Penerapan Pencocokan String dalam Pencarian Musik Sederhana

Farras Mohammad Hibban Faddila - 13518017
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
faddilafarras@gmail.com

***Abstrak*—Makalah ini menunjukkan sebuah variasi dari algoritma pencocokan string, di mana string pattern yang ingin dicari tidak harus muncul sama persis pada string teks tempat pencarian dilakukan, tetapi memiliki karakteristik yang sama, yakni string hasil selisih antar karakter pada pattern muncul pada string hasil selisih antar karakter pada teks. Algoritma ini dapat diaplikasikan pada pencarian musik sederhana dengan menganggap frekuensi sebagai karakter pada string yang akan dicocokkan. dengan asumsi satuan waktu pada teks dan pattern adalah sama.**

Keywords—Pencocokan String, Musik, Frekuensi, Audio

I. PENDAHULUAN

Musik merupakan sebuah hal yang tidak dapat dilepaskan dalam keseharian. Menurut data yang dikeluarkan oleh IFPI (International Federation of the Phonographic Industry) pada 2019, rata-rata tiap orang menghabiskan sekitar 18 jam untuk mendengarkan musik. Ini berarti sekitar 2.5 jam setiap harinya.

Dalam perkembangannya, muncul sebuah permasalahan mengenai musik terkait dengan pencarian sebuah lagu. Setiap lagu pada dasarnya memiliki sebuah identitas yang membedakan lagu tersebut dengan yang lainnya, seperti judul, pencipta atau pengarang, maupun identitas-identitas lainnya. Oleh karena itu, apabila informasi mengenai identitas lagu tersebut disuplai ke dalam sebuah *search engine*, maka dengan mudah diperoleh lagu tersebut, sehingga dapat didengar dan dinikmati. Tetapi, dalam banyak kasus, hal yang terjadi adalah sebaliknya. Terkadang secara tidak sengaja kita mendengar sebuah lagu tanpa mengetahui identitas dari lagu tersebut, dan kita ingin mengetahui judulnya. Informasi yang dimiliki hanyalah barisan nada-nada lagu tersebut, dan informasi yang ingin diperoleh adalah judul dari lagu tersebut. Pencarian dapat terbantu apabila ada informasi lebih yang terkait dengan teks, seperti contohnya lirik lagu. Namun, apabila yang diketahui hanya rangkaian nadanya saja, pencarian akan menjadi sulit.

Seiring dengan kemajuan zaman, pada saat ini sudah banyak berkembang teknologi-teknologi untuk melakukan pencocokan lagu. Salah satu aplikasi pencarian lagu adalah Shazam. Algoritma yang diterapkan pada aplikasi tersebut adalah membaca audio dan membuat larik berdasarkan audio tersebut, mengonversi larik yang diperoleh menjadi larik lain yang berisi frekuensi suara dengan menggunakan algoritma Fast Fourier Transform, lalu membentuk sebuah *fingerprint* dari lagu tersebut berdasarkan frekuensi yang paling penting. *Fingerprint* ini lah yang nantinya akan dicocokkan.

Berbekal rasa keingintahuan penulis akan metode dibalik bagaimana data suara diproses dan diprogram pada komputer, serta pengalaman pribadi akan sulitnya mencari sebuah lagu tanpa mengetahui judulnya, maka penulis mengadaptasi salah satu algoritma pencocokan string untuk menunjukkan sebuah cara mencocokkan audio, yakni dengan menggunakan algoritma Knuth-Morris-Pratt. Asumsi yang dibuat dalam menggunakan algoritma ini adalah satuan waktu pada lagu yang dicari serta seluruh lagu yang ada pada basis data adalah sama (ketukannya memiliki tempo yang sama) karena algoritma yang akan digunakan belum dapat menangani kasus di mana ada perbedaan tempo.

II. DASAR TEORI

A. *String*

Dalam bidang pemrograman komputer, string secara istilah merupakan larik yang beranggotakan karakter-karakter. String dapat diasosiasikan secara informal sebagai sebuah kata, karena kata juga terdiri atas beberapa karakter (huruf). Walaupun begitu, berbeda dengan kata, sebuah string tidak harus memiliki arti atau makna yang definitif, karena string hanya bertujuan untuk menyimpan data saja. Oleh karena itu, string juga dapat dipandang sebagai sebuah struktur data, dengan informasi-informasi yang disimpan oleh string tersebut adalah karakter-karakter yang dimiliki, jumlah total karakter yang dimiliki, dan urutan karakter-karakter tersebut.

