Aplikasi String Matching untuk Menentukan Zaman dari Karya Musik Klasik

Fazel Ginanda - 13521098
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
E-mail (gmail): 13521098@std.stei.itb.ac.id

***Abstrak*—Musik klasik adalah musik yang berkembang di Benua Eropa dari awal abad ke-17 sampai dengan awal abad ke-20. Dalam kurun waktu sekitar 300 tahun tersebut, terdapat tiga zaman penting yang menandai karakteristik umum dari seluruh lagu yang berkembang pada selang waktu yang sama. Ketiga zaman tersebut adalah zaman Barok, zaman Klasik, dan zaman Romantik. Untuk membedakan musik dari setiap zaman tersebut, diperlukan pengalaman bermusik yang cukup. Ini menjadi kendala yang menyulitkan orang awam yang tidak memiliki pengalaman di bidang musik. Oleh karena itu, penulis memanfaatkan aplikasi *string matching* untuk menentukan zaman dari sebuah karya musik klasik berdasarkan karakteristik atau sifat dari karya musik tersebut.**

Kata kunci—musik klasik; zaman; karakteristik; string matching; Knuth-Morris-Pratt; Boyer-Moore

I. PENDAHULUAN

Musik klasik adalah musik yang berkembang di Benua Eropa dari awal abad ke-17 sampai dengan awal abad ke-20. Ciri khas dari musik ini adalah memiliki bentuk yang formal dan terstruktur. Selain itu, musik klasik juga memiliki nuansa yang serius. Perkembangan musik klasik terkait erat dengan dinamika sosial budaya masyarakat Eropa. Pada awalnya, musik yang berkembang di Eropa bertujuan sebagai sarana ibadah. Hal ini terlihat dari keberadaan musik-musik yang dibuat oleh Gereja Kristen di Eropa. Akan tetapi, seiring berjalannya waktu, musik di Eropa berkembang dan tidak lagi hanya berperan sebagai sarana religi. Musik juga berperan sebagai sarana hiburan dan ditampilkan dalam berbagai jenis pertunjukan.

Sejarah musik Eropa dapat dibagi ke dalam dua masa, yaitu masa awal dan masa *common practice*. Pada masa awal, terdapat dua zaman, yaitu zaman Medieval dan zaman Renaissans. Sementara itu, pada masa *common practice*, terdapat tiga zaman, yaitu zaman Barok, zaman Klasik, dan zaman Romantik. Pada masa *common practice* ini terjadi banyak evolusi terhadap musik, seperti sistem tonal, bentuk musik Sonata, dan teori harmoni. Selain evolusi terhadap musik, tiga zaman ini juga ditandai dengan kontibusi komposer-komposer besar yang melahirkan karya musik terkenal dan berpengaruh terhadap sejarah musik dunia. Oleh karena itu, tiga zaman tersebut merupakan zaman-zaman yang paling penting dalam perkembangan musik.

Kemampuan mengidentifikasi zaman dari sebuah karya musik harus dimiliki oleh pemain musik. Alasannya adalah seorang pemain harus menginterpretasikan karya musik dengan tepat dan akurat agar dapat memberikan penampilan yang baik. Kesalahan dalam menginterpretasikan sebuah karya musik dapat mengakibatkan musik tidak dapat dinikmati oleh pendengarnya. Untuk menghindari hal ini, langkah pertama yang harus dilakukan adalah mengidentifikasi zaman dari karya musik yang ingin dimainkan. Pengetahuan tentang zaman dari sebuah karya musik memberikan gambaran awal kepada pemain musik terkait karya yang akan dimainkan. Hal ini dikarenakan setiap zaman memiliki karakteristik yang membedakan karya musik dari zaman tersebut dengan karya musik dari zaman lainnya.

Selain diperlukan oleh pemain musik, kemampuan mengidentifikasi zaman dari sebuah karya musik juga diperlukan oleh pendengar atau penonton dari sebuah acara pertunjukan musik klasik. Hal ini dikarenakan kemampuan tersebut membantu pendengar untuk memahami nilai dan makna dari sebuah karya musik secara lebih mendalam. Akan tetapi, hal ini terasa menyulitkan bagi orang yang belum pernah belajar bermain musik atau belum pernah mempelajari musik klasik. Oleh karena itu, *string matching* dapat dimanfaatkan untuk menentukan zaman dari sebuah karya musik klasik berdasarkan karakteristik musik dari setiap zaman.

II. LANDASAN TEORI

A. *String*

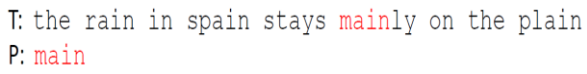
String adalah struktur data yang digunakan untuk menyimpan teks. Implementasi dari struktur data ini bermacam-macam tergantung bahasa pemrograman yang digunakan. Secara umum, *string* dapat dinyatakan sebagai deretan karakter yang bisa diakses setiap karakternya. Akses terhadap setiap karakter dari sebuah *string* dapat dilakukan seperti mengakses elemen larik. Contohnya, pada *string* S yang indeksnya dimulai dari 0, karakter ketiga dari S dinyatakan dengan notasi S[2].

