misalnya ada sebuah usaha pencocokan yang terjadi pada teks[i..i+n-1] teks[i..i+n-1], dan anggap ketidakcocokan pertama terjadi di antara teks[i+j] teks[i+j] dan pattern[j] pattern[j], dengan 0<j<n 0<j<n. Berarti, teks[i+j+1..i+n-1]=pattern[j+1..n-1] teks[i+j+1..i+n-1]=pattern[j+1..n-1] dan a=teks[i+j] a=teks[i+j] tidak sama dengan b=pattern[j] b=pattern[j]. Jika u u adalah akhiran dari pattern sebelum b b dan v v 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] 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] pattern[j]. Jika tidak ada potongan seperti itu, maka algoritme akan menyejajarkan akhiran v v dari teks[i+j+1..i+n-1] teks[i+j+1..i+n-1] dengan awalan dari pattern yang sama.
2. Penggeseran bad-character yang terdiri dari menyejajarkan teks[i+j] 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] teks[i+n+1].

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

Algoritme Boyer-Moore mulai mencocokkan pattern pada awal teks.

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

Berikut adalah pseudocode algoritme Boyer-Moore pada fase pra-pencarian:

```
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

Algoritme:
  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)
```

```
for (i = 0 to n - 2)
     bmGs[n - 1 - suff[i]] := n - 1 - i;
  endfor
```

Pseudocode saat 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

Algoritme:
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 Boyer – Moore cepat untuk mencocokan pattern pada text dengan jenis alpabhet yang banyak dan akan lambat jika digunakan untuk pencocokan binary. Algoritma ini secara signifikan lebih cepat dalam pencocokan string kalimat biasa dibadingkan dengan algoritma brute force. Koompleksitas waktu pada kasus terburuk dari algoritma Boyer Moore adalah O(nm + A) dengan A merupakan jumlah alphabet yang muncul. Pada kasus normal, kompleksitas algoritma ini O(n + m).

Algoritma Boyer-Moore untuk mencocokan sebuah String jauh lebih Efisien dibandingkan dengan Algoritma Brute Force karena Pada algoritma Brute-Force, string dicocokan satu per satu hingga selesai. Pada Algoritma Boyer-Moore ketika karakter pada teks yang dicocokan pada pattern tidak cocok, maka pattern akan digeser sejauh nilai maksimun dari nilai delta atau perpindahan pattern tersebut.

B. SoundHound

1) Tentang

SoundHound adalah applikasi untuk pencarian musik yang tersedia di Apple App Store , Google Play dan Windows Marketplace for Mobile. Hal ini memungkinkan pengguna untuk mengidentifikasi musik saat dimainkan. Hal ini juga memungkinkan untuk melafalkan atau mengetik nama artis atau lagu. Tidak seperti pesaingnya yaitu Shazam , Shazam hanya dapat mengidentifikasi lagu melalui musik yang sedang di mainkan.

Setelah mengidentifikasikan lagu, aplikasi ini memberikan kita lirik dari lagu tersebut, lalu applikasi ini dapat menyediakan link yang dapat menyambungkan kita ke video lagu tersebut di YouTube, iTunes, nada dering, dan dapat mengakses Pandora Radio , serta memberikan rekomendasi untuk lagu lainnya.

Ada tiga versi dari aplikasi ini yaitu: SoundHound, SoundHound Infinity dan Hound. SoundHound gratis tetapi memiliki iklan banner, sementara SoundHound Infinity (gaya SoundHound ∞), harga £ 4,99 di Inggris atau $ 6,99 di AS, adalah persembahan premium dan memiliki fungsi yang sama tetapi tanpa iklan banner, dan Hound hanya memungkinkan pengguna untuk mencari artis atau lagu dengan berbicara ke dalamnya. Serupa dengan aplikasi SoundHound, Hound kemudian kembali preview lagu, lirik, seni album dan video serta bios artis dan tanggal tur.

Terobosan teknologi Sound2Sound SoundHound yang mencari suara terhadap suara, melewati suara tradisional untuk teknik konversi teks bahkan ketika mencari database teks. Sound2Sound telah menghasilkan banyak terobosan termasuk pengakuan tercepat di dunia musik, menyanyi hanya layak di dunia dan pencarian bersenandung, dan instan-respon sistem pengenalan suara skala besar.

SoundHound telah menghasilkan beberapa aplikasi yang memenangkan penghargaan dalam musik dan pencarian suara di platform termasuk iPhone, iPad, Android, Windows Mobile dan perangkat Symbian. SoundHound juga memiliki berbagai kemitraan dengan perusahaan global, termasuk produsen perangkat terkemuka dan operator.

SoundHound kini berkantor pusat di Santa Clara, California, SoundHound didukung oleh terkemuka investor Silicon Valley, termasuk perusahaan Catalyst Global Partners, TransLink Capital dan Walden Ventura Capital .

2) Sound2Sound

Sound2Sound (S2S) adalah teknologi revolusioner, yang diciptakan oleh SoundHound, yang mampu mengenali berbagai masukan suara termasuk musik dan pidato. Applikasi ini menawarkan kombinasi terobosan kecepatan dan akurasi tercapai melalui pendekatan tradisional untuk suara pengakuan.

