APLIKASI ALGORITMA *KNUTH-MORRIS-PRATT* DALAM *CONTENT-BASED MUSIC INFORMATION RETRIEVAL*

Mohammad Rizky Adrian (13507108)

Jurusan Teknik Informatika, Institut Teknologi Bandung, Indonesia
Bandung, Jawa Barat, Indonesia
e-mail: if17108@students.if.itb.ac.id

ABSTRAK

Umumnya pencarian database musik hanya sekedar melakukan pencarian data berdasarkan pencocokan string dengan metadata informasi lagunya, seperti judul lagu, pengarang, nama penyanyi, nama produser, tahun keluaran dan lain sebagainya. Namun jika penyedia database musik ingin menyediakan metode pencarian berdasarkan isi/konten dari lagunya masih sulit diterapkan. Walaupun sudah ada beberapa instansi yang menyediakan *feature* ini. Konsep pencarian dalam pengambilan informasi musik inilah yang akan dibahas oleh penulis. Penulis tidak membahas penanganan penyimpanan data musik.

Kata kunci: Kontribusi, *Knuth-Morris-Pratt*, *Music Information Retrieval(MIR)*, Pencocokan, *String Matching*, String, Substring, Pola, Metode, Pengambilan Informasi, *database*.

1. PENDAHULUAN

Sebelum terlibat lebih jauh dalam bagaimana algoritma Knuth-Morris-Pratt diterapkan dalam Content-Based Music Information Retrieval(MIR), mari kita tinjau terlebih dahulu komponen-komponen atau metode yang akan digunakan di dalamnya.

1.1 String

String dalam ilmu komputer dapat diartikan dengan sekuens dari karakter. Walaupun sering juga dianggap sebagai data abstrak yang menyimpan sekuens nilai data, atau biasanya berupa *bytes* yang mana merupakan elemen yang digunakan sebagai pembentuk karakter sesuai dengan *encoding* karakter yang disepakati seperti ASCII, ataupun EBCDIC.

Hubungan string dengan tulisan ini adalah bahwa karakteristik dari musik yang akan disimpan dalam *database* dapat dianggap serupa dengan string. Hal ini akan memudahkan desainer dalam membangun sistem pencocokan audio dari sampel audio yang akan dikonversi terlebih dahulu menjadi serupa dengan string ataupun deretan bytes. Konversi inilah yang nantinya akan dibandingkan langsung dengan informasi karakteristik yang disimpan dalam *database*.

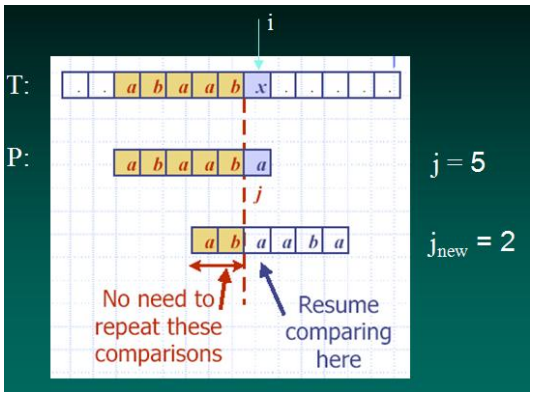
1.2 Algoritma *Knuth-Morris-Pratt* [1][2] serta perbandingan dengan algoritma yang lain[4]

Algoritma ini dikembangkan oleh D.E. Knuth bersama dengan J.H Morris dan VR Pratt. Algoritma ini termasuk salah satu algoritma primitive yang digunakan untuk mencari kesamaan pola pada string atau deretan suatu karakter.

Pada persoalan pencocokan string umumnya diberikan dua buah hal utama:

- Teks, string yang relative panjang yang akan menjadi bahan yang akan dicari kecocokannya (dimisalkan panjang dari teks adalah n)
- *Pattern*, string dengan panjang relatif lebih pendek dari teks (misalkan panjang pattern adalah m, maka m<n). *Pattern* akan digunakan sebagai string yang dicocokkan ke dalam teks

Umumnya persoalan berupa, "carilah apakah ditemukan pattern yang diberikan pada teks tertentu". Jawabannya bisa berupa ada atau tidak, berapa kali ditemukan, ataupun di posisi ke berapa.