Dalam *string*, dikenal dua istilah yaitu prefiks dan sufiks. Dalam ilmu bahasa, prefiks menyatakan awalan dari suatu kata. Sementara itu, sufiks adalah akhiran dari suatu kata. Konsep ini juga digunakan dalam *string*. Prefiks adalah satu atau lebih karakter berurutan yang dimulai dari karakter pertama dari

suatu *string*. Sementara itu, sufiks adalah satu atau lebih karakter berurutan yang diakhiri dengan karakter terakhir dari suatu string. Misalkan terdapat sebuah *string* "andrew." Semua prefiks dari *string* tersebut adalah "a", "an", "and", "andr", "andre", dan "andrew". Sementara itu, semua sufiks dari *string* tersebut adalah "w", "ew", "rew", "drew", "ndrew", dan "andrew". Sama halnya dengan akses terhadap karakter, akses terhadap prefiks dan sufiks juga dinyatakan dengan notasi yang serupa seperti larik. Notasi dari prefiks *string* S adalah S[0..i]. Notasi ini menyatakan prefiks dari *string* S yang terdiri dari karakter pada indeks 0 sampai dengan karakter pada indeks i. Sementara itu, notasi sufiks dari *string* S dinyatakan dengan S[k..m-1]. Nilai k adalah indeks dari karakter pertama sufiks dan m merupakan panjang dari string.

B. *String Matching*

Algoritma *string matching* atau pencocokan *string* adalah algoritma yang bertujuan mencari kemunculan suatu pola di dalam sebuah teks. Diasumsikan panjang dari pola jauh lebih kecil daripada panjang dari teks. Algoritma ini mengembalikan posisi atau indeks dari kemunculan pertama pola di dalam teks.



Gambar 1. Contoh *string matching*
(Sumber: https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Pencocokan-string-2021.pdf diakses pada 22 Mei 2023)

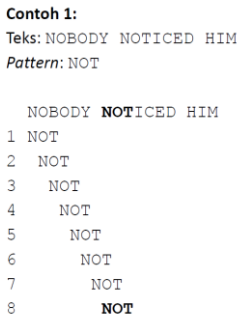
Gambar 1 mengilustrasikan konsep dari pencocokan *string*. Pola yang ingin dicari adalah "main". Sementara itu, teks yang diberikan adalah "the rain in spain stays mainly on the plain". Algoritma pencocokan *string* akan mengembalikan nilai 24 karena huruf m yang merupakan karakter pertama dari "main" terdapat pada indeks 24 dari *string* teks.

String matching digunakan dalam informatika dan berbagai bidang lainnya. Contoh yang paling umum adalah pencarian kata di dalam editor teks. Fitur yang terdapat di dalam setiap editor teks ini memanfaatkan algoritma *string matching*. Selain itu, contoh penerapan lainnya adalah pada *web search engine* dan analisis citra. *String matching* juga diterapkan di bidang bioinformatika, yaitu dalam pencarian rantai asam amino pada suatu rantai DNA.

C. *Algoritma Brute Force*

Algoritma *Brute-Force* adalah algoritma dengan pendekatan yang lempang untuk menyelesaikan suatu persoalan. Algoritma ini sangat sederhana, langsung, dan jelas. Oleh karena itu, algoritma ini disebut juga algoritma naif. Dalam persoalan pencocokan *string*, algoritma *brute force* dapat diterapkan dengan cara berikut.

1. Sejajarkan pola dengan bagian awal dari teks.
2. Periksa setiap karakter dari pola dimulai dari karakter pertama. Bandingkan karakter tersebut dengan karakter pada teks yang sejajar.
3. Jika kedua karakter tersebut sama, maka pemeriksaan dan pembandingan dilanjutkan dengan karakter berikutnya.
4. Jika kedua karakter tersebut berbeda, maka pola digeser sebanyak satu indeks ke kanan dan pemeriksaan dimulai kembali dari karakter pertama pada pola.
5. Apabila seluruh karakter yang dibandingkan sama, maka pencocokan berhasil dan proses pencocokan dihentikan.
6. Apabila pada proses pembandingan terakhir masih ditemukan ketidakcocokan, maka proses pencocokan dinyatakan gagal dan pola tidak ditemukan pada teks.



Gambar 2. Contoh proses pencocokan *string* dengan algoritma *brute force*
(Sumber: https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Pencocokan-string-2021.pdf diakses pada 22 Mei 2023)

Gambar 2 menggambarkan penerapan algortima *brute force* dalam persoalan pencocokan string. Pada pencocokan pertama, ditemukan ketidakcocokan pada karakter kedua dari pola. Oleh karena itu, pola digeser sejauh satu karakter ke kanan. Pada proses pencocokan kedua hingga ketujuh, ketidakcocokan selalu ditemukan pada karakter pertama. Akhirnya pada pencocokan kedelapan, seluruh karakter pada pola sama dengan karakter pada teks di posisi yang sejajar. Secara keseluruhan, dilakukan pembandingan karakter sebanyak sebelas kali pada contoh tersebut.

Proses pencocokan yang terjadi pada Gambar 2 tersebut termasuk ke dalam kasus rata-rata. Hal ini ditandai dengan sedikitnya jumlah kecocokan pada karakter pertama pola dan ketidakcocokan pada salah satu karakter berikutnya. Kompleksitas pada kasus rata-rata ini adalah O(m+n). Sejauh ini, algoritma *brute force* terlihat cepat dan efektif. Akan tetapi, pada kasus terburuk, proses pembandingan menjadi sangat banyak dan menyebabkan algoritma menjadi lambat seperti pada contoh berikut.

