Analisis Algoritma Pencocokan String untuk Pengenalan Lagu

Ferdinandus Richard / 13515066
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika, Institut Teknologi Bandung
Bandung, Indonesia
13515066@std.stei.itb.ac.id

***Abstract*—Lagu-lagu yang sudah ada di dunia dapat terdiri dari berbagai jenis genre, seperti lagu pop, lagu rock, lagu EDM (electronic dance music), dan bermacam-macam genre lagu lainnya. Semua lagu tersebut memiliki karakteristiknya masing-masing. Ada yang sedih, ada yang bersemangat dan lain sebagainya. Seringkali kita mendengar lagu dan tidak mengetahui apa identifikasi lagu tersebut. Algoritma pencocokan string dapat bekerja sebagai alat yang dapat membantu kita mengetahui judul, penyanyi, dan atribut lainnya mengenai lagu yang sedang kita dengar.**

Keywords — Lagu, String Matching, Pattern, Algoritma, Kompleksitas

I. PENDAHULUAN

Lagu merupakan salah satu suara yang sudah tidak asing di telinga kita. Lagu yang jelas terdiri atas kumpulan nada-nada yang berbeda ataupun sama beserta liriknya. Seringkali lagu dimanfaatkan oleh manusia sebagai "obat" yang berfungsi sebagai penyemangat atau bahkan sebagai alat untuk mengekspresikan diri. Tidak hanya itu, bahkan kekuatan lagu bisa dimanfaatkan untuk memengaruhi mental seseorang hingga menjadi gila, sebagai penenang, atau bahkan bisa menjadi alat bagi seseorang untuk memberikan motivasi untuk melakukan bunuh diri.[2]

Lagu yang sudah beredar di dunia pasti memiliki karakteristrik yang berbeda-beda. Karakteristrik yang membedakan antara masing-masing lagu antara lain seperti genre, judul lagu, penyanyi, album, tempo, dan sebagainya. Sebuah lagu dapat diklasifikasikan menjadi beberapa jenis genre seperti pop, rock, jazz, instrumental, dan lain sebagainya. Sebuah lagu juga dapat dikenali berdasar lagunya, seperti lagu "Berita Kepada Kawan" yang ditulis dan dinyanyikan oleh Ebiet G. Ade yang merupakan salah satu hits pada tahun 1970-1980an ataupun lagu yang berjudul "*Thinking Out Loud*" yang merupakan karya dari Ed Sheeran. Setiap kali kita mendengar judul lagu yang sudah disebutkan tersebut, pasti kita akan langsung tahu identitas dari lagu tersebut. Ini berarti setiap lagu memiliki sesuatu yang unik yang dapat membedakan lagu tersebut dengan lagu-lagu yang lainnya. Selain judulnya, kita juga dapat membedakan lagu berdasarkan iramanya. Semua lagi memiliki irama yang unik yang membedakan antara satu lagu dengan lagu yang lain sehingga irama dan lirik dalam sebuah lagu dapat menjadi identitas yang unik untuk sebuah lagu.

Sesekali saat kita mendengar lagu terkadang kita merasa pernah mendengar lagu tersebut, tetapi tidak mengetahui judul dan siapa penyanyinya. Kadang kita juga pernah merasa familiar dengan lagu tersebut karena kita sempat menyukai lagu tersebut, tetapi kita lupa identitas lagu tersebut karena lagu tersebut adalah lagu yang sudah cukup tua. Oleh karena itu, diciptakanlah sebuah aplikasi yang dapat mengenal identitas sebuah lagu dengan mengambil sampel atau cuplikan lagu yang sedang kita dengar. Aplikasi-aplikasi yang dapat digunakan dalam melakukan pencarian identitas sebuah lagu antara lain Shazam dan Musixmatch. Selain itu, sebuah artificial intelligence juga dapat melakukan pencarian informasi sebuah lagu, seperti Cortana yang dibuat oleh Microsoft untuk PC yang memiliki sistem operasi Windows 10.

Gambar 1. Aplikasi Shazam

Sumber: https://www.technobuffalo.com/2014/12/10/shazam-update-android-ios-spotify-rdio/ (diakses 18 Mei 2017)

Konsep dari pencocokan sebuah lagu dalam aplikasi pencocokan lagu adalah dengan menggunakan algoritma pencocokan string. Seluruh algoritma pencocokan string dapat digunakan dalam rangka mencari pola dalam "text" lagu yang diberikan. Contoh algoritma yang dapat digunakan untuk melakukan pencocokan lagu antara lain algoritma brute force, algoritma Knuth-Morris-Pratt (KMP), algoritma Boyer-Moore, dan algoritma-algoritma lainnya seperti algoritma Rabin-Karp dan algoritma Apostolico-Giancarlo. Pada makalah ini, akan dibahas perbedaan antara algoritma Knuth-Morris-Pratt dan algoritma Boyer-Moore dalam pencocokan pattern dalam sebuah "teks" lagu.