Dalam makalah ini, definisi string akan diekstensi sehingga tidak hanya berarti larik yang beranggotakan karakter saja. Karakter yang dapat disimpan oleh sebuah string diperluas menjadi objek apapun, yang selanjutnya akan disesuaikan dengan data yang akan disimpan.

Definisikan sebuah string S. S dapat dituliskan sebagai berikut ini,

$$S = S_0S_1S_2S_3S_4...S_{n-1}$$

dengan S_0, S_1, S_2, S_3, S_4, ..., S_{n-1} masing-masing merupakan karakter yang dimiliki oleh S. String S tersebut memiliki panjang n karena memuat sebanyak n buah karakter. S_i disebut sebagai karakter ke-i, atau indeks ke-i dari S. Karakter-karakter tersebut tidak selalu harus berbeda. Indeks string S tersebut dimulai oleh nol, bukan satu, sehingga dikenal dengan istilah *zero-based-indexing*. Hal ini mengakibatkan indeks berakhir di n-1, bukan n. Apabila n bernilai nol, maka S tidak memiliki karakter sama sekali dan dalam hal ini S disebut sebagai string kosong.

Dalam beberapa bahasa pemrograman, notasi atau operator kurung siku ([]) digunakan untuk pengaksesan sebuah karakter string. Untuk mengakses elemen ke-i dari sebuah string S digunakan ekspresi S[i]. Sebagai contoh, jika diberikan string dengan nama IF berikut ini,

IF = “informatika“

maka berlaku IF[0] = i, IF[2] = f, IF[10] = a. Lebih lengkapnya dapat dilihat pada tabel berikut ini.

TABEL 1. STRING INFORMATIKA

i	*IF[i]*
0	i
1	n
2	f
3	o
4	r
5	m
6	a
7	t
8	i
9	k
10	a

Dalam sebuah string dikenal beberapa istilah prefix, suffix, dan substring. Prefix dari sebuah string S merupakan sebuah string yang dapat dibentuk dengan cara menghapus beberapa karakter terakhir dari S, atau tidak menghapus karakter S sama sekali. Sebagai contoh, mengacu pada string IF di atas, beberapa prefix dari string IF tersebut adalah “info”, “informat", “informatika”, dan “i”. Suffix dari sebuah string S merupakan sebuah string yang dapat dibentuk dengan cara menghapus beberapa karakter pertama dari S, atau tidak menghapus karakter S sama sekali. Sebagai contoh, beberapa suffix dari string IF di atas adalah “ika”, “a”, “matika”. Perlu diperhatikan bahwa S dan string kosong merupakan prefix sekaligus suffix dari S itu sendiri.

Substring secara bahasa berarti string bagian, dan secara istilah merupakan sebuah string yang merupakan bagian dari string yang lain. Secara formal, substring dari string S merupakan sebuah string yang dapat dibentuk dengan cara menghapus sebuah prefix dan sebuah suffix dari S. Dengan kata lain, substring dari sebuah string S merupakan barisan kontigu beberapa karakter pada S. Sebagai contoh, beberapa substring dari string IF adalah “for”, “format”, dan “rmatik”. Prefix dan suffix juga merupakan sebuah substring. Untuk melambangkan substring dari S yang dimulai dari indeks ke-a hingga ke-b, digunakan notasi kurung siku, yakni S[a..b]

Selain itu, didefinisikan juga sebuah string f(S) yang disebut sebagai string selisih dari S. Sebuah string S memiliki string selisih jika dan hanya jika operator pengurangan terdefinisi pada tipe data karakter-karakternya. Apabila [S_0, S_1, …, S_{n-1}] merupakan larik yang berisi karakter-karakter dari S, maka f(S) akan memiliki karakter-karakter yang direpresentasikan oleh larik berikut ini:

$$[S_1\text{-}S_0, S_2\text{-}S_1, S_3\text{-}S_2, \ldots, S_{n-2}\text{-}S_{n-3}, S_{n-1}\text{-}S_{n-2}].$$

Larik tersebut memiliki n-1 buah anggota, sehingga panjang dari f(S) adalah n-1. Sebuah string yang panjangnya kurang dari satu tidak memiliki string selisih. Apabila terdapat sebuah string K yang direpresentasikan oleh larik [1,2,7,4,100], maka f(K) terdefinisi karena operator pengurangan berlaku pada bilangan bulat, dan panjang dari K lebih dari satu. Larik yang menunjukkan string f(K) adalah:

[1, 5, -3, 96].