- Teks: aaaaaaaaaaaaaaaaaaaaaaaaah
- Pola: aaah

Pada contoh tersebut, selalu ditemukan kecocokan pada tiga karakter pertama dari pola dengan karakter teks yang sejajar. Ketika ditemukan ketidakcocokan pada karakter terakhir dari pola, pergeseran pola hanya dilakukan sekali. Hal itu terus-menerus terjadi hingga pola ditemukan pada bagian akhir dari teks. Akibatnya, proses pembandingan yang dilakukan menjadi sangat banyak. Pada contoh tersebut, dilakukan pembandingan karakter sebanyak 96 kali. Secara matematis, pada kasus terburuk, pembandingan dilakukan sebanyak m(n-m+1) kali. Hal ini berarti kompleksitas algoritma pada kasus terburuk

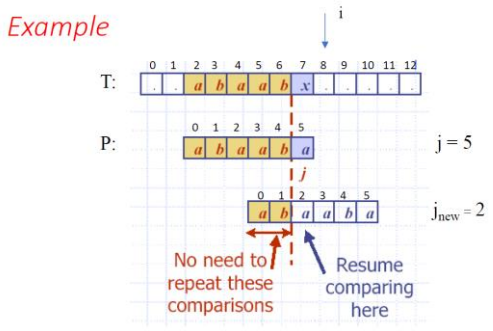
adalah O(mn). Sementara itu, contoh kasus terbaik dalam penerapan algoritma *brute force* adalah sebagai berikut.

- Teks: String ini berakhir dengan zzz
- Pola: zzz

Pada contoh tersebut, karakter pertama dari pola tidak pernah sama dengan karakter teks yang sejajar sebelum mencapai proses pembandingan terakhir. Secara matematis, pola digeser sebanyak (n-m+1) kali. Pada kasus ini, pembandingan dilakukan paling banyak sebanyak n kali. Oleh karena itu, kompleksitas algoritma dari kasus terbaik adalah O(n).

D. Algoritma Knuth-Morris-Pratt

Algoritma Knuth-Morris-Pratt (KMP) adalah algoritma pencocokan *string* yang mirip dengan *brute force*. Kemiripan ini ditandai dengan proses penelusuran yang dilakukan dari kiri ke kanan. Perbedaannya terletak pada banyak pergeseran yang tidak tetap. Jika ditemukan ketidakcocokan antara teks dengan pola pada karakter P[j], maka banyak pergeseran yang dilakukan adalah sama dengan panjang prefiks terbesar dari P[0..j-1] yang merupakan sufiks dari P[1..j-1].



Gambar 3. Pergeseran pola pada algoritma KMP
(Sumber: https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Pencocokan-string-2021.pdf diakses pada 22 Mei 2023)

Untuk menerapkan algoritma KMP dibutuhkan evaluasi fungsi pinggiran terlebih dahulu. Fungsi pinggiran KMP adalah fungsi yang memproses pola sebelum pola tersebut dibandingkan dengan teks secara langsung. Artinya evaluasi fungsi pinggiran ini dapat disebut sebagai tahap pendahuluan dari algoritma pencocokan *string*.

Fungsi pinggiran KMP direpresentasikan sebagai larik atau tabel. Misalkan ketidakcocokan ditemukan pada indeks j dari pola. Terdapat nilai k yang merupakan posisi sebelum ketidakcocokan tersebut ditemukan, yaitu sama dengan j-1. Dengan demikian, indeks dari setiap elemen larik menyatakan nilai k. Sementara itu, nilai yang disimpan pada setiap elemen larik menyatakan nilai evaluasi dari fungsi pinggiran yaitu b(k). Nilai b(k) didefinisikan sebagai panjang terbesar dari prefiks P[0..k] yang juga merupakan sufiks dari P[1..k]. Fungsi pinggiran ini juga disebut dengan *failure function* atau *fail*.

(k = j-1)

j	0	1	2	3	4	5
P[j]	*a*	*b*	*a*	*a*	*b*	*a*

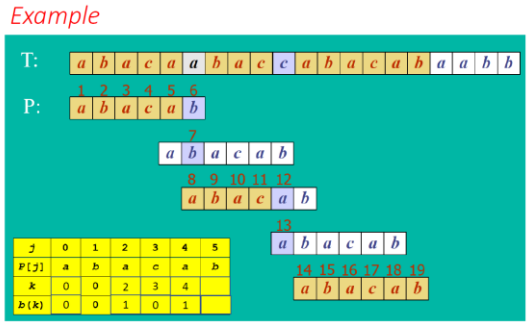
k	0	1	2	3	4
b(k)	0	0	1	1	2

Gambar 4. Hasil evaluasi fungsi pinggiran dalam bentuk tabel
(Sumber: https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Pencocokan-string-2021.pdf diakses pada 22 Mei 2023)

Gambar 4 menunjukkan hasil evaluasi fungsi pinggiran KMP untuk pola "abaaba" yang disimpan dalam tabel. Fungsi tersebut dapat dimaknai sebagai berikut. Jika ditemukan ketidakcocokan pada indeks 5, maka akan dicari nilai b(4). Nilai b(4) adalah panjang prefiks terbesar dari P[0..4], yaitu "abaab" yang juga merupakan sufiks dari P[1..4], yaitu "baab." Substring yang memenuhi definisi tersebut adalah "ab" dengan panjang sebesar 2. Oleh karena itu, b(4) bernilai 2.