II. DASAR TEORI

Persoalan pencarian string (*string matching*) sudah cukup lama dikenal dalam bidang informatika. Persoalan string matching dapat dirumuskan sebagai berikut:

1. Diberikan sebuah teks dengan panjang *n* karakter.
2. Diberikan sebuah *pattern* dengan panjang *m* karakter, dimana *m* pasti lebih kecil dari *n* (*m* < *n*).
3. Permasalahannya adalah ingin dicari string *pattern* dalam string teks.

Dalam permasalahan ini, dikenal beberapa algoritma yang dapat membantu kita untuk melakukan pencarian pattern dalam sebuah teks. Algoritma yang sudah dikenal dalam pencocokan string antara lain algoritma brute force, algoritma Knuth-Morris-Pratt, dan algoritma Boyer-Moore.

A. Algoritma Brute Force

Algoritma *brute force* adalah salah satu algoritma paling naif yang dapat digunakan dalam menyelesaikan permsalahan pencarian string dalam sebuah teks. Algoritma brute-force merupakan algoritma yang *straight-forward*, jadi cara yang paling terpikirkanlah yang akan dilakukan dalam pencocokan.

Andaikan diberikan teks (T) dan sebuah *pattern* (P). Cara kerja dari algoritma brute force adalah sebagai berikut:

1. Pertama, *pattern* P dicocokkan dengan awal teks T.
2. Pencocokan dilakukan untuk setiap karakter P dengan teks T dari kiri ke kanan sampai seluruh karakter dalam P yang dicocokkan dengan T sudah sama atau ditemukan ketidakcocokan antara karakter dalam P dan karakter dalam T.
3. Jika ketidakcocokan antara P dengan T ditemukan dan karakter dalam T belum habis, P digeser sebanyak 1 karakter, kemudian langkah 2 dilakukan kembali (pencocokan string kembali dilakukan).

Algoritma ini bisa dikatakan sebagai algoritma yang sangat naif karena algoritma ini akan melakukan pencocokan setiap karakter dalam pattern dengan substring dari teks hingga ditemukan sebuah kecocokan antara *pattern* dan substring dari teks. Algoritma ini akan memiliki waktu komputasi yang cukup lama karena harus melakukan pencocokan satu persatu untuk setiap karakternya.

Kasus terbaik yang dapat diterima oleh algoritma brute force memiliki kompleksitas sebesar O(*n*), yaitu bergantung pada panjang karakter dari teks yang diberikan. Kasus terbaik pada algoritma *brute force* terjadi apabila karakter pertama pada *pattern* yang diberikan selalu tidak sama dengan karakter yang ada pada teks hingga pada *m* karakter terakhir dalam teks yang diberikan. Salah satu contoh kasus terbaik dalam *brute force* adalah ketika:

```
T: "halo nama saya andi wiranto"
P: "wiranto"
```

Huruf `w` di dalam teks pasti tidak ditemukan hingga substring "`wiranto`" pada teks yang bersangkutan. Jumlah perbandingan yang terjadi disini adalah sebanyak *n* kali. Oleh karena itu, ini adalah kasus terbaik yang mungkin terjadi dalam penggunaan algoritma *brute force*.

Kasus terburuk dalam algoritma *brute force* memiliki kompleksitas sebesar O(*mn*) yang merupakan perkalian dari panjang teks dan *pattern*. Kasus terburuk dalam penggunaan algoritma *brute force* terjadi apabila setiap karakter dalam *pattern* selalu cocok dengan teks yang diberikan. Akan tetapi, kegagalan selalu terjadi pada akhir karakter dalam *pattern*. Salah satu contoh kasus terburuk algoritma *brute force* terjadi ketika:

```
T: "oooooooooooooooooooooooooh"
P: "ooooh"
```

Pada kasus ini, *pattern* akan selalu menemukan kecocokan dalam pencocokan huruf `o` dengan teks yang diberikan (sebanyak 5 kali) lalu selalu gagal dalam pencocokan huruf `h`. Ini berarti pencocokan akan selalu terjadi sebanyak 6 kali (panjang *pattern*) dikalikan dengan banyaknya *pattern* maju untuk menemukan kecocokan dengan substring dalam teks, yaitu sebanyak *n*-*m*+1 kali. Oleh karena itu banyak pencocokan yang terjadi adalah sebanyak *m*×(*n*-*m*+1) kali. Oleh karena itu, akan didapat kompleksitas dari kasus terburuk dalam algoritma *brute force* dalam pencocokan string, yaitu sebesar O(*mn*).

