Penerapan Algoritma *String Matching* dalam Pembuatan Rekomendasi Playlist Spotify

Ferindya Aulia Berlianty - 13521161
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
E-mail (gmail): 13521161@std.stei.itb.ac.id

***Abstract*—Tidak dipungkiri, saat ini musik dan lagu menjadi hiburan tersendiri untuk sebagian orang karena dinilai dapat mengungkapkan perasaan melalui lirik yang terdapat dalam sebuah lagu. Terdapat banyak *music streaming platform* yang semakin hari semakin banyak peminatnya seperti *Spotify, YouTube Music, Apple Music, Resso*, dan lainnya. Setiap orang memiliki selera musik yang berbeda dengan beragam genre yang ada. Namun, terkadang sulit untuk menemukan lagu atau musik lainnya dengan genre dan topik yang sama. Dalam salah satu *music streaming platform* yaitu *Spotify*, menyediakan fitur yang membuat penggunanya mendapatkan rekomendasi playlist otomatis sesuai dengan lagu yang didengarkan. Dengan menggunakan algoritma *string matching*, pembuatan rekomendasi playlist dapat dilakukan dengan mudah.**

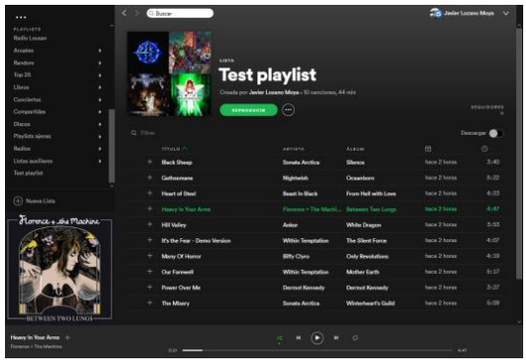
Keywords—string matching; rekomendasi; lirik; lagu

I. PENDAHULUAN

Teknologi semakin berkembang dengan pesat dan memberikan dampak positif pada berbagai sektor kehidupan, salah satunya yaitu dalam bidang musik. Musik merupakan suara yang dihasilkan dari instrumen atau vokal yang disusun dalam sebuah susunan yang teratur dan memiliki irama yang enak didengar. Musik atau lagu merupakan dua hal yang tidak dapat dipisahkan dalam kehidupan manusia saat ini. Music dapat memberikan pengaruh pada emosi dan mood seseorang sehingga sering kali dijadikan sebagai sarana untuk menghilangkan stres atau menghibur diri. Selain itu, musik juga menjadi hal yang penting dalam kehidupan manusia karena dapat dijadikan sebagai pengiring dalam berbagai aktivitas seperti mengerjakan tugas, olahraga, berkendara, atau acara pesta.

Perkembangan teknologi juga memberikan dampak positif pada industri musik. Salah satunya yaitu hadirnya layanan streaming musik, seperti *Spotify, Apple Music*, dan lain sebagainya. Layanan streaming musik ini memudahkan pengguna untuk mendengarkan lagu-lagu favorit mereka kapan saja dan dimana saja. Selain itu, layanan ini juga menyediakan fitur-fitur seperti pembuatan playlist, rekomendasi lagu, dan pencarian lagu berdasarkan genre atau artis.

Pembuatan playlist menjadi hal yang penting dalam layanan streaming musik. Playlist merupakan daftar lagu yang dipilih oleh pengguna dan disusun berdasarkan urutan tertentu untuk didengarkan secara berurutan. Pembuatan playlist dapat dilakukan oleh pengguna secara manual dengan memilih lagu-lagu yang diinginkan. Namun, hal ini terkadang menjadi sulit dan membutuhkan waktu yang lama bagi pengguna yang ingin membuat playlist dengan jumlah lagu yang banyak.



Gambar 1.1 Ilustrasi playlist yang dibuat oleh pengguna

Sumber: https://www.lifewire.com/spotify-tips-and-tricks-4105795

Hadirnya fitur rekomendasi lagu menjadi hal yang penting dalam layanan streaming musik. Dengan fitur ini dapat memberikan rekomendasi lagu kepada pengguna berdasarkan lagu-lagu yang pernah didengarkan atau disukai pengguna sebelumnya.

Pada makalah ini, penulis akan membahas penggunaan algoritma *string matching* dalam pembuatan fitur rekomendasi playlist oleh layanan streaming musik.

