# Penerapan Algoritma String Matching dalam Aplikasi *Music Recognition*

Jonathan (13516058)
Program Studi Teknik Informatika
Institut Teknologi Bandung
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
jonathantjandra.98@outlook.com

***Abstract*—Seiring dengan perkembangan teknologi, maka penggunaan smartphone semakin meningkat dan fitur-fitur dasar yang sudah ada dapat dilengkapi dengan aplikasi untuk berbagai macam sistem operasi. Salah satu fitur yang popular adalah pengenalan lagu, atau music recognition. Aplikasi pengenalan lagu sendiri menggunakan algoritma string matching untuk mencocokkan pattern dan lagu yang telah disampling yaitu gelombang lagu yang telah diubah dari analog menjadi digital.**

***Keywords—String Matching, Music Recognition, Algoritma, Knuth-Morris-Pratt (KMP), Boyer-Moore, Brute Force***

## I. INTRODUCTION

Perkembangan teknologi yang sangat pesat membuat ponsel pintar menjadi sesuatu bagian yang tidak dapat dilepaskan dari kehidupan sehari-hari. Sebuah ponsel pintar dapat menyimpan aplikasi di memori internal ataupun eksternal dan dapat mengunduh aplikasi tersebut di toko aplikasi sesuai sistem operasinya masing-masing ataupun dari internet. Jenis aplikasi yang tersedia sangat beragam sehingga yang disorot kali ini adalah music recognition.

Seringkali, ketika kita sedang bersantai atau mengerjakan hal lain, kita mendengar sebuah lagu dan tidak dapat mengingat judul lagu tersebut, tetapi kita dapat menggunakan aplikasi music recognition ini untuk membantu kita mengetahui informasi dari lagu yang kita dengar.

Aplikasi music recognition adalah aplikasi yang mengenali lagu yang sedang dimainkan, dan direkam secara otomatis oleh aplikasi untuk dicocokkan dengan katalog lagu pada server aplikasi atau penyedia layanan dan menampilkan ke layar informasi tentang lagu tersebut, misalnya Penyanyi, Album, Durasi, bahkan menyediakan link untuk membeli lagu tersebut di berbagai situ seperti Google Play, Apple iTunes Store, dll.



Gambar 1 dan 2 Contoh Aplikasi Music Recognition – Shazam dan Soundhound

Sumber :
https://www.shazam.com/resources/6745425c24f3b5b93d238844d5a86f04fb744f17/shazambrand.jpg (diakses 13 Mei 2017)

https://i0.wp.com/www.deteched.com/wp-content/uploads/2017/06/SoundHound-Tout-Home-Page-Asset.jpg?fit=800%2C520 (diakses 13 Mei 2017)

Sehingga dengan adanya aplikasi ini, maka diharapkan akan membantu orang yang ingin mengenal sebuah lagu. Seperti yang telah dibahas sebelumnya, pencocokan antara pattern (recording lagu) dan database dilakukan oleh aplikasi ini dengan menggunakan algoritma string matching. Sehingga lagu dapat diidentifikasi dan pengguna aplikasi dapat mengetahui lagu apa yang sedang didengarkan.

## II. TEORI DASAR ALGORITMA

### A. *Algoritma String Matching*

String matching pada dasarnya adalah sebuah proses yang dilakukan sebagai berikut :

1. Diberikan teks dengan panjang n karakter
2. Diberikan pattern dengan panjang m karakter, diasumsikan panjang m jauh lebih pendek dibandingkan n (m <<< n)
3. Mencari string dari pattern pada teks.

Algoritma yang digunakan dalam aplikasi-aplikasi ini adalah string matching, dan algoritma string matching yang

digunakan dapat memiliki 3 alternatif, yaitu Knuth-Morris-Pratt (KMP), Boyer-Moore, dan Brute Force.

### B. *Algoritma Brute Force*

Brute force dapat dikatakan algoritma yang tidak perlu dipikir lagi, atau sangat straightforward dalam penggunaannya. Sesuai dengan namanya, penyelesaian dengan algoritma ini memanfaatkan kekuatan murni. Semua masalah pasti dapat diselesaikan dengan algoritma brute force meskipun belum tentu optimal.