B. Algoritma Knuth-Morris-Pratt (KMP)

Algoritma Knuth-Morris-Pratt merupakan algoritma pencocokan string yang lebih pintar daripada algoritma *brute force*. Algoritma ini diciptakan oleh Donald Knuth, James H. Morris, dan Vaughan Pratt pada tahun 1977. Algoritma ini memiliki kemiripan dengan algoritma *brute force*. Akan tetapi, algoritma ini tidak pasti menggeser *pattern* sebanyak 1 karakter. Algoritma KMP menggeser *pattern* berdasarkan nilai yang sudah dihitung sebelumnya yang dikhususkan untuk melakukan pergeseran. Nilai yang dihitung ini disebut sebagai fungsi pinggiran. Fungsi pinggiran akan menyatakan banyaknya karakter pada *prefix* [0..*m*] yang sama dengan *suffix* [1..*m*] pada *pattern* yang diberikan.

Penghitungan fungsi pinggiran pada algoritma KMP dapat dinyatakan sebagai sebuah larik dengan panjang *m*. Contoh hasil penghitungan fungsi pinggiran adalah sebagai berikut.

P[i]	a	b	a	b	c	a
b[i]	0	0	1	2	0	1

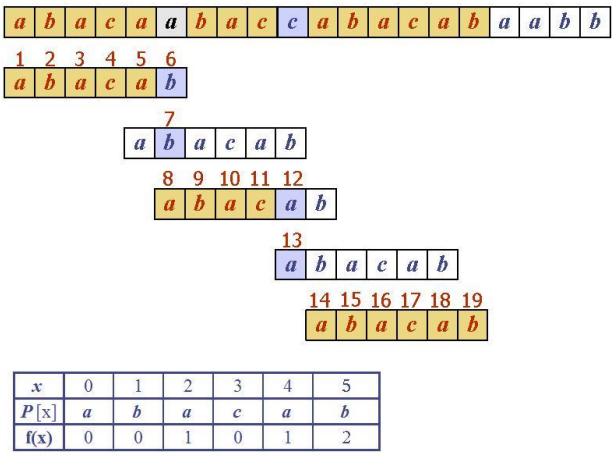
Pada contoh *pattern* di atas, karakter pertama `a` diberikan nilai pinggiran 0 karena tidak ada *suffix* P[1..*m*] yang merupakan `a`. String `ab` juga tidak memiliki *prefix* P[0..1] yang sama dengan *suffix* P[1..1], sehingga diberikan nilai 0. String `aba` memiliki *prefix* dan *suffix* yang sama, yaitu karakter `a`. Oleh karena itu, b[2] diberikan nilai 1. Begitupula dengan `abab` yang memiliki *prefix* yang sama dengan *suffix*-nya yaitu string `ab`, sehingga b[3] diberikan nilai 2. Ketika stringnya `ababc`, tidak ada *prefix* yang sama dengan *suffix*nya,

sehingga b[4] diberikan nilai 0. Untuk b[5] diberikan nilai 1, karena ada *prefix* yang sama dengan *suffix*nya, yaitu karakter `a`.

Andaikan diberikan teks (T) dan sebuah *pattern* (P). Cara kerja dari algoritma KMP adalah sebagai berikut:

1. Pertama-tama penghitungan untuk fungsi pinggiran dilakukan.
2. *Pattern* P dicocokkan dengan awal teks T hingga seluruh karakter dalam P sama dengan substring dari teks T atau ditemukan ketidakcocokan antara karakter pada P dengan karakter pada T.
3. Jika ketidakcocokan ditemukan pada karakter pertama, pergeseran dilakukan sebanyak 1 karakter, sedangkan jika ketidakcocokan ditemukan pada karakter yang bukan karakter pertama, dilihat karakter yang berhasil sebelumnya dan dilihat fungsi pinggirannya. Itulah indeks pencocokan string harus dilakukan kembali, dimulai dari indeks teks sekarang, kemudian dilakukan langkah 2, yaitu pencocokan.