Gambar 1. Contoh penghematan pergeseran pointer *pattern* pada KMP

Dasar utama dari algoritma knuth-morris-pratt adalah untuk mereduksi pergeseran pointer pencocokan string

lebih jauh daripada harus menggesernya satu persatu layaknya yang dilakukan pada metode pencocokan string dengan algoritma brute force.

Uniknya, berbeda dengan cara konvensional(*brute force*) yang melakukan pergeseran sebanyak satu, informasi dari pattern akan menentukan jumlah pergeseran yang relatif lebih jauh. (Perhatikan gambar 1)

Ada beberapa definisi yang perlu dipahami pada algoritma ini:

Misalkan A adalah alfabet dan x = x1x2…xk adalah string yang panjangnya k yang dibentuk dari karakter-karakter di dalam alfabet A.

- **Awalan (prefix)** dari x adalah upa-string (substring) u dengan
  $u = x_1x_2\ldots x_{j-1}$, $j \in \{1, 2, \ldots, k\}$
  dengan kata lain, x diawali dengan u.

- **Akhiran (suffix)** dari x adalah upa-string(substring) u dengan
  $u = x_{j-b}\, x_{j-b+1} \ldots x_k$, $j \in \{1, 2, \ldots, k\}$
  dengan kata lain, x di akhiri dengan v.

- **Pinggiran (border)** dari x adalah upa-string r sedemikian sehingga
  $r = x_1x_2\ldots x_{j-1}$
  dan
  $u = x_{j-b}\, x_{j-b+1} \ldots x_j$,
  $j \in \{1, 2, \ldots, k\}$
  dengan kata lain, pinggiran dari x adalah upastring yang keduanya awalan dan juga akhiran sebenarnya dari x.

Jika misalkan ditentukan sebuah pattern adalah
x = abacab.
Maka awalan dari x adalah
□, a, ab, aba, abac, abaca
Akhiran dari x adalah
□, b, ab, cab, acab, bacab
Pinggiran dari x adalah
□, ab
Pinggiran □ mempunyai panjang 0, pinggiran ab mempunyai panjang 2.

Jika pada pencocokan *pattern* lalu pada posisi ke y ditemukan ketidakcocokan pada teks, apa ide yang dilakukan oleh algoritma ini?

Mengeliminasi sejumlah besar pergeseran yang tidak perlu yakni sepanjang awalan terbesar yang juga merupakan akhiran dari *pattern* sesuai dengan posisi terakhir pencocokkan pada teks.

Yang perlu diketahui oleh pemakai kode adalah bahwa algoritma KMP melakukan proses awal(*preprocessing)* terhadap *pattern* P dengan menghitung fungsi pinggiran yang mengindikasikan pergeseran s terbesar yang mungkin dengan menggunakan perbandingan yang dibentuk sebelum pencarian string. Dalam literatur lain fungsi ini sering disebut dengan *overlap function, failure function*, fungsi awalan dan lainnya. Fungsi preprosesing pinggiran tersebut dapat dipahami sebagai berikut.

```
procedure FungsiHitungPinggiran
(
    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 : interger

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
```

Fungsi tersebut akan menghasilkan output berupa array integer yang merupakan angka-angka pinggiran untuk setiap posisi iterasi pada *pattern*.

Barulah kemudian dapat diproses pencocokkan antara *pattern* dan teks yang diberikan. Algoritma Knuth-Morris-Pratt selengkapnya adalah sebagai berikut:

```
procedure Algoritma KMP
(
    input m, n: integer,
    P: array [1..m] of char,
    T: array [1..n] of char
    output idx : integer
)

{ Menghitung nilai b[1..m] untuk
pattern P[1..m] }

Deklarasi
i,j  : interger
ketemu : Boolean
b : array [1..m] of integer
```

```
procedure FungsiHitungPinggiran
(
    input m: integer,
    P: array [1..m] of char,
    output b : array[1..m] of integer
)

Algoritma:
FungsiHitungPinggiran(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
else
  idx ← -1
endif
```

Sama seperti algoritma pencocokkan string lainnya mengalami dua jenis loop, yakni *outer loop(* diacu oleh pointer pada teks untuk bergeser) dan *inner loop*(diacu oleh pointer pada *pattern* untuk mencocokkan string dimulai dari pointer pada teks).