S2S melakukan pengakuan dengan mengekstraksi fitur dari sinyal input dan mengkonversi mereka ke representasi kristal kompak dan fleksibel. Ini input Crystal kemudian dicocokkan dengan database Sasaran Kristal yang telah diperoleh dari konten dicari.

Kristal sasaran secara otomatis dihasilkan dari berbagai format data termasuk data audio (seperti rekaman musik dan rekaman suara pengguna) dan data non-audio. Konten berbasis teks biasa, misalnya, bisa mulus dicerna oleh S2S karena teknik sintesis kuat.

Dalam semua kasus, pengakuan dilakukan langsung pada representasi Crystal - cocok "suara terdengar" - menghindari konversi kesalahan-sarat (seperti dari "suara ke teks") hadir dalam sistem tradisional. Selanjutnya, S2S sepenuhnya parallelizable, sehingga kemampuan untuk mencapai kecepatan dan akurasi secara bersamaan sebagai lawan mengorbankan satu untuk yang lain.

3) Cara Kerja

Salah satu contoh gambar cara kerja pembacaan frekuensi, nada yang di analisa oleh aplikasi setelah suatu lagu dikirim ke server shazam maupun soundhound:

Untuk lebih jelasnya, SoundHound mempunyai database lagu yang sangat besar. Lagu-lagu yang dikirim oleh pengguna SoundHound melalui aplikasi SoundHound, data lagunnya diambil terpisah dan dianalisis untuk sifat akustik mereka oleh server SoundHound - khususnya "frekuensi puncak" di berbagai titik dari suatu lagu. Ketika Anda mengcapture 10 detik audio dan mengirimkannya ke server SoundHound, Anda juga telah mengirimkan frekuensi dari lagu tersebut: catatan dari instrumen, harmoni yang khas dari suatu lagu dan sejenisnya. Orang lain mungkin akan bilang, pemanas mobil Anda atau A / C dapat mengeluarkan suara saat anda mengcapture suatu lagu yang dapat mengganggu proses identifikasi lagu via SoundHound, tetapi jika catatan data dan nada suatu lagu masih dapat dikenali oleh SoundHoundatau sound hound (dalam teorinya) masih cocok dengan kombinasi tertentu dari frekuensi suatu lagu yang ditemukan dalam database SoundHound, maka lagu tersebut masih dapat teridentifikasi.

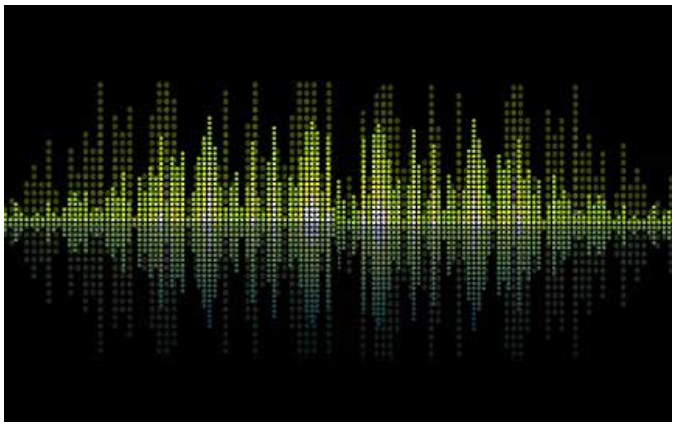
Gambar cara kerja SoundHound secara gamblang :

1. Pemakai aplikasi Soundhound mengcapture sebagian nada dari suatu lagu ke aplikasi Soundhound,
2. Aplikasi Soundhound akan mengextract data dari suatu lagu ke server,
3. Data dari suatu lagu dicocokkan dengan database lagu yang ada, kemudian diextract lagi untuk mengetahui lagu yang kita kirim via aplikasi Soundhound cocok dengan algoritme lagu yang sesuai.
4. Kemudian dalam database jika telah diketahui suatu kecocokkan lagu maka akan mengirim balik ke pengguna aplikasi, informasi berupa judul lagu, artis, dan tag picture lagu tersebut (Jika ada).

III. Analisis & Pembahasan

Musik memiliki berbagai elemen. Ketukan, nada, dan lirik merupakan sebagian dari elemen musik. Namun, tidak jarang juga musik yang tidak memiliki lirik. Musik biasa digambarkan dengan nada – nada (dengan frekuensi masing – masing) yang dimainkan pada ketukan tertentu dan pitch tertentu. Cara menggambarkan musik pun telah menjadi perdebatan yang cukup lama (kurang lebih 20 tahun terakhir). Biasa orang – orang membandingkan elemen - elemen dari lagu – lagu yang ada contohnya tinggi rendahnya nada, durasi, volume suara, warna nada, notasi, dan fitur fitur lain. Namun, saya dalam makalah ini akan membahas lebih jauh tentang nada. Terutama dalam pencocokan nada yang diinput dengan database lagu.