Contoh perbandingan *pattern* dengan teks dengan menggunakan algoritma KMP adalah sebagai berikut:



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

Gambar 2. Pencocokan string dengan Algoritma KMP

Sumber:
https://koding4fun.wordpress.com/2010/05/29/knuth-morris-pratt-algorithm/ (diakses 18 Mei 2017)

Algoritma KMP lebih baik dalam melakukan pencocokan string daripada algoritma *brute force* karena pergeseran *pattern* pada algoritma KMP dilakukan dengan melihat *border function* yang dapat membuat pergeseran yang lebih banyak. Kecepatan pencarian string dengan algoritma KMP juga dapat dihitung dengan melihat kompleksitas algoritma yang dihasilkan dengan menggunakan algoritma KMP. Algoritma KMP memiliki kompleksitas sebesar O(*m*) untuk menghitung *border function* dari *pattern* yang diberikan. Setelah itu, algoritma ini harus melakukan pencarian string dalam teks. Pencarian string dalam teks hanya dilakukan dalam sekali *loop*. Oleh karena itu, kompleksitas dari pencarian *pattern* dalam teks akan memiliki kompleksitas sebesar O(*n*). Jadi, kompleksitas algoritma dengan menggunakan algoritma KMP adalah O(*m*+*n*).

C. Algoritma Boyer-Moore

Algoritma Boyer-Moore juga merupakan salah satu algoritma pencocokan string yang cukup mangkus untuk digunakan. Algoritma ini diciptakan oleh Robert S. Boyer dan J. Strother Moore pada tahun 1977. Algoritma ini cukup unik karena menggunakan teknik *looking-glass*, yaitu mencari *pattern* P dalam teks T dari kanan ke kiri dalam P. Selain itu, algoritma ini juga menggunakan teknik lompat untuk melakukan pergeseran *pattern* P. Pergeseran yang dilakukan pada algoritma Boyer-Moore tidak menggunakan *border function* seperti pada algoritma KMP. Akan tetapi, algoritma Boyer-Moore menggunakan sebuah larik yang berisikan indeks kemunculan terakhir sebuah karakter dalam *pattern*.

Penghitungan kemunculan terakhir dari sebuah *pattern* pada algoritma Boyer-Moore dinyatakan sebagai larik dengan panjang *s*, dengan *s* adalah banyaknya karakter yang mungkin muncul dalam sebuah teks. Misalkan diberikan *pattern* P yang isinya adalah `abacab` dan karakter yang mungkin muncul dalam sebuah teks adalah hanya `a` hingga `e`. Maka, contoh hasil penghitungan kemunculan karakter terakhir dalam

```
P: "abacab"
```

algoritma Boyer-Moore dinyatakan dalam larik di bawah ini.

a	b	c	d	e
4	5	3	-1	-1

Dalam string abacab, kemunculan terakhir karakter a adalah pada indeks ke-4. Oleh karena itu, larik yang berisikan kemunculan karakter terakhir untuk karakter `a` diisi dengan 4, sedangkan karakter `b` muncul terakhir pada indeks ke-5. Karakter `c` dalam *pattern* P muncul terakhir pada karakter indeks ke -3. Karakter `d` dan `e` tidak muncul sama sekali pada *pattern* P yang diberikan, sehingga nilai dari kemunculan terakhir untuk karakter `d` dan `e` di-*assign* dengan nilai -1, yang menandakan bahwa karakter tersebut tidak pernah muncul pada *pattern* yang diberikan.

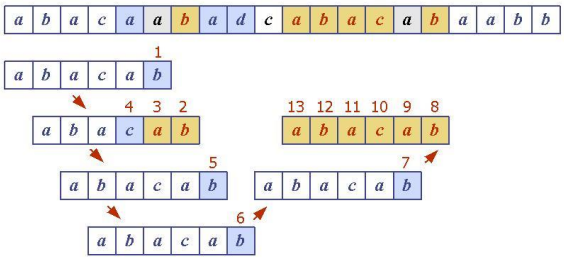
Andaikan diberikan teks T dan *pattern* P. Cara kerja dari algoritma Boyer-Moore adalah sebagai berikut:

1. Pertama-tama, dihitung dulu kemunculan terakhir setiap karakter yang ada dalam *pattern* P.
2. *Pattern* P di-align dengan teks T pada karakter pertama. Kemudian pencocokan karakter dimulai dari kanan (karakter terakhir dari *pattern* P) ke kiri hingga seluruh karakter sama pada *pattern* dengan teks atau ditemukan ketidakcocokan pada karakter teks dengan karakter pada *pattern*.
3. Ada 3 kasus ketidakcocokan dalam pencocokan string dengan algoritma Boyer-Moore:
    - Karakter pada teks yang tidak sama dengan karakter *pattern* ada di *pattern* di posisi yang lebih awal → lakukan alignment pada karakter tersebut

- Karakter pada teks yang tidak sama dengan karakter patter ada di *pattern* di posisi yang lebih akhir → geser 1 karakter
- Jika karakter dalam teks yang berbeda sama sekali tidak ada dalam *pattern* → align karakter pertama *pattern* agar mulai dari T[*i*+1] dimana *i* adalah indeks perbedaan pada teks dengan *pattern*.