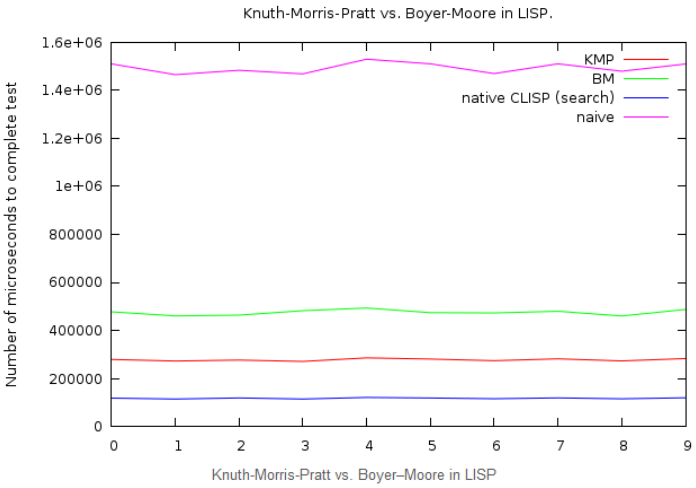
Coba kita analisa bagaimana kompleksitas dari algoritma ini dibandingkan beberapa algoritma pencocokkan string lainnya.

Outer loop akan dieksekusi m kali iterasi. Sementara itu pada setiap iterasi *inner loop* nilai dari Fungsi Pinggiran P[j+1] dikurangi satu dan nilai ini hanya akan dinaikkan satu ketika pointer *outer loop* bergeser satu. Karena jumlah dilakukannya penurunan paling banyak sejumlah dilakukannya penambahan, *inner loop* juga memiliki paling banyak m iterasi. Sehingga kompleksitas untuk algoritma ini adalah O(m).

Ada yang perlu diperhatikan bahwa menjalankan fungsi pinggiran membutuhkan waktu juga walaupun relatif kecil. Fungsi pinggiran tersebut memiliki kompleksitas sebesar O(n).

Total keseluruhan kompleksitas dari algoritma Knuth-Morris-Pratt adalah O(m+n).

Pada algoritma *brute force* juga dilakukan iterasi *outer loop* yakni pada teks yang diberikan sejumlah m kali. Besarnya m adalah panjang teks. Juga dilakukan iterasi *inner loop*, yang merupakan iterasi pada *pattern*. Karena semua masing-masing iterasi dilakukan pergeseran sebanyak satu saja, maka kompleksitas algoritma pencocokkan string dengan *brute force* adalah O(m.n).



Gambar 2. Grafik perbandingan kecepatan beberapa algoritma pencocokkan string[5]

Grafik pada gambar 2 di atas menunjukkan beberapa perbandingan algoritma pencocokkan string yang sering digunakan oleh programmer. Algoritma Booyer moore yang memiliki kompleksitas O(n) tidak lebih baik untuk kasus worst case dibandingkan dengan Knuth-Morris-Pratt.

Keuntungan dari algoritma Knuth-Morris-Pratt selain cepat juga sangat baik digunakan pada file berukuran besar karena pencarian kecocokkan tidak perlu kembali ke belakang pada input teks. Namun memiliki kekurangan yakni efektifitas dari algoritma ini akan berkurang seiring dengan bertambahnya jumlah alphabet(jenis karakter) dari teks.

1.3 *Content-Based Music Information Retrieval (MIR)*[3]

Musik dari masa ke masa sangatlah besar pengaruhnya di kehidupan kita. Alunan musik dari masa ke masa tidak ada habisnya walaupun lagu-lagu yang tercipta sudah tidak terhitung lagi.

Untuk itu penelitian mendalam di bidang musikal dipelajari. *Music information retrieval(*MIR) adalah salah

satunya. Strategi untuk menyediakan akses kepada koleksi musik baik itu musik lama atau musik terdahulu disebut MIR. Mengapa perlu diteliti? Hal ini perlu untuk menjaga harapan performansi bagaimana mengakses data-data audio musik yang jumlahnya tak terhitung.

Selama ini akses terhadap file-file musik tersebut cenderung lebih kepada pencarian berdasarkan metadata tekstualnya saja. Sebut saja nama judul lagu, nama penyanyi, nama penyair, nama komposer, tahun rilis, dan masih banyak lagi atribut-atribut yang menjadi identitas dari lagu tersebut.