II. TEORI DASAR

A. *String Matching*

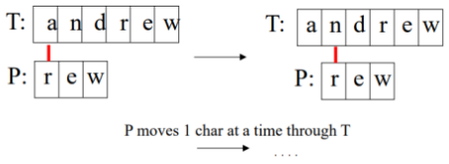
String Matching merupakan algoritma yang digunakan untuk mencari suatu pola yang terdapat pada sebuah teks. *String matching* sering digunakan dalam berbagai aplikasi seperti mesin pencari, sistem pengenalan teks, pengolahan bahasa alami, dan masih banyak lagi. Algoritma ini juga bertujuan untuk mencari kecocokan antara satu teks dengan teks yang lain. Teks pada kasus ini adalah string yang panjangnya n karakter, sedangkan pola atau pattern yang dimaksud adalah string dengan panjang m karakter yang akan dicari di teks dengan asumsi panjang karakter pola lebih kecil jika dibandingkan dengan panjang karakter teks (m <<< n).

Terdapat dua konsep dalam string, yaitu prefix dan suffix. Prefix adalah huruf atau beberapa karakter (substring) yang ada di awal sebuah kata dan dapat merubah arti kata yang asli, sedangkan suffix adalah huruf atau beberapa karakter (substring)

Terdapat beberapa jenis algoritma yang dapat digunakan untuk menerapkan *string matching*, antara lain:

1. Algoritma *Brute Force*

Algoritma *Brute Force* akan memeriksa setiap posisi dalam teks untuk mencari apakah pola atau pattern dimulai pada posisi tersebut. Alur kerja algoritma Brute Force dimulai dengan membandingkan karakter pertama dari pattern dengan karakter pertama yang ada di dalam teks. Jika kedua karakter sama, maka pengecekan akan dilanjutkan ke karakter selanjutnya (satu indeks berikutnya) baik dari teks maupun pattern. Proses ini dilakukan hingga keseluruhan karakter di dalam string pattern telah dicocokkan dengan karakter pada teks. Jika ditemukan karakter yang tidak sama atau tidak cocok, maka pattern akan bergeser sebanyak satu indeks ke kanan dan melakukan pengecekan dimulai dari awal karakter pattern.



Gambar 2.1 Ilustrasi Penggunaan Algoritma Brute Force dalam *String Matching*

Sumber: https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Pencocokan-string-2021.pdf

Dengan panjang atau jumlah karakter dari teks adalah n dan panjang atau jumlah karakter dari pattern adalah m, terdapat beberapa kemungkinan kasus kompleksitas dari algoritma Brute Force.

- *Worst Case*

Worst Case merupakan kemungkinan terburuk dari penggunaan algoritma Brute Force dengan jumlah perbandingan m(n – m + 1) = O(mn). Berikut ini contoh dari worst case:

Teks: aaaaaaaaaaaaaaaaaaaaaaaaab

Pattern: aaab

- *Best Case*

Best Case merupakan kemungkinan terbaik dari penggunaan algoritma Brute Force. Best Case terjadi bila karakter pertama pattern P tidak pernah sama dengan karakter teks T yang dicocokkan. Kompleksitas kasus terbaik dari Best Case adalah O(n) dengan jumlah perbandingan maksimal sebanyak n kali. Berikut ini contoh dari best case:

Teks: String ini berakhir dengan zzz

Pattern: zzz

- *Average Case*

Average case atau kasus rata-rata memiliki kompleksitas waktu O(m + n) yang menandakan bahwa penggunaan algoritma Brute Force dalam kasus rata-rata berjalan dengan sangat cepat. Berikut ini contoh dari average case:

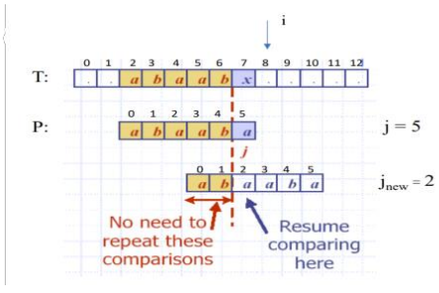
Teks: a string searching example is standard

Pattern: store

Algoritma *Brute Force* akan sangat cepat jika digunakan dalam pencarian string karakter dari teks sangat besar misalnya A…Z, a…z, 1…9, dan sebagainya. Namun, algoritma *Brute Force* akan menjadi sangat lambat jika digunakan dalam teks yang memiliki variasi karakter atau alfabet yang sedikit misalnya dalam teks hanya terdapat karakter 0 atau 1 (seperti binary files, image files, dan sebagainya).