Setelah itu, kembali melakukan langkah 2 (langkah pencocokan string).

Contoh pencocokan *pattern* dengan teks dengan menggunakan algoritma Boyer-Moore diilustrasikan dalam gambar di bawah ini.



Gambar 3. Pencocokan string dengan algoritma Boyer-Moore

Sumber: https://koding4fun.wordpress.com/2010/05/29/boyer-moore-algorithm/ (diakses 18 Mei 2017)

Algoritma Boyer-Moore memiliki kompleksitas yang lebih baik dari algoritma *brute force*. Akan tetapi tidak selalu lebih baik daripada algoritma KMP. Ada suatu saat dimana algoritma Boyer-Moore tidak lebih baik daripada algoritma KMP. Algoritma Boyer-Moore memiliki kompleksitas O(*m*+x) untuk melakukan inisialisasi array kemunculan karakter terakhir dari *pattern*. Langkah pencarian dalam algoritma Boyer-Moore memiliki kompleksitas terbaik yaitu O(*n*/*m*) dan kompleksitas terburuk yaitu O(*mn*), sama dengan kompleksitas *worst case brute force*. Salah satu contoh kasus yang menyebabkan algoritma Boyer-Moore berada dalam *worst case* adalah contoh kasus di bawah ini.

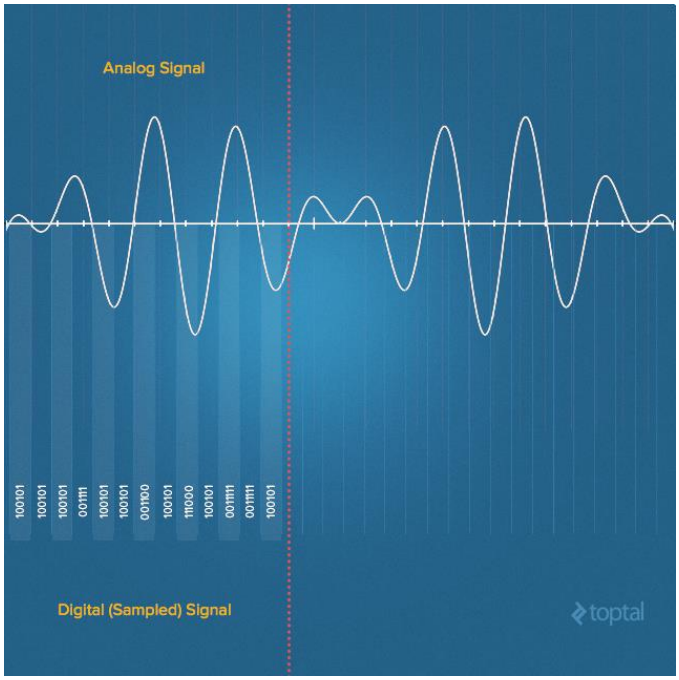
```
T: "aaaaaaaaaaaaaaaaaaaabaaaa"
P: "baaaa"
```

Contoh kasus di atas menyebabkan berkali-kali pencocokan gagal di karakter pertama pada *pattern*. Selain itu, huruf pada teks yang menyebabkan kegagalan berada di sebelah kanan pada *pattern*, menyebabkan banyak pergeseran yang dilakukan menjadi hanya 1 karakter, sehingga *pattern* akan selalu digeser sebanyak 1 kali hingga ditemukan huruf `b` pada teks yang kemudian menyebabkan *alignment* antara teks dan *pattern* pada karakter `b`. Oleh karena itu, dapat disimpulkan dari sini bahwa kompleksitas algoritma untuk *worst case* pada algoritma Boyer-Moore adalah O(*mn*).