B. Pencocokan String

Persoalan pencocokan string merupakan persoalan yang melibatkan dua buah string dan mencari tahu apakah salah satu string tersebut merupakan substring dari string yang satunya. Kedua string ini disebut dengan teks (T) dan pattern (P), dengan pattern merupakan string yang ingin dicari tahu keberadaannya pada teks. Biasanya, teks memiliki jumlah karakter yang jauh lebih banyak dibandingkan pattern, karena pattern hanya merupakan keyword atau frasa yang ingin dicari keberadaannya pada teks. Salah satu kemunculan persoalan ini pada *real-world problem* adalah pada fitur *find* milik sebuah perangkat lunak yang menampilkan sebuah teks, Dengan memasukkan kata yang ingin dicari pada fitur *find* tersebut, kata tersebut akan berperan sebagai P, dan teks tempat dilakukan pencarian akan berperan sebagai T. Selain itu, ada beberapa permasalahan lain yang dapat dimodelkan dengan pencocokan string, yakni pencarian pola asam amino pada rantai DNA serta pencocokan sidik jari.

Terdapat beberapa algoritma yang dapat dilakukan untuk melakukan pencocokan string, yakni algoritma Brute-Force, algoritma Knuth-Morris-Pratt (KMP) dan algoritma Boyer-Moore. Selanjutnya, notasikan m sebagai panjang dari P dan n sebagai panjang dari T, dan

$$T = T_0T_1T_2\ldots T_{n-2}T_{n-1},$$

serta,

$$S = S_0S_1S_2\ldots S_{m-2}S_{m-1}.$$

Kegagalan pada saat proses pencocokan string disebut sebagai *mismatch*.

1) Algoritma Brute Force

Algoritma brute force akan melakukan pengecekan pada seluruh lokasi yang mungkin menjadi tempat kemunculan P pada T. Dengan kata lain, algoritma ini akan melakukan iterasi pada setiap substring dari T yang memiliki panjang m, lalu setiap substring tersebut dicocokkan apakah bernilai sama dengan P atau tidak.

Perhatikan bahwa string T memiliki sebanyak n-m+1 buah substring yang memiliki panjang m, yakni substring-substring yang memiliki karakter pertama T_0, T_1, T_2, …, T_{n-m} berturut-turut (T_{n-m+1}, …, T_{n-2}, T_{n-1} tidak dapat digunakan sebagai string pertama pada sebuah substring dengan panjang m karena banyaknya karakter yang terletak di sebelah kanan mereka secara inklusif bernilai kurang dari m). Oleh karena itu, terdapat n-m+1 kemungkinan string yang akan dicek dengan P.

Setiap pengecekan P dengan substring-substring tersebut akan menghabiskan maksimal sebanyak m buah pengecekan karakter. Hal ini disebabkan mereka memiliki masing-masing m buah karakter, dan setiap karakter perlu dicek sehingga membutuhkan setidaknya m buah pengecekan. Namun, apabila sudah terjadi mismatch sebelum m buah pengecekan terjadi, dapat disimpulkan bahwa kedua string tersebut tidak sama sehingga algoritma dapat langsung berlanjut ke substring selanjutnya.

Berdasarkan dua observasi di atas, maksimal banyak pengecekan yang terjadi adalah (n-m+1)m. Oleh karena itu, algoritma bruteforce ini memiliki kompleksitas waktu maksimal O((n-m+1)m). Apabila diasumsikan nilai m jauh lebih kecil dari n, atau m << n, maka akibatnya n-m+1 ~ n, sehingga dapat disimpulkan algoritma brute force ini memiliki kompleksitas waktu yang sama dengan O(nm).

2) Algoritma Knuth-Morris-Pratt (KMP)

Algoritma KMP merupakan sebuah algoritma pencocokan string yang juga melakukan iterasi pencocokan dengan substring dari string teks T. Namun apabila terjadi *mismatch*, string pattern P tidak digeser hanya sebanyak satu ke kanan, namun bergantung pada indeks P pada saat itu dan sebuah kuantitas tertentu. Kuantitas ini ditentukan dengan melakukan prekalkulasi sebuah fungsi yang bernama Border Function. Hasil dari fungsi ini disimpan dalam sebuah larik, dan digunakan ketika melakukan iterasi pencocokan string.