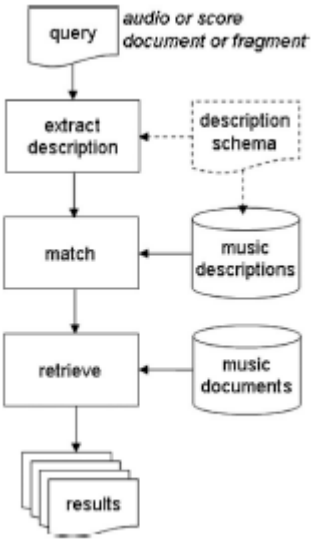
Namun terlepas dari semua atribut metadata yang menjadi identitas suatu file musik, konsumen musik membutuhkan akses lain pada file musik itu, yakni konten dari file musik itu sendiri. Anggap saja kita punya suatu cuplikan lagu yang kita dengar dari radio. Bagaimana kita dapat mencari lagu tersebut di database musik hanya berdasarkan cuplikannya saja.

2. KONSEP PENGAMBILAN FILE MUSIK

Gambar 3 di bawah ini dapat merepresentasikan secara kasar bagaimana flowchart konsep pengambilan file musik berdasarkan konten/isi lagunya.

Setelah file-file musik disimpan dalam *database* berdasarkan criteria yang ditentukan oleh desainer sistem. Barulah dapat dilakukan pencarian file musik berdasarkan kontennya.

Pencarian file musik diawali dengan pengambilan query sampel audio. Sampel audio akan diproses dianalisa dan dikonversi menjadi deretan string. Setelah itu algoritma Knuth-Morris-Pratt baru bekerja pada tahap ini. Adanya kecocokan sampel audio dengan sumber musik itu yang akan menjadi hasil dari pencarian tersebut.



Gambar 3. Flowchart bagaimana pengambilan file music dari *database* berdasarkan kontennya[3]

Dalam persoalan *content-based Music Information Retrieval*, teks pada algoritma knuth-morris-pratt akan dapat dianggap sebuah file-file sumber musik yang berada di database, entah itu karakteristiknya ataupun langsung secara bit per bit dari file tersebut.

Sementara, yang dapat dianggap sebagai *pattern* pada algoritma tersebut adalah, sampel musik yang telah diolah dan dikonversi menjadi deretan karakter atau string yang telah disepakati oleh desainer sistem tersebut.

2.1 Penyimpanan File Musik pada *Music Database*

Beberapa sistem penyimpanan file musik yang telah ada di dunia maya saat ini telah menyediakan *content-based music information retrieval.* Setiap sistem mendesainnya berbeda-beda.

Ada sistem yang menyimpan file musik secara utuh tanpa ada pengelolaan atas kontennya. Walau mudah menyimpannya sistem tersebut akan mengalami kesulitan pengaksesan karena dalam pencarian akan ada jutaan atau lebih file sehingga akan memakan waktu yang cukup lama karena harus mencari dengan kerja keras bit per bit byte per byte.

Kebanyakan sistem mengekstrak file music menjadi beberapa ciri atau karakteristik. Lihat saja beberapa karakteristik yang dapat diekstrak dari suatu file musik seperti pada tabel 1.

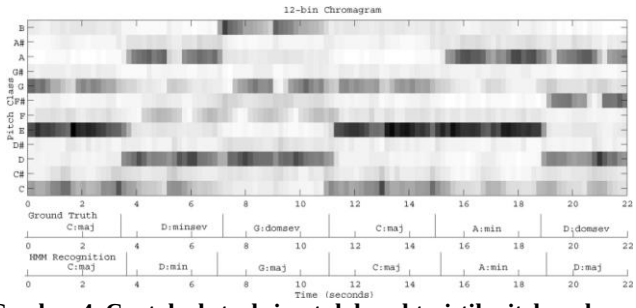
High-level Description	Data Source	Task Description
Timbre	Audio	Instrument Recognition Percussive, Pitched, Ensemble Recognition
Melody / Bass	Audio / Symbolic	Melody-line extraction Bass-line extraction
Rhythm	Audio	Onset detection Meter identification Meter alignment (bars) Beat (tactus) tracking Tempo tracking Average tempo
Pitch	Audio	Single fundamental freq. Multiple fundamental freq.
Harmony	Audio / Symbolic	Chord label extraction Bass-line extraction
Key	Audio / Symbolic	Modulation tracking Pitch spelling
Structure	Audio / Symbolic	Verse / chorus extraction Repeat extraction
Lyrics	Audio	Singing detection, lyrics-identification, word recognition
Non-Western music	Audio	Micro-tonal tuning systems Non-Western canon of concepts

Tabel 1. Beberapa penyimpanan karakteristik atau ciri dari setiap file audio atau musik[3]

Sebut saja Sistem Caterpillar[6] yang membagi file musik menjadi deskripsi(karakteristik) untuk diekstrak:

- Unit(durasi, tipe)
- Sumber(*playing style)*
- Nada
- Sinyal
- Perseptual(kekerasan suara, ketajaman, lebar timbre
- Spektral; dan
- Harmonik

Ciri-ciri ini yang akan disimpan menjadi bagian-bagian string atau deretan karakter yang mewakili setiap ciri tersebut.