Algoritma brute force dalam pattern matching dilakukan dengan cara :

1. Memulai pencocokan pattern dengan text dari karakter pertama.
2. Pencocokan dilakukan hingga panjang pattern, jika cocok maka pencocokan berhenti.
3. Jika terjadi ketidakcocokan di titik manapun, geser pattern dan ulangi pencocokan.

Sehingga, algoritma brute force akan memiliki kasus terbaik ketika terjadi kecocokan di posisi pertama atau dalam kasus cocok terjadi di akhir string tetapi awal pattern tidak pernah sama dengan awalan text, dan kasus terburuk adalah ketika cocok terjadi di akhir string dan semua karakter selalu dicocokan.

Contoh kasus terbaik :

`Text : “Sidewinder is the best ship.”`

`Pattern : “Side”`

`Text : “This text ends in xor.”`

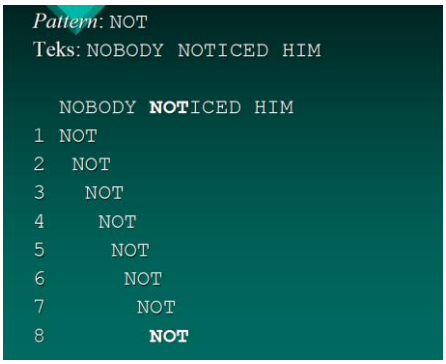
`Pattern : “xor”`

Contoh kasus terburuk :

`Text : “aaaaaaaaaaaaaaaaaaaaaaaaaw”`

`Pattern : “aaw”`

Kompleksitas untuk kasus terbaik adalah O(n),sedangkan untuk kasus terburuk adalah O(mn). Sedangkan pada umumnya, kompleksitasnya adalah O(m+n).




Gambar 3 Contoh Brute Force

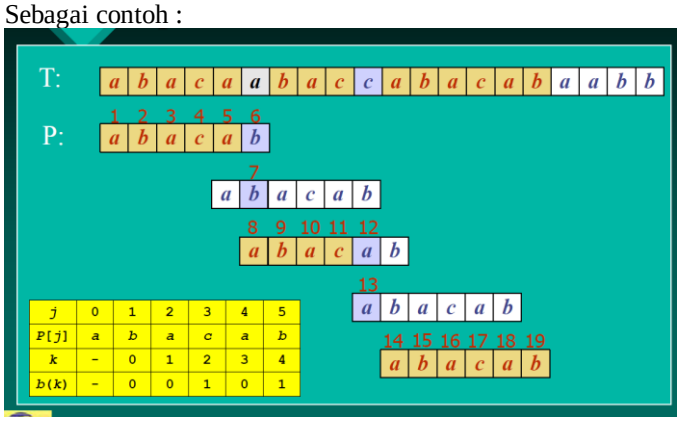
Sumber : Slide Kuliah Pencocokan String. Rinaldi Munir. 2018.

### C. *Algoritma Knuth-Morris-Pratt*

Pencocokan pada algoritma KMP pada dasarnya sama seperti pada algoritma brute force, hanya saja pengeseran tidak dilakukan satu-satu, tetapi bisa beberapa karakter sekaligus dengan adanya border function. Sehingga pencocokan yang sia-sia dilakukan tidak ada.

Border function sendiri adalah mencari prefix terpanjang yang juga merupakan anggota suffix. Contohnya adalah dalam teks “Aquafina”, maka border function untuk huruf tersebut adalah semua 0 atau 1 bergantung pada indeks awal mulainya.

Sebagai contoh :




Gambar 4 Contoh Knuth-Morris-Pratt dengan Border Function

Sumber : Slide Kuliah Pencocokan String. Rinaldi Munir. 2018.

Terlihat bahwa border function dihitung terlebih dahulu, sehingga pergeseran dilakukan lebih efisien dibandingkan dengan menggunakan brute force.

Terkait kompleksitas, untung menghitung border function, karena hanya untuk panjang karakter saja, maka O(m), sedangkan untuk pencarian string adalah O(n), karena tidak seperti brute force, pergeseran bergantung pada nilai border function. Sehingga kompleksitas algoritma KMP adalah O(m+n).