Border Function merupakan sebuah fungsi yang spesifik terhadap suatu string S, dan parameternya sebenarnya ada dua, yakni sebuah string S dan sebuah bilangan bulat k yang berperan sebagai indeks. Namun, pada algoritma KMP, pencarian nilai dari fungsi Border Function ini hanya dikalkulasi untuk string pattern P saja, sehingga nilai default dari parameter S nya adalah P, dan dapat diasumsikan mulai sekarang fungsi ini hanya memiliki satu parameter berupa bilangan bulat k, dan parameter satunya lagi adalah string P.

Notasikan fungsi ini sebagai b(). Nilai dari b(i) merupakan panjang prefix terbesar dari P[0..i] yang juga merupakan suffix dari P[1..i]. Perhatikan bahwa, apabila terjadi sebuah mismatch ketika sedang membandingkan P[j] dengan T[k] untuk suatu j dan k merupakan indeks, maka dapat dilakukan shift sehingga sekarang indeks pada P yang akan dicocokkan dengan T[k] merupakan P[b(j-1)], atau ekuivalen dengan melakukan *assignment* j menjadi b(j-1).

Sebagai contoh, untuk string P = “abaaba”, nilai dari border functionnya untuk setiap indeks dari P ditunjukkan pada tabel berikut ini.

TABEL 2. TABEL NILAI BORDER FUNCTION

j	**0**	**1**	**2**	**3**	**4**	**5**
P[j]	a	b	a	a	b	a
k	-	0	1	2	3	4
b(k)	-	0	0	1	1	2

Dari tabel tersebut dapat dilihat bahwa nilai b(4) = 2, karena string terpanjang yang menjadi prefix dari P[0..4] dan suffix dari P[1..4] adalah string “ab” dan string ini memiliki panjang 2.

Untuk menganalisis kompleksitasnya, anggap terdapat sebuah *pointer* atau penunjuk semu pada T yang awalnya berada pada T[0]. Perhatikan bahwa apabila tidak terjadi *mismatch*, maka *pointer* tersebut akan bergerak ke kanan, begitupula dengan P. Sedangkan, jika terjadi *mismatch*, maka *pointer*-nya akan diam di tempat, dan S akan bergerak ke kanan sesuai dengan nilai dari Border Function-nya. Dari sini disimpulkan bahwa *pointer* tidak pernah kembali ke kiri (berbeda dengan Brute Force) dan tiap langkah pasti ada setidaknya satu di antara P dan *pointer* tersebut yang bergerak ke kanan. Karena *pointer* paling banyak melangkah sebanyak n buah (banyak karakter pada T) dan sebanyak m maka kompleksitas waktu maksimal dari algoritma ini adalah O(m+n).

Sifat *pointer* yang tidak pernah bergerak ke kiri ini menjadi salah satu keuntungan KMP, dan dapat digunakan untuk memproses teks yang cukup besar ukurannya. Akan tetapi, jika banyak kemungkinan karakter pada string bertambah, maka

banyak kemungkinan *mismatch* juga bertambah, sehingga menurunkan kinerja algoritma ini.

C. Audio dan Frekuensi

Pada teori musik, setiap musik tersusun atas komponen dasar yang disebut dengan nada. Setiap nada merupakan sebuah gelombang suara sehingga memiliki nilai frekuensi, dan nada yang terdengar berbeda berarti memiliki nilai frekuensi yang berbeda juga.

Terdapat nada-nada yang terdengar sama, tetapi frekuensi mereka berbeda, atau tingkat kelengkingannya berbeda. Ini berarti, kedua nada tersebut terletak pada oktaf yang berbeda. Oktaf merupakan sebuah interval nada yang memisahkan dua buah nada yang nilai frekuensi yang satu merupakan dua kali lipat nilai frekuensi yang lainnya. Satu oktaf sendiri terdiri atas dua belas nada (tangga nada diatonik) yang telah terkuantisasi. Jarak antar tiap nada ini disebut dengan *semitone*. Karena setiap naik beberapa tingkat nada tertentu akan menghasilkan rasio frekuensi yang konstan, maka dari sini dapat disimpulkan bahwa setiap naik satu buah *semitone*, nilai frekuensi akan dikalikan dengan $2^{1/12}$

Dalam satu oktaf, daftar kedua belas nada pada oktaf tersebut disimbolkan sebagai C, C#, D, D#, E, F, F#, G, G#, A, A#, B. Setiap nada yang berurutan dipisahkan oleh sebuah interval *semitone*. Untuk membedakan sebuah nada pada satu oktaf dengan oktaf yang lainnya, biasanya ditambahkan sebuah angka setelah notasi nada tersebut untuk menandakan letak oktaf. Sebagai contoh, nada B_1 jika naik secara *semitone* akan menghasilkan nada C_2, yakni nada yang terdengar sama dengan C_1, tetapi terletak satu oktaf lebih tinggi.