Gambar 4. Contoh ekstraksi untuk karakteristik pitch nada yang nantinya dapat disimpan sebagai deret string[3]

2.2 Analisa dan Konversi Pengambilan Sampel Audio

Sampel musik yang akan dicari file nya pada sumber akan dianalisa sesuai dengan karakteristik-karakteristik yang ditentukan oleh sistem pencarian tersebut. Hasil ekstraksi tersebut yang nantinya akan menjadi *pattern-pattern* untuk dicocokkan dengan file sumber.

2.3 Pencocokan String dari Sampel Audio dengan File Musik Sumber

Barulah pada tahap ini algoritma pencocokkan string Knuth-Morris-Pratt diterapkan.

Sebenarnya bukanlah suatu keharusan bahwa sistem harus menggunakan algoritma apa, algoritma yang diterapkan dipilih berdasarkan yang terbaik menurut desainer sistem.

Namun disini penulis menganalisa berdasarkan poin 1.2 bahwa algoritma KMP relatif sangat cepat dalam menemukan pola dalam teks. Dalam kasus ini teks yang dimakud adalah karakteristik-karakteristik file musik.

Penulis juga telah mencoba membuat aplikasi untuk membandingkan antara algoritma *brute force*, Knuth-Morris-Pratt, dan Booyer-Moore dimana ditemukan bahwa baik KMP maupun Booyer Moore relatif lebih cepat dibandingkan dengan *brute force.*

Pencocokkan string diterapkan berdasarkan algoritma KMP yang telah dijelaskan pada poin 1.2 pada setiap karakteristik file-file musik di *database.*

2.4 Hasil Pencarian

Setelah pencarian selesai dilakukan, barulah hasil pencarian ditampilkan kepada peminta query.

Desainer sistem dapat menentukan apakah query sampel dengan file musik harus eksak sama untuk setiap karakteristiknya atau diberi toleransi.

Dari file-file yang ditemukan itulah baru peminta query disajikan tekstual metadata yang menyimpan informasi berupa judul lagu, nama penyanyi, dan lainnya.

3. KESIMPULAN

Jika diperhatikan secara seksama peran dari algoritma pencocokkan string yang dipakai dalam *content-based music information retrieval* sangatlah kecil.

Faktor paling menonjol yang menentukan efektivitas dari akses ke file musik lebih cenderung kepada desain database, penentuan karakteristik file musik, serta manajemen file musik yang baik.

Walaupun peran dari algoritma pencocokkan string ataupun pola tersebut sangat kecil, namun sedikit saja perbedaan kompleksitas antara satu algoritma dengan algoritma lainnyayang dipakai akan berdampak besar pada persoalan *database* musik.

Database musik relatif besar. Sekecil apapun perbedaan kecepatan untuk melakukan pencocokkan pola string itu akan sangat membantu. Terbukti ada saja sistem pencarian musik ada yang menggunakan algoritma Knuth-Morris-Pratt sebagai ujung tombak pencocokkan pola string[3].

REFERENSI

[1] Munir, Rinaldi, "Diktat Kuliah IF3051 Strategi Algoritma", Program Studi Teknik Informatika Sekolah Teknik Elektro dan Informatika Institut Teknologi Bandung, 2009.

[2] http://www.ics.uci.edu/~eppstein/161/960227.html

[3] C. Michael A., V. Remco, G. Masataka, L. Marc, R. Christophe, S. Malcolm, "Content-Based Music Information Retrieval: Current Directions and Future Challenges", Vol. 96, No.4, April 2008, 696 hal.

[4] http://www-igm.univ-mlv.fr/~lecroq/string/node14.html

[5] http://www.ioremap.net/node/78

[6] Diemo Schwarz, "The Caterpillar System for data-driven concatenative sound synthesis", London, UK, 2003.