III. String Matching untuk Pencocokan Lagu

A. *Proses Pencocokan Lagu*

Dalam rangka melakukan pengenalan lagu, harus dilakukan sebuah *sampling* yang berupa cuplikan lagu yang ingin dicari tahu identitasnya. Lagu yang dalam bentuk "mentah" hanyalah berupa suara yang terdiri dari nada dan lirik. Nada adalah dapat direpresentasikan sebagai gelombang suara yang memiliki frekuensi tertentu. Untuk dapat mencocokkan sebuah cuplikan lagu tersebut dengan lagu-lagu yang sudah tersimpan dalam *database* harus dilakukan konversi dari gelombang suara yang merupakan cuplikan dari lagu tersebut menjadi angka-angka biner yang merepresentasikan lagu tersebut. Oleh karena itu diperlukan konversi sinyal dari analog menjadi digital.



Gambar 4. Konversi sinyal analog ke digital

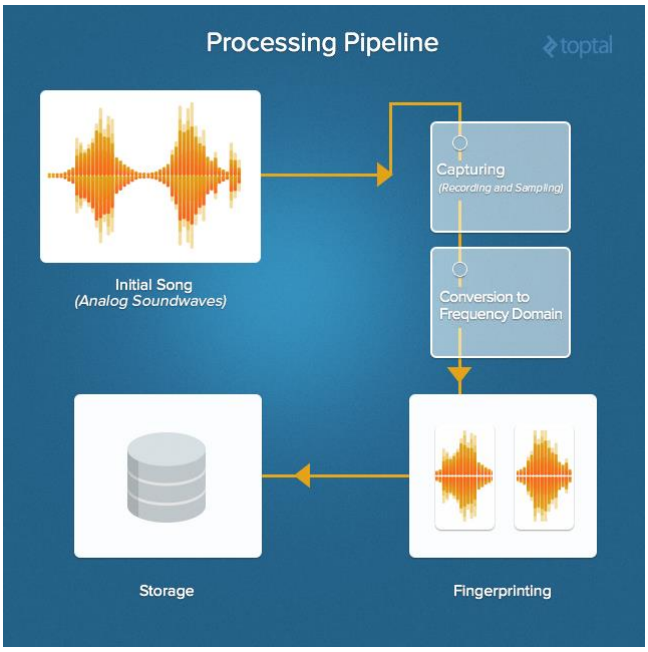
Sumber: https://www.toptal.com/algorithms/shazam-it-music-processing-fingerprinting-and-recognition (diakses 18 Mei 2017)

Hal pertama yang dilakukan untuk mengubah sinyal analog dari sebuah lagu menjadi sinyal digital adalah dengan melakukan konversi gelombang suara dari lagu dari domain waktu, menjadi domain frekuensi. Pengubahan ini dapat dilakukan dengan memanfaatkan transformasi Fourier. Konversi dengan transformasi Fourier ini akan menghasilkan sebuah rekaman gelombang suara yang memiliki domain frekuensi. Setelah didapatkan gelombang suara ini, barulah dilakukan konversi dari domain frekuensi menjadi dalam biner (bentuk digital).

Konversi yang paling mudah yang bisa dilakukan dalam pengubahan sinyal analog ke digital adalah dengan mengubahnya menjadi biner berdasarkan besar frekuensi pada setiap satuan waktu (detik). Misal pada detik ke-1 didapatkan bahwa frekuensi dari lagu yang direkam adalah 10 Hz, maka binernya akan menjadi 1010. Setelah itu, pada detik ke-2,

didapat bahwa besar frekuensi pada lagu yang direkam adalah 13 Hz, jika dikonversi ke biner didapat 1101, dst. Dari hasil konversi untuk setiap satu satuan waktu akan didapatkan sebuah string panjang yang merupakan gabungan dari hasil setiap konversi pada setiap satuan waktu yang terdiri atas 2 abjad, yaitu 0 dan 1.[5] String ini kemudian yang akan dijadikan sebagai pattern yang akan dicari dalam "teks" lagu yang sudah disediakan dalam media penyimpanan atau kamus lagu yang sudah ada.

Jika digambarkan dalam bentuk skema, proses sebuah penentuan sebuah lagu dapat digambarkan dalam skema di bawah ini.



Gambar 5. Alur Proses Pencocokan Lagu

Sumber: https://www.toptal.com/algorithms/shazam-it-music-processing-fingerprinting-and-recognition (diakses 18 Mei 2017)

Teks tempat ditemukannya *pattern* yang ingin dicari akan diambil indeksnya. Indeks tersebut nantinya akan dipakai untuk menentukan atribut-atribut dari lagu yang ingin ditampilkan. Pengambilan judul lagu, penyanyi lagu, dan lain sebagainya diambil berdasarkan indeks yang sudah didapat dari pencocokan pattern dengan "teks" biner lagu yang ada dalam kamus yang ada.