2. Algoritma *Knuth-Morris-Pratt (KMP)*

Algoritma ini dikembangkan secara terpisah oleh James H. Morris Bersama Vaughan R. Pratt pada tahun 1966 dan Donald E-Knuth pada tahun 1967, namun dipublikasikan bersama pada tahun 1977. Algoritma *Knuth-Morris-Pratt* atau yang biasa disingkat dengan KMP merupakan algoritma *string matching* yang melakukan pengecekan dari arah kiri ke kanan string (mulai dari indeks 0 hingga m – 1 pada string dengan panjang m). Algoritma *Knuth-Morris-Pratt* mirip dengan algoritma *Brute Force*. Perbedaan algoritma KMP dengan algoritma Brute Force terletak pada pergeseran yang dilakukan algoritma KMP jauh lebih efisien daripada algoritma *Brute Force*. Algoritma *Brute Force* melakukan pergeseran posisi pattern satu persatu, sedangkan pada algoritma KMP menggunakan prefix terbesar pattern yang juga merupakan suffix dari pattern tersebut untuk dipakai sebagai acuan mengetahui jarak paling jauh untuk melakukan pergeseran pengecekan pattern. Prefix terbesar yaitu P[0…j-1] yang juga merupakan suffix P[1…j-1] dengan j merupakan indeks dari pattern saat terjadi *mismatch* atau ketidakcocokan karakter pattern pada indeks tersebut dengan teks.



Gambar 2.2 Ilustrasi Penggunaan Algoritma KMP dalam *String Matching*

Sumber: https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Pencocokan-string-2021.pdf

Jumlah pergeseran yang dilakukan oleh pattern apabila terdapat ketidakcocokan dihitung dengan border function (failure function). Border function atau yang biasa disimbolkan dengan b(k) didefinisikan sebagai ukuran terbesar prefix dari P[0…k] yang juga merupakan suffix dari P[i…k], dengan k

merupakan posisi karakter pada pola sebelum ketidakcocokan (ketidakcocokan terjadi di P[j], k = j – 1).

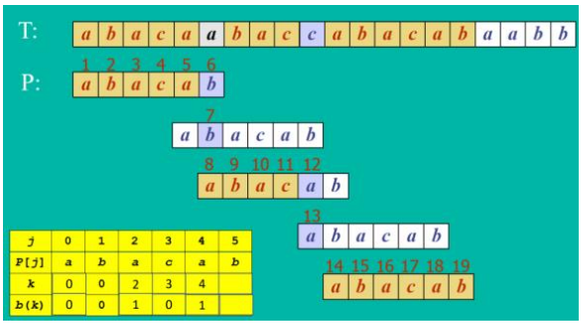
(k = j-1)

j	0	1	2	3	4	5	6	7	8	9
P [j]	a	b	a	b	a	b	a	b	c	a
k	0	1	2	3	4	5	6	7	8	
b[k]	0	0	1	2	3	4	5	6	0	

Gambar 2.3 Ilustrasi Penggunaan Border Function

Sumber:
https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Pencocokan-string-2021.pdf

Algoritma *Knuth-Morris-Pratt* akan menggunakan border function untuk meningkatkan efisiensi pergeseran yang terjadi pada algoritma brute force. Apabila terjadi ketidakcocokan pada P[j] dan T[i], maka P[j] akan diubah menjadi P[b(k)] dimana k adalah j – 1, setelah itu pencarian dan pencocokan pola akan dilanjutkan.



Gambar 2.4 Ilustrasi Penggunaan Algoritma KMP dan *Border Function* dalam *String Matching*

Sumber:
https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Pencocokan-string-2021.pdf

Dengan panjang atau jumlah karakter dari teks adalah n dan panjang atau jumlah karakter dari *pattern* adalah m, maka kompleksitas waktu algoritma KMP adalah sebagai berikut:

- Menghitung fungsi pinggiran: O(m)
- Pencarian string: O(n)
- Kompleksitas waktu algoritma KMP adalah O(m+n), yang mana sangat cepat dibandingkan dengan algoritma Brute Force

Kelebihan dari algoritma KMP yaitu algoritma ini tidak memerlukan *backwards* dalam teks input sehingga hal ini membuat algoritma KMP baik untuk pemrosesan file yang sangat besar yang dibaca dari perangkat eksternal atau melalui sebuah aliran jaringan. Algoritma KMP juga memiliki kekurangan yaitu algorima ini tidak bekerja dengan baik seiring dengan meningkatnya ukuran dari string. Semakin besar ukuran sebuah string, maka akan lebih banyak mismatch atau ketidakcocokan. Ketidakcocokan cenderung terjadi di awal pattern, tetapi algoritma KMP akan lebih cepat ketika mismatch atau ketidakcocokan terjadi nantinya.