Setelah fungsi pinggiran KMP untuk pola yang dicari telah dievaluasi, maka algoritma utama KMP sudah dapat digunakan. Berikut adalah langkah-langkah pencocokan *string* dengan menggunakan algoritma KMP.

1. Sejajarkan pola dengan bagian awal teks.
2. Bandingkan karakter pada pola dengan karaker pada teks satu per satu.
3. Apabila ditemukan ketidakcocokan pada karakter P[j], maka geser pola ke kanan sejauh j-b(k) karakter. Setelah pergeseran, pencocokan berikutnya dimulai dari karakter pada indeks b(k).
4. Apabila seluruh karakter pada pola telah diperiksa dan cocok dengan karakter pada teks di posisi yang bersesuaian, maka pola ditemukan dan pencocokan berhasil.
5. Apabila pola telah berada pada posisi akhir teks dan tidak ditemukan kecocokan pola dengan *substring* dari teks, maka pola dinyatakan tidak ditemukan pada teks tersebut.



Gambar 5. Proses pencocokan *string* dengan algoritma KMP
(Sumber: https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Pencocokan-string-2021.pdf diakses pada 22 Mei 2023)

Algoritma KMP dapat dipandang sebagai algoritma yang terdiri dari dua tahap. Tahap pertama adalah penghitungan nilai fungsi pinggiran dari pola dan menyimpannya ke dalam tabel.

Tahap kedua adalah melakukan pencocokan pola terhadap teks. Pada tahap penghitungan fungsi pinggiran, kompleksitas algoritmanya adalah O(m). Sementara itu, pada tahap pencarian pola, kompleksitasnya adalah O(n). Oleh karena itu, kompleksitas seluruhnya adalah O(m+n). Hal ini menunjukkan algoritma KMP jauh lebih cepat daripada *brute force.*

Kelebihan dari algoritma KMP adalah tidak memerlukan pencarian mundur pada teks, sehingga algoritma ini memiliki kinerja yang baik untuk memproses file berukuran sangat besar. Akan tetapi, algoritma KMP juga memiliki kelemahan, yaitu kinerja akan menurun seiring bertambahnya ukuran alfabet. Hal ini disebabkan, dengan banyaknya jenis huruf yang muncul dalam suatu teks, maka ketidakcocokan lebih cepat ditemukan. Ketika ketidakcocokan ditemukan lebih awal, maka pergeseran yang dilakukan juga tidak banyak. Akibatnya, kelebihan dari algoritma KMP yang menggunakan fungsi pinggiran untuk menentukan pergeseran tidak dapat dimanfaatkan dengan optimal.

E. Algoritma Boyer-Moore

Algoritma Boyer-Moore adalah algoritma pencocokan *string* yang berdasar pada dua teknik, yaitu teknik *looking-glass* dan teknik *character-jump*. Perbedaan utama algoritma Boyer-Moore dengan *brute force* dan KMP adalah urutan pencocokan. Pada dua algoritma sebelumnya, pembandingan karakter selalu dimulai dari karakter pertama dari pola. Sementara itu, pada algoritma Boyer-Moore, proses pencocokan dimulai pada karakter terakhir dari pola. Selain itu, ciri khas dari algoritma ini adalah mempertimbangkan karakter pada teks yang menyebabkan ketidakcocokan pada proses pembandingan karakter.

1. Teknik *looking-glass*

   Teknik *looking-glass* adalah teknik pencarian secara mundur pada pola yang dimulai dari karakter terakhir.

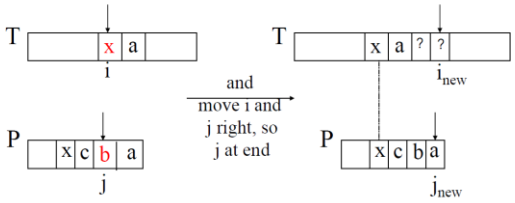
2. Teknik *character-jump*

   Teknik *character-jump* adalah teknik yang digunakan untuk menentukan langkah yang diambil ketika ditemukan ketidakcocokan karakter.

Untuk mendeskripsikan langkah-langkah dalam algoritma Boyer-Moore, diperlukan untuk menjelaskan ketidakcocokan secara lebih rinci. Misalkan, ketidakcocokan terjadi saat karakter pola yang dicocokkan berada di indeks j (P[j]) dan karakter teks yang sejajar berada di indeks i (T[i]). Dalam hal ini, dapat dinyatakan bahwa P[j] ≠ T[i]. Misalkan karakter T[i] adalah x, maka kasus yang mungkin dari ketidakcocokan tersebut adalah sebagai berikut.

1. Kasus 1

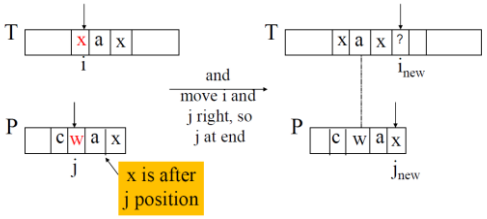
   Kasus 1 terjadi ketika pola mengandung huruf x dan kemunculan terakhir dari huruf x tersebut adalah di sebelah kiri dari karaker pola yang dicocokkan saat ini. Pada kasus ini, pola digeser sedemikian sehingga huruf x terakhir pada pola sejajar dengan huruf x pada karakter teks yang sedang dicocokkan.