B. Efektivitas Algoritma KMP dan Boyer-Moore

Dengan melakukan konversi terhadap sinyal analog ke sinyal digital, diperoleh sebuah string yang berisi karakter 0 atau 1 (biner). Inilah yang dimanfaatkan sebagai *pattern* untuk melakukan string matching. String matching dilakukan antara *pattern* dengan teks yang diberikan. Teks didapat dari kamus lagu yang sudah ada. Pada bagian ini, akan dilakukan eksperimen untuk membandingkan efektivitas kedua algoritma yang dapat dipakai untuk melakukan string matching, antara lain algoritma KMP dan algoritma Boyer-Moore.

Hasil eksekusi program yang dihasilkan memberikan hasil sebagai berikut:

1. Test case 1

```
Enter text 1 here: 00000000010101110110101001011010001 01
Enter pattern 1 here: 10110
[Test case 1] Comparisons using KMP Algorithm: 23
[Test case 1] Comparisons using BM Algorithm: 33
```

Gambar 6. Test case 1 Pencocokan String

Sumber: Dokumen Pribadi

Pada test case 1 ini, terlihat bahwa jumlah perbandingan yang dilakukan pada algoritma Boyer-Moore berjumlah 10 lebih banyak daripada algoritma KMP. Ini dapat dilihat dari awal, jika digunakan algoritma Boyer-Moore untuk kasus ini, tentu akan selalu terjadi kegagalan di awal-awal pencocokan, yaitu pada pattern[3], yaitu 1. Akan tetapi, kemunculan terakhir 0 sudah dilewati, sehingga pergeseran hanya boleh dilakukan sebanyak 1 karakter. Ini menyebabkan jumlah perbandingan karakter pada *pattern* dengan teks pada Boyer-Moore di awal-awal akan menjadi 2 kali lebih banyak daripada algoritma KMP, karena KMP pasti selalu menemukan kegagalan di awal pencocokan dan bergeser sebanyak 1 karakter.

2. Test case 2

```
Enter text 2 here: 10000101101110101001010111110100010001001011 0
Enter pattern 2 here: 1110100
[Test case 2] Comparisons using KMP Algorithm: 41
[Test case 2] Comparisons using BM Algorithm: 29
```

Gambar 7. Test case 2 Pencocokan String

Sumber: Dokumen Pribadi

Pada test case 2, dapat dilihat bahwa algoritma KMP tidak selalu memberikan hasil yang lebih baik daripada algoritma Boyer-Moore dalam melakukan pencocokan string, terutama string yang berbasis biner. KMP memberikan hasil yang lebih buruk daripada Boyer-Moore kali ini karena kali ini Boyer-Moore dapat memanfaatkan pergeserannya dengan lebih baik, karena kemunculan terakhir karakter 1 berada di lebih kiri daripada *pattern* pada test case 1. Ini akan mengakibatkan jika ditemukan ketidakcocokan pada 2 karakter terakhir, pergeseran bisa dilakukan lebih banyak.

3. Test case 3

```
Enter text 3 here: 0101010100101110101101010001010101101110111010110101 0
Enter pattern 3 here: 110110
[Test case 3] Comparisons using KMP Algorithm: 74
[Test case 3] Comparisons using BM Algorithm: 99
```

Gambar 8. Test case 3 Pencocokan String

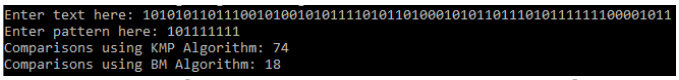
Sumber: Dokumen Pribadi

Pada test case 3, terlihat perbedaan jumlah perbandingan yang cukup ekstrim antara algoritma KMP dan Boyer-Moore. Ini terjadi karena kemunculan terakhir dari karakter 0 dan 1 pada *pattern* muncul bersebelahan, sehingga pergeseran Boyer-Moore pasti akan lebih kecil, menyebabkan ada banyak

perbandingan karakter yang harus dilakukan pada algoritma Boyer-Moore. Pada algoritma KMP, border functionnya banyak yang tidak bernilai 0, ini menjadi keuntungan bagi algoritma KMP dalam melakukan pencocokan string.