KMP memiliki kelemahan jika alphabet yang digunakan besar, maka kondisi mismatch atau ketidakcocokan akan lebih besar, selain itu Knuth-Morris-Pratt juga memiliki kelemahan apabila kesalahan sering terjadi di awal pattern, karena border function tidak dapat bekerja maksimal, karena jika kesalahan terjadi di akhir maka nilai border function lebih besar sehingga pergeseran dapat dilakukan lebih banyak.

Algoritma KMP untuk Java (dari Slide Kuliah Pencocokan String). Rinaldi Munir. :

```
public static int[] computeFail(String pattern) {
        int fail[] = new int[pattern.length()];
        fail[0] = 0;
        int m = pattern.length();
        int j = 0;
        int i = 1;
        while (i < m) {
                if (pattern.charAt(j) ==
                pattern.charAt(i)) { //j+1 chars
                matchfail[i] = j + 1;
                i++;
                j++;
                } else if (j > 0) // j follows
                matching prefix
                j = fail[j-1];
                else {
                // no matchfail[i] = 0;
                i++;}
        }
        return fail;
        } // end of computeFail()

public static int kmpMatch(String text, String
pattern) {
        int n = text.length();
        int m = pattern.length();
        int fail[] = computeFail(pattern);
        int i=0;
        int j=0;
        while (i < n) {
        if (pattern.charAt(j) == text.charAt(i)) {
        if (j == m -1)
        return i -m + 1; // match
        i++;
        j++;
        }
        else if (j > 0)
        j = fail[j-1];
        else
        i++;
        }
        return -1; // no match
        } // end of kmpMatch()
```

*D. Algoritma Boyer-Moore*

Boyer-Moore menggunakan 2 teknik, yaitu Teknik looking glass, dimana pencocokan string dilakukan dari belakang pattern, lain halnya dengan algoritmaq brute force dan Knuth-Morris-Pratt yang telah dijelaskan sebelumnya, dan Teknik kedua yaitu character jump. Terdapat 3 kasus untuk character jump saat terjadi ketidakcocokan pada karakter tertentu misalnya "x". Kasus pertama yaitu jika terdapat "x" pada pattern, maka geser pattern ke kanan hingga x pada text sejajar dengan x pada pattern. Kasus kedua yaitu sama seperti kasus pertama, akan tetapi pergeseran ke kanan hingga posisi "x" terakhir ditemukan tidak dapat dilakukan, sehingga pattern hanya digeser sekali ke kanan. Kasus ketiga adalah kasus selain 1 dan 2, dan yang dilakukan adalah posisikan awal pattern dengan posisi text selanjutnya.

Untuk memudahkan pergeseran yang menggunakan last occurrence ini, maka Boyer-Moore memiliki fungsi last occurrence. Last occurrence ini mencatat kemunculan terakhir sebuah karakter pada sebuah pattern, dan jika ada karakter yang terdapat pada alphabet dan tidak ada pada pattern maka indeks akan ditulis -1, alias tidak ada.


Gambar 5 Contoh Boyer-Moore dengan Last Occurrence Function

Sumber : Sumber : Slide Kuliah Pencocokan String. Rinaldi Munir. 2018.

Kompleksitas algoritma Boyer-Moore adalah untuk terburuknya O(mn + A). Hal ini dapat dilihat dari contoh berikut :

```
Text : "aaaaaaaaaaaaaaaaaaaaaaaa"
Pattern : "waa"
```

Hal ini dikarenakan ketidakcocokan terjadi di karakter paling awal, sedangkan Boyer-Moore melakukan pencocokan dari akhir pattern, sehingga setiap karakter selalu dicocokan hingga akhir, dan digeser satu per satu karena karakter w tidak ada dalam text.

Boyer-Moore tidaklah selalu lebih baik dari KMP, akan tetapi Boyer-Moore berkebalikan dengan Knuth-Morris-Pratt dalam 1 hal yaitu, efisien dalam alphabet yang besar, dibandingkan dengan alphabet yang kecil, Hal ini dikarenakan last occurrence tadi, karena jika -1 maka langsung masuk ke kasus 3, sedangkan jika ada, maka dipertimbangkan untuk kasus 1 atau kasus 2.