Gambar 6. Kasus 1 pada teknik *character-jump*
(Sumber: https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Pencocokan-string-2021.pdf diakses pada 22 Mei 2023)

2. Kasus 2

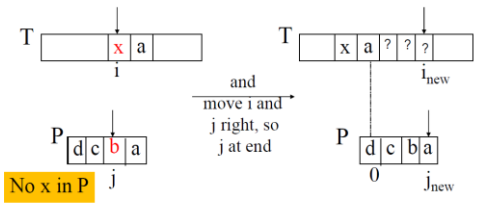
   Kasus 2 terjadi ketika pola mengandung huruf x dan kemunculan terakhir dari huruf x tersebut adalah di sebelah kanan dari karaker pola yang dicocokkan saat ini. Pada kasus ini, pola digeser ke kanan sejauh satu karakter.



Gambar 7. Kasus 2 pada teknik *character-jump*
(Sumber: https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Pencocokan-string-2021.pdf diakses pada 22 Mei 2023)

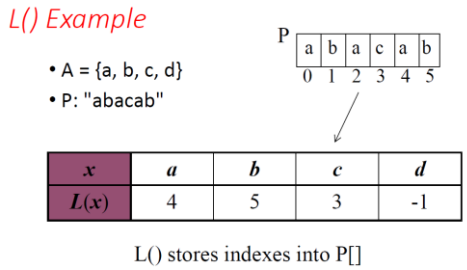
3. Kasus 3

   Kasus 3 merupakan komplemen dari kasus 1 dan 2, yaitu pola tidak mengandung huruf x. Pada kasus ini, pola digeser sedemikian sehinggan P[0] sejajar dengan T[i+1].



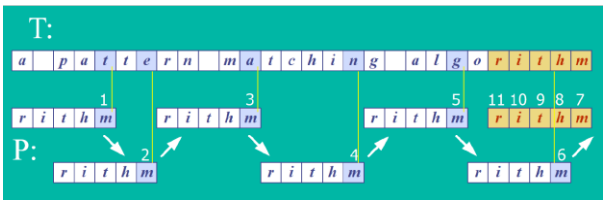
Gambar 8. Kasus 3 pada teknik *character-jump*
(Sumber: https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Pencocokan-string-2021.pdf diakses pada 22 Mei 2023)

Sama seperti algoritma KMP, algoritma Boyer-Moore juga memproses pola terlebih dahulu sebelum melakukan pencocokan dengan teks. Pemrosesan yang dilakukan adalah penghitungan *last occurrence function.* Fungsi ini adalah fungsi yang memetakan semua huruf di alfabet A ke bilangan bulat. Fungsi ini dilambangkan dengan L(). L(x) didefinisikan sebagai indeks kemunculan terakhir huruf x pada pola. Apabila huruf x tidak ada pada pola, maka L(x) bernilai -1. Sama halnya dengan fungsi pinggiran, *last occurrence function* juga direpresentasikan dengan larik atau tabel.



Gambar 9. Hasil evaluasi *last occurrence function* untuk pola "abacb"
(Sumber: https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Pencocokan-string-2021.pdf diakses pada 22 Mei 2023)

Kinerja algoritma Boyer-Moore berada pada tingkat yang terbaik jika alfabet A berukuran besar. Kasus ini dapat dijumpai pada pencarian teks berbahasa Inggris. Akan tetapi, kinerja algoritma ini menjadi buruk jika alfabet A berukuran kecil, seperti pencocokan bilangan biner. Pada kasus terburuk, kompleksitas algoritma Boyer-Moore adalah O(nm + A).



Gambar 10. Proses pencocokan *string* dengan algoritma Boyer-Moore
(Sumber: https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Pencocokan-string-2021.pdf diakses pada 22 Mei 2023)

Gambar 10 mengilustrasikan proses pencocokan *string* menggunakan algoritma Boyer-Moore. Pola yang dicari adalah "rithm". Pada contoh tersebut terlihat bahwa pada lima pembandingan pertama, selalu ditemukan ketidakcocokan pada karakter pertama. Pada pembandingan pertama, kasus *character-jump* yang terjadi adalah kasus 1 karena huruf t terdapat pada pola. Sementara itu, pada pembandingan kedua hingga kelima, kasus *character-jump* yang terjadi adalah kasus 3 karena karakter teks yang dicocokkan tidak ada pada pola. Secara keseluruhan, proses pembandingan karakter dilakukan sebanyak sebelas kali. Pada contoh ini terlihat bahwa algoritma Boyer-Moore memiliki kinerja yang baik karena setiap pergeseran dilakukan relatif jauh, sehingga dapat mengurangi jumlah pembandingan karakter.

F. Periodisasi Musik Barat

Perkembangan musik di dunia Barat, khususnya Benua Eropa telah berlangsung dalam waktu yang sangat lama. Pada masa kekaisaran Romawi dan Yunani Kuno, masyarakat Eropa telah mengenal musik sebagai bagian dari peradaban mereka. Musik terus berkembang pada masa awal hingga masa *common practice.* Masa *common practice* inilah masa yang dianggap paling penting dalam perkembangan musik Barat, khususnya musik klasik karena banyaknya evolusi dalam musik, mahakarya yang dihasilkan, dan kontribusi dari berbagai komposer-komposer besar. Pada masa *common practice* ini terdapat tiga zaman, yaitu zaman Barok, zaman Klasik, dan zaman Romantik.