Dari ketiga *test case* yang diberikan di atas, cukup dapat disimpulkan bahwa algoritma KMP lebih baik dalam melakukan pencocokan string untuk pencocokan lagu. Ini karena algoritma Boyer-Moore dapat dikatakan cukup lemah dalam melakukan pencocokan string dengan variasi karakter yang sedikit, apalagi apabila karakter yang berbeda tersebut muncul bersebelahan di akhir-akhir *pattern*. Ini menyebabkan pergeseran yang harus dilakukan oleh Boyer-Moore juga haruslah sedikit, yaitu sebanyak 1. Dengan kecilnya pergeseran yang dilakukan pada algoritma Boyer-Moore, menyebabkan banyaknya operasi perbandingan karakter yang harus dilakukan untuk mencari kesamaan pattern dengan teks yang diberikan.

Akan tetapi, algoritma Boyer-Moore mungkin saja menjadi jauh lebih baik daripada algoritma KMP dalam pencocokan lagu, tetapi hanya terbatas apabila *pattern* yang diberikan memiliki kemunculan terakhir karakter yang tidak berdekatan. Salah satu contohnya diberikan pada gambar di bawah ini.



Gambar 9. Test case Boyer-Moore unggul

Sumber: Dokumen Pribadi

Algoritma Boyer-Moore di sini dapat memberikan hasil yang jauh lebih baik daripada algoritma KMP, karena kemunculan karakter terakhir dari 0 dan 1 sangatlah jauh, sehingga pergeseran yang dilakukan oleh Boyer-Moore dapat dilakukan dengan lebih banyak juga, sehingga dapat meminimalisasi jumlah perbandingan karakter yang harus dilakukan.

IV. Simpulan

Seluruh algoritma *string matching* dapat digunakan dalam rangka pencocokan lagu, mulai dari algoritma yang paling naif (*brute force*), ataupun algoritma yang lebih kompleks, seperti algoritma KMP dan algoritma Boyer-Moore. Algoritma KMP lebih baik dalam melakukan pencocokan string dengan data yang ada, karena string yang ingin dicocokkan berbasis biner yang terdiri atas karakter 0 dan 1. Hal ini menjadi kelemahan bagi algoritma Boyer-Moore dalam pencocokan string terutama apabila kemunculan terakhir dari kedua karakter tersebut berada berdekatan di akhir pattern. Akan tetapi, algoritma Boyer-Moore dapat bekerja jauh lebih baik apabila letak kemunculan terakhir kedua karakter berada agak jauh berbeda, sehingga pergeseran yang dilakukan oleh algoritma Boyer-Moore dapat dilakukan dengan lebih banyak dan lebih cepat untuk melakukan pencocokan lagu.

V. Ucapan Terima Kasih

Pertama-tama, penulis ingin bersyukur kepada Tuhan Yang Maha Esa karena berkat dan bimbingan-Nya lah, penulis diberikan kemampuan untuk membuat makalah ini walaupun masih terdapat sangat banyak kekurangan. Penulis juga ingin berterima kasih kepada orangtua penulis yang sudah memberi dukungan dalam bentuk doa dan material, sehingga penulis dapat menyelesaikan makalah ini. Penulis juga berterima kasih kepada seluruh dosen IF2211 Strategi Algoritma, Ibu Masayu Leylia Khodra, S.T., M.T., Ibu Dr. Nur Ulfa Maulidevi, S.T., M.Sc., terutama Bapak Dr. Ir. Rinaldi Munir, M.T. selaku dosen pengampu mata kuliah IF2211 Strategi Algoritma dan dosen untuk kelas K-03. Saya berterima kasih juga atas bimbingannya sepanjang berjalannya perkuliahan IF2211 Strategi Algoritma, sehingga penulis dapat mengerti materi String Matching. Saya juga berterima kasih kepada teman-teman Teknik Informatika 2015 yang sudah mendukung dalam pembuatan makalah ini mengenai analisis algoritma string matching pada pencocokan lagu sehingga dapat selesai dengan cukup baik.

Referensi

[1] Munir, Rinaldi. 2006. *Diktat Kuliah IF2211 Strategi Algoritma*. Bandung: Institut Teknologi Bandung.

[2] https://brainwavepowermusic.com/blog/blog/your-musical-taste-shows-and-affects-your-personality-and-behavior. Diakses 18 Mei 2017.

[3] http://www-igm.univ-mlv.fr/~lecroq/string/node14.html. Diakses 18 Mei 2017.

[4] https://www.toptal.com/algorithms/shazam-it-music-processing-fingerprinting-and-recognition. Diakses 18 Mei 2017.

[5] http://www.bbc.co.uk/education/guides/zpfdwmn/revision/3. Diakses 18 Mei 2017.

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, 19 Mei 2017

Ferdinandus Richard
13515066