Algoritma Boyer-Moore dari Slide Kuliah Pencocokan String. Rinaldi Munir.

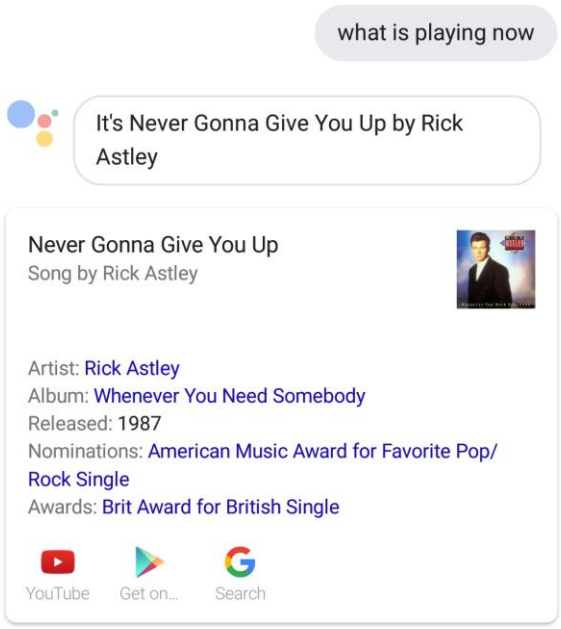
```
public static int[] buildLast(String pattern)
/* Return array storing index of
lastoccurrence of each ASCII char in
pattern. */{
        int last[] = new int[128]; // ASCII
        char set
        for(int i=0; i < 128; i++)
                last[i] = -1; // initialize array
        for (int i = 0; i < pattern.length();
        i++)
                last[pattern.charAt(i)] = i;
        return last;
} // end of buildLast()

public static int bmMatch(String text, String
pattern) {
        int last[] = buildLast(pattern);
        int n = text.length();
        int m = pattern.length();
        int i = m-1;
        if (i > n-1)
                return -1; // no match if
        pattern is // longer than text
        int j = m-1;
        do {
                if (pattern.charAt(j) ==
        text.charAt(i))
                if (j == 0)
                        return i; // match
                else {
                // looking-glass technique
                i--;
                j--;
                }else { // character jump
                technique
                int lo =
                last[text.charAt(i)]; //last
                occ
                i = i + m -Math.min(j,
                1+lo);
                j = m -1;
                }
        } while (i <= n-1);
        return -1; // no match
        } // end of bmMatch()
```

## III. MUSIC RECOGNITION

Aplikasi-aplikasi music recognition sudah banyak beredar di toko-toko aplikasi, contohnya telah disebutkan diantaranya Shazam,Soundcloud, Track ID, akan tetapi tidak hanya itu, sekarang personal virtual assistant milik Google, dan Apple juga mampu mengenali lagu. Berikut sebagai contoh :