1. Zaman Barok

   Zaman Barok berada dalam rentang tahun 1600 sampai dengan 1750. Musik yang berkembang pada zaman ini disebut musik barok. Kata "Barok" berasal dari bahasa Portugis, yaitu *borocco* yang berarti Mutiara berbentuk tak wajar. Istilah ini muncul karena banyak kritikus musik pada abad ke-19 yang berpendapat bahwa musik barok terlalu berlebihan dan terlalu banyak ornamen. Sebuah filosofi penting pada zaman barok adalah kepercayaan bahwa musik dapat menjadi alat komunikasi yang sangat kuat. Berdasar pada filosofi ini, para komposer menciptakan karya-karya yang mampu membangkitkan emosi pendengarnya. Terdapat beberapa bentuk musik yang dikaitkan erat dengan zaman Barok, yaitu opera, oratorio, cantata, sonata, concerto, dan suite.

2. Zaman Klasik

   Zaman Klasik berada dalam rentang tahun 1750 sampai dengan 1820. Musik yang berkembang pada zaman ini disebut dengan musik klasik. Karakter utama dari musik yang berkembang pada zaman ini adalah memiliki tekstur yang terang dan jelas, serta menekankan kesederhanaan dan elegan. Selain itu, pada zaman klasik ini juga terjadi pergantian instrument harpsichord dan organ menjadi piano. Tema dari musik yang berkembang di zaman ini didasari oleh melodi dan *rhythm* yang kontras. Secara khusus, karakteristik penting dari sebuah musik didukung dengan penggunaan dinamik. Bentuk musik yang berkembang di zaman ini adalah sonata, trio, string quartet, quintet, symphony, concerto, serenades, dan divertimentos. Di antara bentuk-bentuk musik tersebut, bentuk yang mengalami perkembangan paling signifikan adalah sonata. Sonata digunakan untuk membentuk komposisi musik yang lebih besar, seperti symphony. Selain itu, sonata juga digunakan untuk membentuk musik tunggal, seperti overture.

3. Zaman Romantik

   Zaman Romantik berada dalam rentang tahun 1820 sampai dengan 1910. Musik yang berkembang di zaman ini disebut dengan musik romantik. Musik yang berkembang di zaman ini sangat dipengaruhi oleh terjadinya Revolusi Perancis. Revolusi ini membawa perubahan besar dalam kehidupan sosial masyarakat Eropa pada masa itu. Selain itu, terjadinya Revolusi Industri juga memberikan pengaruh terhadap karya-karya musik pada zaman ini. Revolusi industri menyebabkan perpindahan masyarakat ke kota-kota dan meninggalkan budaya pertanian. Selain itu, revolusi industri juga menjadikan masyarakat semakin jauh dari alam. Rangkaian peristiwa yang terjadi ini mempengaruhi komposer dalam mencipatakan karya musik. Pengaruh itu dapat dilihat dari karya musik yang mengandung kritik terhadap kehidupan politik dan juga ajakan kepedulian terhadap lingkungan. Oleh karena kuatnya emosi dan ekspresi yang diungkapkan dalam karya-karya musik, zaman ini disebut dengan zaman Romantik.

III. IMPLEMENTASI ALGORITMA

Setiap zaman memiliki karakteristik yang membedakan karya musik yang berkembang di zaman itu dengan karya musik dari zaman lainnya. Dengan kata lain, zaman dari sebuah karya musik dapat ditentukan berdasarkan karakteristik dari lagu tersebut. Dalam hal ini, pendengar musik dapat menganalisis sebuah lagu dan menganalisis karakteristiknya untuk menentukan zaman yang tepat untuk karya musik tersebut. Tahapan yang terakhir yaitu menganalisis karakteristik lagu dan menentukan zaman yang sesuai dapat dilakukan dengan memanfaatkan algoritma *string matching*.

Untuk merancang algoritma yang dapat menyelesaikan persoalan ini dengan memanfaatkan *string matching*, diperlukan untuk mendeskripsikan domain masalah dengan lebih rinci. Karakteristik dari sebuah karya musik dapat bermacam-macam yang terdiri dari unsur-unsur musik yang dapat diamati dari karya musik tersebut. Karakteristik karya musik inilah yang menjadi masukan dari algoritma. Sementara itu, karakteristik suatu zaman dapat disimpan dalam sebuah tabel atau larik. Setiap karakteristik ini disimpan dengan struktur data *string*. Hal lain yang perlu diperhatikan adalah sebuah zaman dengan zaman lainnya terhubung secara berkelanjutan. Akibatnya, sebuah karya musik dapat memiliki dua karakteristik atau lebih yang berasal dari dua zaman yang berbeda. Oleh karena itu, zaman dari sebuah karya musik ditentukan berdasarkan kesamaan paling banyak antara karakteristik musik tersebut dengan karakteristik suatu zaman.