3. Algoritma *Boyer-Moore (BM)*

Algoritma *Boyer-Moore* merupakan algoritma yang memiliki pendekatan berbeda dari algoritma *Brute Force* dan juga algoritma *Knuth-Morris-Pratt* (KMP). Algoritma *Boyer-Moore* didasarkan pada dua teknik, yaitu sebagai berikut:

- Teknik *looking-glass*

  Algoritma BM melakukan pencarian pattern P di dalam teks T dimulai dari kanan ke kiri atau bergerak mundur.

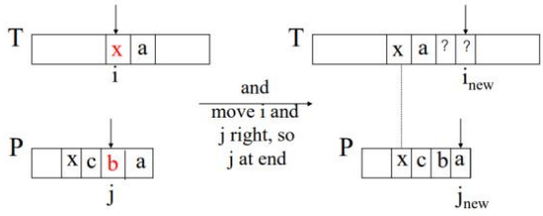
- Teknik *character-jump*

  Teknik ini digunakan berdasarkan kemunculan karakter pada teks T yang tidak cocok ketika dilakukan pengecekan dengan salah satu huruf dalam pattern P kemudian menyesuaikan posisi atau indeks pattern P agar sesuai dengan karakter tersebut. Teknik character-jump berfungsi untuk membuang proses pengecekan yang sebenarnya tidak perlu dilakukan.

Dapat disimpulkan bahwa algoritma *Boyer-Moore* merupakan kebalikan dari algoritma *Brute Force* namun dengan perbaikan dalam hal proses pergeseran pattern. Dengan adanya dua teknik tersebut, algoritma ini cukup efisien dan popular untuk digunakan. Terdapat tiga kasus yang mungkin terjadi dalam proses pencocokan dengan algoritma *Boyer-Moore*, yaitu sebagai berikut:

- Kasus 1

Kasus 1 terjadi ketika *pattern* P mengandung karakter x, maka geser *pattern* P ke kanan sehingga x yang terakhir muncul sejajar dengan teks T[i].



Gambar 2.5 Ilustrasi Penggunaan Teknik Character-Jump untuk kasus 1

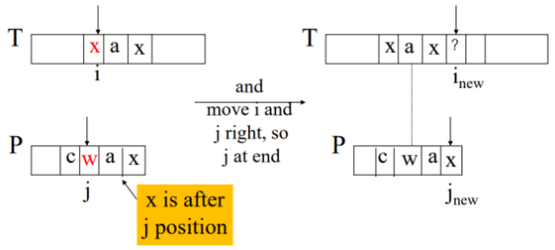
Sumber:
https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Pencocokan-string-2021.pdf

Pada kasus 1 tersebut terjadi ketidakcocokan pada teks indeks ke – i dengan karakternya adalah x (T[i] = x) dimana pada *pattern* P terdapat juga karakter x di sebelah kiri posisi pengecekan saat ini atau posisi saat *mismatch* terjadi, maka akan dilakukan teknik *character-jump* untuk menyamakan karakter x pada *pattern* P dengan karakter x pada teks T. Kemudian akan dilakukan pengecekan ulang dimulai dari belakang.

- Kasus 2

Kasus 2 terjadi ketika *pattern* P mengandung karakter x tetapi tidak dapat dilakukan pergeseran ke kanan untuk

meluruskan karakter x dengan teks T[i], maka geser *pattern* P ke kanan sebanyak 1 karakter.