Salah satu nada yang frekuensinya menjadi nilai standar adalah nada A_4, yakni dengan frekuensi 440 Hz. Nada-nada lainnya dapat dihitung frekuensinya dengan mengalikan frekuensi A_4 tersebut dengan perpangkatan $2^{1/12}$ yang sesuai. Sebagai contoh, berikut ini merupakan tabel frekuensi untuk nada-nada pada oktaf ke-4.

TABEL 3. NILAI FREKUENSI NADA PADA OKTAF 4

Note	*Frekuensi (Hz)*
C_4	261.63
$C^{\#}_4$	277.18
D_4	293.66
$D^{\#}_4$	311.13
E_4	329.63
F_4	349.23
$F^{\#}_4$	369.99
G_4	392.00
$G^{\#}_4$	415.30
A_4	440.00
$A^{\#}_4$	466.16
B_4	493.88

III. ALGORITMA PENCOCOKAN STRING PADA PENCARIAN L MUSIK

A. Definisi dan Terminologi

1) String Representasi Audio

Dalam algoritma ini, audio akan direpresentasikan sebagai sebuah string yang memiliki karakter-karakter berupa bilangan real. Sebuah karakter merepresentasikan frekuensi audio tersebut dalam satu satuan waktu, dalam satuan Hz. Definisikan sebuah audio X yang direpresentasikan dengan string A. Indeks ke-i dari A akan merepresentasikan nilai frekuensi pada X dalam waktu ke-i satuan.

2) Kemunculan Pattern Audio pada Audio Lainnya

Selanjutnya akan ditentukan sifat apa yang harus dipenuhi oleh string-string yang merepresentasikan audio-audio tersebut untuk menghasilkan kecocokan. Notasikan T dan P sebagai string-string representasi musik yang tempat pencarian dan musik pola yang ingin dicari, dengan panjang n dan m berturut-turut. Salah satu yang paling trivial tentunya adalah apabila P muncul secara eksak sebagai substring dari T. Namun, salah satu permasalahan yang cukup sering muncul adalah banyak rangkaian melodi-melodi yang sama namun dimainkan pada tangga nada yang berbeda. Oleh karena itu, walaupun rangkaian melodi yang direpresentasikan oleh P muncul pada audio yang direpresentasikan T, string P tidak akan pasti muncul secara eksak pada T.

Salah satu cara yang dapat dilakukan adalah mengubah P ke tangga nada yang lain agar dapat ditemukan pada T. Muncul permasalahan baru, yakni tidak diketahui tangga nada mana yang harus dicari. Karena ada banyak kemungkinan tangga nada, mencoba satu persatu tangga nada menjadi metode yang tidak efisien.

Misalkan larik yang merepresentasikan P adalah $[P_0, P_1, P_2, \ldots, P_{m-1}]$ dan larik yang merepresentasikan T adalah $[T_0, T_1, T_2, \ldots, T_{n-1}]$. Misalkan audio yang direpresentasikan oleh P muncul pada audio yang direpresentasikan oleh T, tetapi kemunculannya ini berbeda tangga nada, yakni setelah frekuensinya dikalikan dengan suatu bilangan real α yang tidak diketahui. Akibatnya, terdapat suatu substring dari T yang memiliki panjang m, yakni $[T_k, T_{k+1}, T_{k+2}, \ldots, T_{k+m-1}]$ dan cocok dengan string hasil perkalian P dengan α. Hal ini dapat dituliskan sebagai berikut:

$$[\alpha P_0, \alpha P_1, \alpha P_2, \ldots, \alpha P_{m-1}] = [T_k, T_{k+1}, T_{k+2}, \ldots, T_{k+m-1}].$$