Tabel 1. Karakteristik Musik pada Setiap Zaman

Zaman	Karakteristik
Barok	megah, kompleks, rumit, memiliki ornamen, dinamik yang kontras, polifoni, memiliki trill, memiliki mordent, kaya harmoni, integrasi vokal dan instrumen, banyak pergerakan, pergerakan cepat, perubahan tempo dramatis, perubahan volume dramatis, menggunakan basso continuo
Klasik	jelas, terang, sederhana, struktur simetris, tekstur homofonik, perubahan tempo bertahap, artikulasi presisi, menggunakan sonata, integrasi vokal dan instrumen, kontras gelap terang, kontras tonic dominant, semua cadence terdengar sama
Romantik	menekankan ekspresi, ekspansi bentuk musik, banyak harmoni kromatik, kebebasan struktur, komposisi fleksibel, kedalaman emosional, naratif, instrument beragam, progresi harmoni tidak biasa, rubato, tempo fleksibel, rhythm fleksibel, peran konduktor luas, peran dinamik bertambah, leitmotif, menggunakan transformasi tematik, ekspresif, imajinatif

Tabel 1 menunjukkan karakteristik musik dari setiap zaman. Dalam implementasi kode program, tabel ini disimpan dalam struktur data larik. Setiap karakter yang terdefinisi untuk setiap zaman merupakan teks yang akan dicocokkan dengan pola dari masukan. Langkah-langkah penyelesaian persoalan secara keseluruhan dijelaskan sebagai berikut.

1. Berikan masukan sekumpulan karakteristik dari suatu karya musik. Jumlahnya bebas, yaitu boleh satu atau lebih
2. Nyatakan karakteristik musik dari suatu zaman sebagai sebuah string. String ini terdiri dari sejumlah kata atau kelompok kata yang dipisahkan dengan tanda koma. String inilah yang menjadi teks yang akan dicocokkan dengan pola masukan. Untuk setiap zaman, lakukan pencocokan setiap karakteristik dari masukan dengan karakteristik yang berkoresponden dengan zaman tersebut menggunakan algoritma *string matching*.
3. Hitung jumlah kecocokan karakteristik pada setiap zaman
4. Tentukan zaman dengan kecocokan karakteristik paling banyak dengan karakteristik masukan. Zaman dengan kecocokan karakteristik paling banyak inilah yang menjadi keluaran dari algoritma.

Langkah-langkah tersebut dinyatakan dalam notasi pseudocode sebagai berikut.

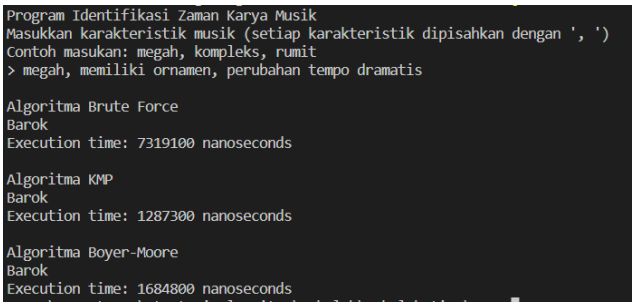
```
Deklarasi
zaman: array of string
karakteristikZaman: array of string
karakteristik: string
countMatch: array of integer
i, j, maxMatch: integer
Algoritma
input(karakteristik)
zaman ← {"Barok", "Klasik", "Romantik"}
karakteristikZaman ← { semua karakteristik musik pada zaman Barok, semua karakteristik musik pada zaman Klasik, semua karakteristik musik pada zaman Romantik }
countMatch ← {0,0,0}
for i ← 0 to len(karakteristik)-1 do
    for j ← 0 to 2 do
        if PatternMatch(karakteristikZaman[j], karakteristik[i]) ≠ -1 then
            countMatch[j] ← countMatch[j] + 1
        endif
    endfor
endfor
```

```
maxMatch ← max(countMatch)
for i ← 0 to to 2 do
    if countMatch[i] = maxMatch then
        output(zaman[i])
    endif
endfor
```

IV. Pengujian dan Analisis Algoritma

Rancangan algoritma yang telah dijelaskan pada bagian sebelumnya diimplementasikan menggunakan bahasa Java. Terdapat tiga fungsi utama yang masing-masing fungsi tersebut mewakili algoritma yang digunakan untuk melakukan *string matching*. Selain itu, juga terdapat program utama yang memuat algoritma program ini secara keseluruhan. Hasil pengujian program tersebut adalah sebagai berikut.

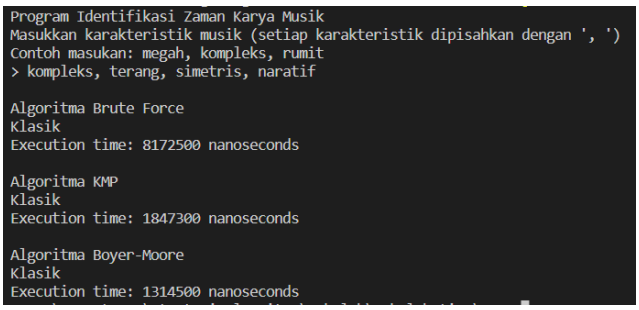
1. Pengujian kasus 1



Gambar 11. Hasil pengujian kasus 1

Gambar 11 menunjukkan hasil pengujian terhadap kasus 1. Pada kasus ini, masukan karakteristik musik dari pengguna hanya cocok dengan karakteristik dari salah satu zaman, yaitu zaman Barok. Hasil pengujian ini sudah memenuhi ekspektasi. Hal ini diperlihatkan oleh ketiga algoritma yang memberikan hasil evaluasi yang benar yaitu zaman Barok. Pada kasus ini, algoritma yang memiliki kinerja terbaik adalah algoritma KMP yang dibuktikan melalui waktu eksekusi yang tercepat.