Nada dapat ditangkap dan direpresentasikan dalam bilangan biner. Bilangan – bilangan tersebut lah yang akan digunakan sebagai media pencocokan pattern. Sebelum melakukan pencocokan pola antara penggalan lagu yang didapat dengan cara merekam lagu dari sekitar, pola yang didapat tersebut masih berupa gelombang suara. Gelombang tersebut tidak hanya merepresentasikan lirik lagu tetapi juga nada, tempo, dan atribut lagu lainnya.



Gambar 3. Gelombang suara. Sumber : https://cepatrambatbunyi.blogspot.com/

Pattern matching akan dilakukan dengan langkah – langkah sebagai berikut :

1. Pengguna akan merekam potongan lagu melalui aplikasi SoundHound
2. Aplikasi pun membaca potongan lagu tersebut dan mengubahnya menjadi string panjang yang terdiri atas bilangan biner (01) yang merepresentasikan potongan nada dari lagu tersebut.
3. Dilakukan proses pencocokan string tersebut dengan database yang dimiliki oleh aplikasi.

Setiap aplikasi tentu memiliki algoritma tersendiri untuk melakukan pencocokan pattern tersebut. Namun, umumnya algoritma yang digunakan merupakan hasil optimalisasi algoritma Knuth-Morris-Pratt ataupun Boyer Moore. Mereka melakukan pengembangan sehingga pencarian dapat dilakukan lebih efisien dan optimal.
Contoh:
Database lagu :
01010100001110101001001
10010100111010110101001
01101010010**011011100100**
10010000100111101011010
…
Pattern ;
11011100100

Ini merupakan salah satu gambaran dari proses pencocokan pattern dari aplikasi SoundHound. Mendeskripsikan musik dengan berbagai alat (piano, gitar, bass, dll) pasti jauh lebih rumit dari ini. Meski realitanya mungkin jauh lebih rumit dari ini, namun secara garis besar saya mencoba menggambarkan penerapan pattern matching dalam proses pencarian lagu dari aplikasi SoundHound.

Umumnya akan dilakukan proses pengecekan kualitas rekaman di awal (sebelum melakukan pencocokan string). Jika kualitas suara tidak sesuai dengan standar akan diminta pengambilan ulang. Kualitas rekaman menjadi salah satu elemen penting dalam penentuan jawaban. Semakin baik kualitas suara akan menghasilkan pattern yang semakin identic dengan di database (suara – suara lain dapat menyebabkan naik turunnya frekuensi nada / lainnya).

Setelah pencocokan, umumnya akan disimpan pula kecocokan terbesar. Kecocokan akan diukur melalui persentase. Lagu yang memiliki 'sub-string' dengan kecocokan terbesar dengan pattern yang diinginkan akan ditampilakan sebagai jawaban. Jawaban tidak selalu benar, dikarenakan error – error yang disebabkan (kualitas rekaman, kelengkapan database, baiknya algoritma).

IV. Kesimpulan dan Saran

Algoritma pattern matching banyak digunakan untuk membantu menyelesaikan berbagai persoalan. Salah satu penggunaanya adalah dalam aplikasi SoundHound. Algoritma tersebut digunakan untuk membantu dalam pencocokan rekaman dengan lagu yang sesuai sehingga kita dapat menemukan informasi yang diinginkan.

V. Ucapan Terima kasih

Pertama – tama dan yang terutama saya mengucapkan terima kasih kepada Tuhan Yang Maha Esa yang telah melimpahkan berkat dan penyertaan-Nya sehingga makalah Strategi Algoritma ini dapat terselesaikan dengan baik dan tepat waktu. Saya juga mengucapkan terima kasih kepada kedua orang tua saya yang selalu memberi dukungan, doa, dan semangat kepada saya sehingga saya dapat menempuh pendidikan sampai saat ini. Tak lupa juga saya mengucapkan terima kasih kepada Bapak Dr. Ir. Rinaldi Munir, MT serta Ibu Dr. Nur Ulfa Maulidevi yang berperan sebagai dosen matakuliah IF2211 Strategi Algoritma sehingga dengan ilmu pengetahuan seputar Strategi Algoritma saya dapat membuat dan menyelesaikan makalah ini.

References

[1] Davison Andrew, Pattern Matching, 2006 (updated by Rinaldi Munir)

[2] Lecture 3 : Boyer Moore Algorithm. 2005. University of Kuopio Davison Andrew, Pattern Matching, 2006 (updated by Rinaldi Munir)

[3] Mandumula, Kranthi Kumar. Knuth Morris Pratt Algorithm. 2011.

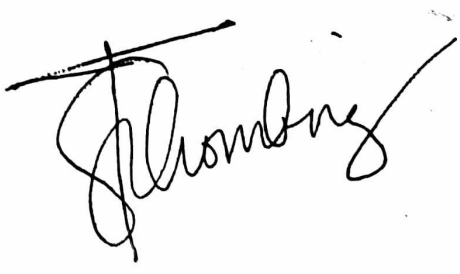
[4] Cliffor, Raphel and Iliopoulos, Costas. Approximate String Matching for Music Analysis.

[5] Kilpelainen, Pekka.

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, 14 May 2018



Timothy Thamrin Andrew H. Sihombing - 13516090