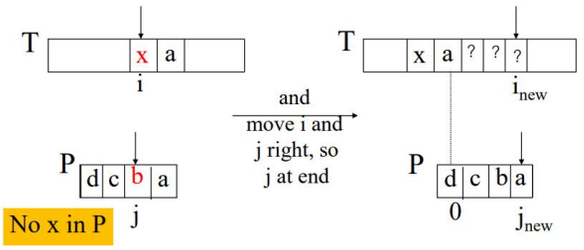
Gambar 2.6 Ilustrasi Penggunaan Teknik Character-Jump untuk Kasus 2

Sumber: https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Pencocokan-string-2021.pdf

Pada kasus 2 tersebut terjadi ketidakcocokan pada teks indeks ke – i dengan karakternya yaitu 'x' (T[i] = x) dimana pada *pattern* P terdapat juga karakter x di sebelah kanan posisi pengecekan saat ini atau posisi saat *mismatch* terjadi tidak ada karakter x di sebelah kiri posisi pengecekan, maka tidak mungkin dilakukan teknik *character-jump* sehingga dilakukan pergeseran ke kanan sejauh satu karakter saja. Kemudian dilakukan pengecekan ulang dimulai dari belakang.

- Kasus 3

Kasus 3 terjadi ketika kasus 1 dan kasus 2 tidak berlaku, maka geser *pattern* P sehingga P[0] sejajar dengan teks T[i + 1].



Gambar 2.7 Ilustrasi Penggunaan Teknik Character-Jump untuk Kasus 3

Sumber: https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Pencocokan-string-2021.pdf

Pada kasus 3 tersebut terjadi ketidakcocokan pada teks T indeks ke – i dengan karakternya yaitu 'x' (T[i] = x) dimana pada *pattern* P tidak terdapat karakter x di sebelah kanan maupun sebelah kiri posisi pengecekan saat ini, maka tidak mungkin dilakukan teknik *character-jump* sehingga dilakukan pergeseran ke kanan hingga posisi karakter paling kiri *pattern* P sejajar dengan karakter setelah karakter *mismatch* pada teks T (sejajar dengan T[i + 1] dengan i adalah posisi *mismatch* pada teks T). Kemudian dilakukan pengecekan ulang dimulai dari belakang.

4. Regular Expression (Regex)

Regular Expression merupakan urutan karakter yang membentuk pola pencarian yang digunakan untuk mencocokkan dan memanipulasi teks dalam proses pengolahan bahasa alami dan manipulasi string. Regex digunakan secara luas dalam pemrograman, pengolahan teks, dan analisis data untuk mencari, mengganti, dan memvalidasi pola dalam teks.

Regex terdiri dari karakter-karakter literal (misalnya huruf, angka, atau karakter khusus) yang akan dicocokkan secara harfiah, serta karakter khusus yang membantuk metakarakter. Metakarakter ini digunakan untuk menggambarkan pola tertentu dan memberikan fleksibilitas dalam pencarian teks. Beberapa metakarakter yang umum digunakan dalam regex adalah sebagai berikut.

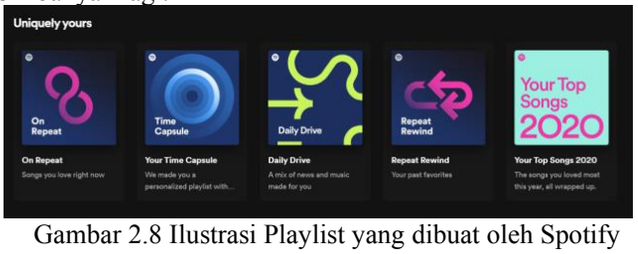
- Dot (.) : mencocokkan dengan satu karakter apapun, kecuali karakter baris baru
- Asterisk (*) : mencocokkan dengan nol atau lebih kemunculan karakter sebelumnya
- Plus (+) : mencocokkan dengan satu atau lebih kemunculan karakter sebelumnya
- Tanda tanya (?) : mencocokkan dengan nol atau satu kemunculan karakter sebelumnya
- Kurung siku ([ ]) : mencocokkan dengan satu karakter yang ada di dalam kurung siku. Dapat digunakan untuk mendefinisikan kelas karakter, seperti [a-z] atau [A-Z]
- Garis vertikal (|) : mencocokkan dengan salah satu dari dua pola yang diberikan sebelumnya
- Karat (^) : mencocokkan awal baris
- Dolar ($) : mencocokkan akhir baris

B. Rekomendasi Playlist Spotify

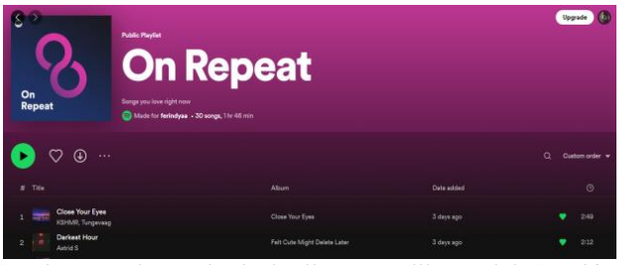
Spotify merupakan salah satu platform streaming musik yang popular dan menawarkan berbagai fitur, termasuk rekomendasi playlist yang disesuaikan dengan preferensi musik pengguna. Spotify menggunakan berbagai metode dan algoritma untuk membuat rekomendasi playlist berdasarkan lagu yang didengarkan oleh pengguna.