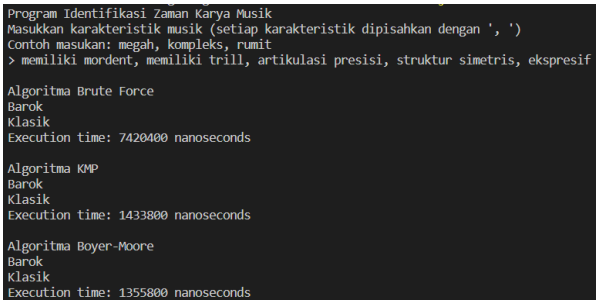
2. Pengujian kasus 2



Gambar 12. Hasil pengujian kasus 2

Gambar 12 menunjukkan hasil pengujian terhadap kasus 2. Pada kasus ini, masukan karakteristik musik dari pengguna cocok dengan karakteristik dari tiga zaman berbeda. Karakteristik terang merupakan bagian dari zaman Barok. Sementara itu, terang dan simetris adalah karakteristik dari zaman Klasik. Terakhir, karakteristik naratif merupakan bagian dari zaman Romantik. Ekspektasi terhadap hasil pengujian ini adalah program menampilkan zaman Klasik karena masukan pengguna paling banyak cocok dengan karakteristik zaman Klasik. Hasil pengujian ini sudah memenuhi ekspektasi. Hal ini diperlihatkan oleh ketiga algoritma yang memberikan hasil evaluasi yang benar yaitu zaman Klasik. Pada kasus ini, algoritma yang memiliki kinerja terbaik adalah algoritma Boyer-Moore yang dibuktikan melalui waktu eksekusi yang tercepat.

3. Pengujian kasus 3



Gambar 13. Hasil pengujian kasus 3

Gambar 13 menunjukkan hasil pengujian terhadap kasus 3. Pada kasus ini, masukan karakteristik musik dari pengguna cocok dengan karakteristik dari tiga zaman berbeda. Karakteristik memiliki morden dan memiliki trill merupakan bagian dari zaman Barok. Sementara itu, artikulasi presisi adalah karakteristik dari zaman Klasik. Terakhir, karakteristik ekspresif merupakan bagian dari zaman Romantik. Ekspektasi terhadap hasil pengujian ini adalah program menampilkan zaman Barok dan zaman Klasik karena jumlah kecocokan karakteristik masukan dengan kedua zaman ini adalah sama, yaitu ditemukan kecocokan sebanyak dua karakteristik. Hasil pengujian ini sudah memenuhi ekspektasi. Hal ini diperlihatkan oleh ketiga algoritma yang memberikan hasil evaluasi yang benar yaitu zaman Barok dan zaman Klasik. Pada kasus ini, algoritma yang memiliki kinerja terbaik adalah algoritma Boyer-Moore yang dibuktikan melalui waktu eksekusi yang tercepat.

V. Kesimpulan

Zaman dari suatu karya musik dapat ditentukan dengan mencocokkan karakteristik dari karya musik tersebut dengan karakteristik musik dari suatu zaman menggunakan algoritma *string matching*. Program dalam bahasa Java yang digunakan untuk mengimplementasikan rancangan algoritma sudah dapat menyelesaikan masalah yang menjadi bahasan makalah ini. Hal ini diperlihatkan pada tahap pengujian program yang sudah memenuhi semua ekspektasi terhadap kasus-kasus yang

diuji. Dalam semua pengujian kasus, algoritma *brute force* memiliki kinerja paling buruk dan memiliki waktu eksekusi paling lama. Sementara itu, kinerja algoritma KMP dan Boyer-Moore relatif tidak berbeda signifikan.

PRANALA REPOSITORI PADA GITHUB

https://github.com/fazelginanda/MakalahStima

UCAPAN TERIMA KASIH

Penulis mengucapkan puji dan syukur terhadap Tuhan Yang Maha Esa karena berkat rahmat dan karunia-Nya, penulis dapat menyelesaikan penulisan makalah yang berjudul "Aplikasi String Matching untuk Menentukan Zaman dari Karya Musik Klasik". Penulis juga mengucapkan terima kasih kepada dosen-dosen pengajar mata kuliah Strategi Algoritma semester 2 tahun ajaran 2022/2023, terutama kepada Ibu Nur Ulfa Maulidevi yang telah memberikan ilmu dan pelajaran yang mendukung penulisan makalah ini.

REFERENSI

[1] Munir, Rinaldi. (2021). "Pencocokan String (String/Pattern Matching)" https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Pencocokan-string-2021.pdf diakses pada 22 Mei 2023

[2] https://www.baroque.org/baroque/whatis diakses pada 22 Mei 2023

[3] https://www.easternct.edu/speichera/understanding-literary-history-all/the-romantic-period.html#:~:text=The%20Romantic%20Period%20began%20roughly,social%20change%20during%20this%20period. diakses pada 22 Mei 2023

[4] https://study.com/academy/lesson/what-is-classical-music-definition-history-composers.html diakses pada 22 Mei 2023

[5] https://www.masterclass.com/articles/classical-era-music-guide diakses pada 22 Mei 2023.

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, 22 Mei 2023

Fazel Ginanda - 13521098