Sehingga akibatnya

$$\alpha P_i = T_{k+i} \text{ untuk setiap i pada } \{0, 1, 2, \ldots, m-1\}$$

$$\leftrightarrow P_i/T_{k+i} = \alpha \text{ untuk setiap i pada } \{0, 1, 2, \ldots, m-1\},$$

yang mengakibatkan nilai dari P_i/T_{k+i} konstan untuk setiap i pada {0, 1, 2, …, m-1}. Sehingga, untuk setiap dua bilangan i pada {0, 1, 2, …, m-2} berlaku:

$$P_i/T_{k+i} = P_{i+1}/T_{k+i+1}$$

$$\leftrightarrow P_{i+1}/P_i = T_{k+i+1}/T_{k+i} \quad (*)$$

Apabila didefinisikan sebuah string P' dan T' dengan panjang masing-masing m-1 dan n-1 dengan karakternya masing-masing didefinisikan sebagai $P_i' = P_{i+1}/P_i$ untuk setiap i pada {0, 1, 2, …, m-2} dan $T_i' = T_{i+1}/T_i$ untuk setiap i pada {0, 1, 2, …, n-2}, maka persoalan pencocokan audio ini akan dapat disederhanakan menjadi persoalan pencocokan string dengan pattern P' dan T'. Hal ini disebabkan bahwa (*) ekuivalen dengan $P_i' = T_{k+i}'$ untuk setiap i pada {0, 1, 2, …, m-2}.

Terakhir, perhatikan bahwa string P' dan T' masing-masing memiliki hubungan dengan P dan T sebagai berikut: (1) Buat string baru P1 yang karakter-karakternya adalah logaritma terhadap suatu basis b dari karakter-karakter pada P; (2) Buat string selisih dari P1, yakni P2; (3) Buat string baru P3 yang karakter-karakternya merupakan perpangkatan b terhadap karakter-karakter dari P2. String terakhir yang terbentuk, yakni P3, akan bernilai sama dengan P'.

Sehingga, untuk mengaplikasikan algoritma pencocokan string, lakukan *preprocessing* datanya terlebih dahulu dengan cara membentuk string baru yang karakter-karakternya merupakan hasil pembagian karakter-karakter yang berdekatan pada string awal untuk masing-masing string P dan T.

Maka, dapat disimpulkan bahwa audio yang direpresentasikan oleh string P muncul pada audio yang direpresentasikan pada T jika dan hanya jika string P' muncul pada string T'.

B. Algoritma

1) Pembacaan Musik menjadi Larik Frekuensi

Pembacaan musik dapat dilakukan dengan dua cara, yakni dengan membaca file audio secara langsung dan mengubahnya menjadi struktur data larik yang berisikan frekuensi-frekuensi, serta melakukan *hardcode* dengan cara memasukkan barisan nada pada program.

Untuk pembacaan dari file audio, data dari file audio dibaca dengan menggunakan *library* yang mengimplementasikan fungsi FFT (Fast Fourier Transform), Sedangkan untuk melakukan *hardcode* audio secara manual, dibuat sebuah *map* yang memetakan sebuah nada dengan frekuensinya, lalu barisan nada tersebut akan secara otomatis menjadi larik frekuensi audionya. Map yang dibuat dapat mengikuti tabel 3, yang berisi pasangan beberapa nada beserta frekuensinya, dan dapat diekstensi untuk oktaf-oktaf yang lainnya.

2) Preprocessing Larik

Larik frekuensi yang dihasilkan pada langkah sebelumnya diubah sesuai dengan langkah yang telah disebutkan pada bagian A.2, yakni membagi tiap dua karakter yang berdekatan pada larik frekuensi.

$$\alpha P_i = \chi_i \quad (1)$$

```
function preprocessingLarik(S)
Sf = {}
    for each i in {0,1,2,…, |S|-2}
        Sf = Sf + (Si+1/Si)
    endfor
    return Sf
endfunction
```

3) Algoritma Pattern Matching

Dua larik dari hasil langkah sebelumnya di-*pass* pada algoritma pencocokan string. Algoritma pencocokan string yang manapun dapat dicoba dan dibandingkan performanya.