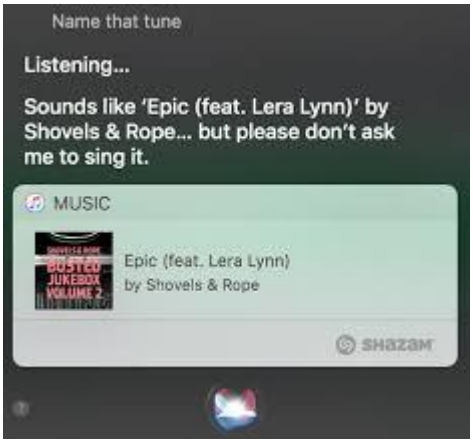
1. Google Assistant




Gambar 6. Google Assistant UI

Sumber : Google Images

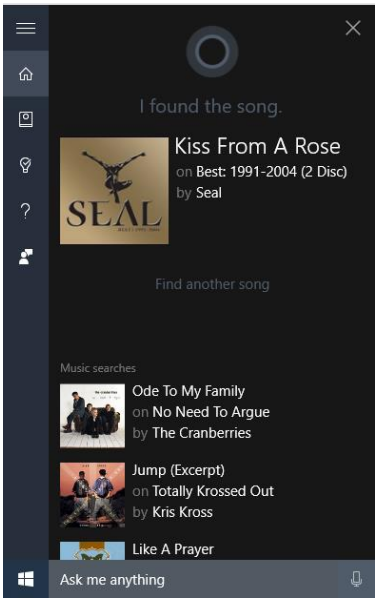
2. Apple Siri




Gambar 7 : Apple Siri UI

Sumber : Google Images

3. Microsoft Cortana




Gambar 8. Microsoft Cortana UI

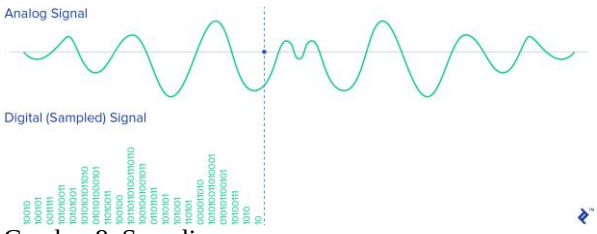
Sumber : Google Images

Dengan adanya berbagai aplikasi dan asisten pribadi ini, tentu saja penggunaan music recognition ini meningkat, dan fitur ini dapat dibuat dan diimplementasikan tentu saja setelah berbagai proses. Berikut adalah teknik untuk mencocokkan lagu,

1. Sampling

   Sampling sendiri adalah proses mengubah rekaman suara analog menjadi digital. Di udara, sebuah gelombang suara adalah sinyal tekanan yang kontinu, mikrofon mengubah gelombang suara menjadi sinyal elektrik, namun bentuknya tetap kontinu sehingga agar bisa diproses maka sinyal ini harus diubah terlebih dahulu bentuknya menjadi sinyal diskrit.

   Transformasi dari analog ke digital ini dilakukan menggunakan transformasi Fourier (Fourier transform).

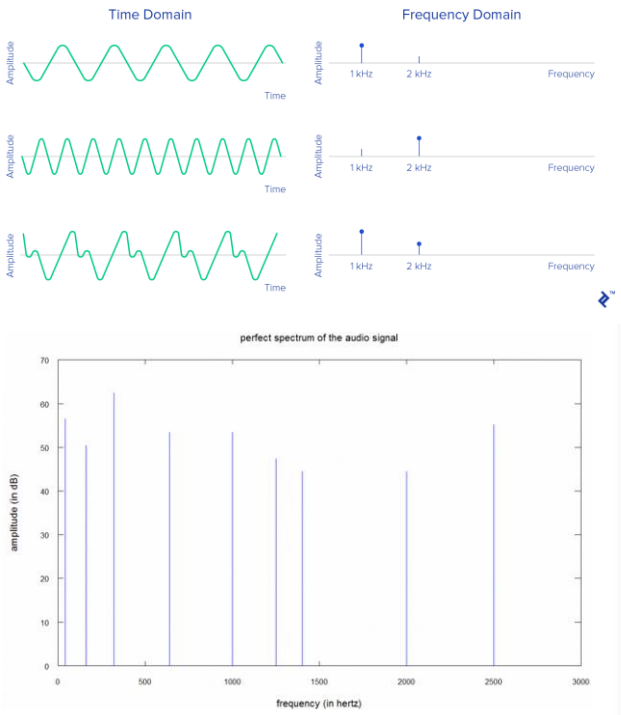



Gambar 9. Sampling

Sumber : https://www.toptal.com/algorithms/shazam-it-music-processing-fingerprinting-and-recognition

2. Discrete Fourier Transform

   Transformasi Fourier seperti yang telah dinyatakan sebelumnya digunakan untuk transformasi sinyal analog menjadi digital, mengubah dari domain waktu, menjadi domain frekuensi.