Untuk melakukan pencocokan karakter, dibutuhkan sebuah fungsi tambahan karena karakter yang digunakan merupakan bilangan real. Masalah presisi dapat muncul ketika melakukan preprocessing, yakni ketika melakukan pembagian dua bilangan real. Oleh karena itu, digunakan fungsi isEqual() berikut untuk membandingkan dua buah bilangan real.

```
function isEqual(x,y)
    return |x-y| < 10^-3
endfunction
```

Konstanta 10^{-3} merupakan konstanta yang dianggap cukup untuk dijadikan sebagai galat karena rasio dari tiap dua frekuensi yang berdekatan adalah $2^{1/12}$, yakni sekitar 1.059, yang berarti setiap rasio frekuensi merupakan perpangkatan dari 1.059.

IV. PEMBAHASAN

Dengan membandingkan beberapa algoritma pencocokan string, dapat dilihat bahwa algoritma brute force berjalan paling lambat seperti yang diharapkan. Selain itu, walaupun karakter dari string yang merepresentasikan audio tersebut merupakan bilangan real, namun banyak bilangan real yang dapat menjadi anggota array tersebut sama banyaknya dengan banyak nada yang mungkin. Karena banyaknya nada yang mungkin pada tiap oktaf ada 12 dan hanya terdapat sedikit buah oktaf, maka banyak karakter total yang mungkin muncul tidaklah terlalu banyak. Salah satu alternatif yang dapat dilakukan adalah melakukan pemetaan terhadap bilangan real yang menyatakan frekuensi ke dalam suatu karakter alfabet, sehingga string yang dicocokkan akan memiliki karakter berupa huruf,, dan akibatnya fungsi equal() tidak perlu digunakan.

Algoritma Boyer-Moore dapat bermasalah ketika digunakan jika karakternya merupakan bilangan real, karena masalah presisi bilangan real. Pemetaan yang disebutkan pada paragraf sebelumnya dapat menjadi solusi dari permasalahan ini.

Salah satu hal penting yang menjadi kelemahan algoritma ini adalah tempo. Asumsi dasar yang digunakan adalah satuan waktu pada audio teks dan pattern bernilai sama, padahal banyak sekali lagu-lagu yang dimainkan tidak hanya pada tangga nada yang berbeda tetapi juga pada tempo yang berbeda. Oleh karena itu diperlukan pembahasan lebih lanjut pada masalah ini.

Selain itu, terdapat sebuah kekurangan lagi terkait lagu yang memiliki banyak suara (polifonik). Untuk mengatasi hal ini, dibutuhkan sebuah algoritma tambahan untuk memproses lapisan suara tersebut dan memisahkannya jadi beberapa suara yang monofonik.

LINK SOURCE CODE PADA GITHUB

https://github.com/donbasta/melody-matcher-pattern-matching

File-file source code dan kasus uji dapat ditemukan pada pranala tersebut. Perubahan pada repository dapat terjadi sewaktu-waktu sehingga menghasilkan ketidaksesuaian dengan laporan ini.

LINK VIDEO PADA YOUTUBE

Untuk penjelasan lebih lanjut, berikut penulis lampirkan pranala berupa video yang dapat diakses melalui link berikut ini.

https://youtu.be/6dPzEs_jsLc

Video tersebut berjudul “Pencocokan Musik Sederhana dengan Algoritma String Matching”.

UCAPAN TERIMA KASIH

Penulis mengucapkan terima kasih kepada Tuhan Yang Maha Esa, kedua orang tua yang telah mendukung penulis, serta seluruh Bapak dan Ibu dosen pengampu mata kuliah IF2211 Strategi Algoritma: Bapak Rinaldi Munir, Ibu Nur Ulfa Maulidevi, dan Ibu Masayu Leylia Khodra yang telah membimbing penulis selama perkuliahan di semester empat ini, baik di kelas maupun jarak jauh. Penulis juga mengucapkan terima kasih kepada teman-teman dan pihak-pihak lain yang telah mendukung selama proses pengerjaan makalah ini.

REFERENSI

[1] https://www.cs.utah.edu/~germain/PPS/Topics/strings.html, diakses pada 4 Mei 2020

[2] https://www.ifpi.org/news/IFPI-releases-music-listening-2019, diakses pada 4 Mei 2020

[3] https://pages.mtu.edu/~suits/notefreqs.html, diakses pada 4 Mei 2020

[4] https://www.toptal.com/algorithms/shazam-it-music-processing-fingerprinting-and-recognition, diakses pada 4 Mei 2020

[5] http://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2017-2018/Pencocokan-String-(2018).pdf, diakses pada 4 Mei 2020

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, 4 Mei 2020

Farras Mohammad Hibban Faddila
13518017