Gambar 10 dan 11. Discrete Fourier Transform Illustration (Time Domain – Frequency Domain)

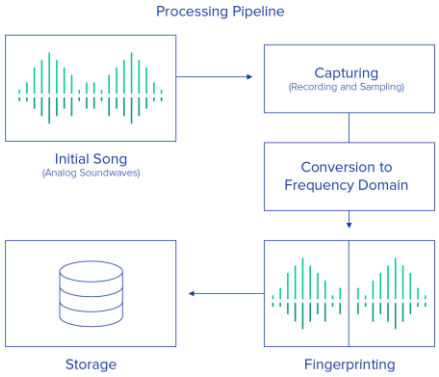
Sumber : https://www.toptal.com/algorithms/shazam-it-music-processing-fingerprinting-and-recognition

Karena batasan dalam mengubah dari kontinu menjadi diskrit, make transformasi dilakukan dalam Batasan waktu tertentu, misalnya tiap 0.1s, ini disebut windows function.

Setelah didapat frekuensi dalam rentang-rentang waktu tertentu, maka sesuai dengan encoding database yang disimpan di server, maka format data yang telah ditransform dapat disesuaikan, dalam hal ini salah satu cara yang dapat dipakai adalah mentransformasi frekuensi menjadi biner, jika dibatasi menjadi 8 bit untuk setiap pencatatan , maka jika frekuensi yang didapat adalah 255 maka dapat dituliskan 11111111, jika 100 maka dapat dituliskan 1100100.

Selanjutnya, pattern yang sudah didapat ini disubmit ke server dan dicocokan dengan database yang ada, jika cocok maka ditulis ada, , dan jika tidak cocok, maka terdapat opsi untuk mengulanginya kembali.

Contoh ringkas :




Teknik encoding menjadi biner itu hanya salah satu contoh, dan untuk menjaga keamanan data maka dapat digunakan hash untuk memetakan bahwa biner ini masuk ke hash mana. Dalam aplikasi atau proses ini, Teknik encoding dinamakan fingerprinting.

Meskipun terdapat Teknik-teknik di atas, beserta fingerprinting, bukan tidak mungkin terdapat false positive ataupun kesalahan dalam proses music recognition. Sebagai contohnya adalah pada saat perekaman mungkin terdapat noise sehingga suara menjadi kurang jelas dan bisa mengacu pada proses fingerprinting yang salah juga. Kemudian juga terdapat kemungkinan bahwa fingerprint sangat mirip sehingga saat mengambil data dari server maka yang dimunculkan akan salah.

Selain itu terdapat juga factor karakter suara seperti timbre, sehingga string matching tidak dapat dilakukan semudah itu pada music sehingga harus dicocokan dengan factor fingerprint lain untuk menandakan alat music tertentu.

## IV. KESIMPULAN

String matching dapat digunakan untuk music recognition, karena file music diubah dari bentuk analog menjadi bentuk digital dan di fingerprinting sesuai dengan standar perusahaan masing-masing, dalam hal ini menjadi biner sesuai frekuensi, atau mapping dengan hash nantinya. Ketiga algoritma dapat digunakan dalam penyelesaian string matching untuk masalah pencocokan lagu atau music recognition, namun jika disimpan dalam bentuk biner, maka KMP dapat dikatakan lebih unggul karena alphabet yang digunakan kecil (0 dan 1) sehingga border function memiliki angka yang besar karena kemungkinan kombinasi jauh lebih kecil ketimbang menggunakan decimal ataupun hexadecimal (basis 16 – 0..9 dan A..F).

## ACKNOWLEDGMENT

Saya ingin mengucap syukur kepada Tuhan Yang Maha Esa karena berkat dan bimbingan maka makalah ini dapat selesai tepat waktu, dan kepada orang tua yang selalu mendukung kuliah saya, dan Bapak Rinaldi Munir, dan Ibu Masayu selaku dosen kelas K1 yang telah mengajar saya materi Strategi Algoritma selama satu semester ini.

## REFERENCES

[1] Munir,Rinaldi. Slide Kuliah Pencocokan String.. 2018
[2] http://coding-geek.com/how-shazam-works/
[3] Jovanovic,Jovan. https://www.toptal.com/algorithms/shazam-it-music-processing-fingerprinting-and-recognition

## 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 Mei 2018

Jonathan - 13516058