Spotifiy mengumpulkan data tentang preferensi musik pengguna, termasuk lagu yang didengarkan, artis yang disukai, genre musik, dan data lainnya seperti penilaian dan ulasan yang diberikan pengguna. Data ini membantu Spotify memahami preferensi musik pengguna dengan lebih baik.

Untuk lebih memahami karakteristik musik yang disukai pengguna, Spotify juga menganalisis atribut lagu seperti tempo, kunci musik, irama, instrumen yang digunakan, dan masih banyak lagi.



Gambar 2.8 Ilustrasi Playlist yang dibuat oleh Spotify
Sumber: Dokumen Pribadi



Gambar 2.9 Ilustrasi Isi Playlist yang dibuat oleh Spotify
Sumber: Dokumen Pribadi

Selain pembuatan playlist otomatis oleh Spotify, platform streaming musik ini juga memungkinkan pengguna untuk membuat playlist sendiri secara manual. Playlist yang dibuat manual oleh pengguna juga dapat berkolaborasi dengan pengguna lain. Hal ini berarti beberapa pengguna lain dapat menambahkan atau menghapus lagu pada playlist yang kita buat.

III. Proses Pembuatan Rekomendasi Playlist

Untuk membuat rekomendasi playlist oleh Spotify, akan dikumpulkan terlebih dahulu data dari pengguna, seperti lagu yang sering didengar, lagu yang disukai, genre lagu paling sering didengar, lirik lagu paling sering didengarkan, artis yang disukai, serta data lainnya seperti penilaian dan ulasan dari pengguna. Setelah mengumpulkan data tersebut, Spotify akan menganalisis atribut lagu, seperti tempo, kunci musik, irama, instrumen yang digunakan, dan lainnya, untuk membantu Spotify dalam memahami karakteristik musik yang disukai pengguna dan menciptakan hubungan antara lagu-lagu berbeda.

Selain itu, Spotify juga menggunakan metode collaborative filtering untuk menghasilkan rekomendasi playlist. Metode ini melibatkan perbandingan preferensi musik pengguna dengan pengguna lain yang memiliki kesamaan dalam preferensi musik. Jika ada pengguna lain yang memiliki kesamaan dalam lagu yang didengarkan, Spotify dapat merekomendasikan lagu-lagu yang disukai oleh pengguna lain tersebut.

Terdapat juga metode lain yang digunakan Spotify yaitu metode *content-based filtering*. Metode ini melibatkan analisis terhadap karakteristik lagu yang disukai oleh pengguna. Misalnya, jika pengguna sering mendengarkan lagu-lagu dengan tempo cepat dan genre pop, Spotify dapat merekomendasikan lagu-lagu dengan karakteristik serupa.

Selain melihat preferensi musik pengguna, Spotify juga mempertimbangkan konteks saat pembuatan rekomendasi playlist. Misalnya, mereka dapat mempertimbangkan waktu hari, suasana hati, aktivitas yang sedang dilakukan, lokasi geografis, dan preferensi musik yang berubah seiring waktu.

Untuk membuat rekomendasi playlist berdasarkan lagu yang sering didengarkan oleh pengguna, perlu diketahui terlebih dahulu kata apa saja yang terdapat dalam lirik lagu yang sering didengar. Kata-kata tersebut kemudian dapat disimpan dalam bentuk array apabila jumlahnya sedikit atau disimpan dalam database apabila jumlahnya banyak. Keempat algoritma *string matching* yang telah dikemukakan pada bagian dasar teori dapat digunakan untuk pembuatan rekomendasi playlist lagu. Pada penulisan makalah ini, penulis menggunakan algoritma Boyer-Moore untuk mencocokkan genre dan algoritma Knuth-Morris-Pratt untuk mencocokkan lirik lagu.

Algoritma dasar pembuatan rekomendasi playlist lagu adalah sebagai berikut:

```
songs : array of String // array komponen lagu
title : String // judul lagu
lyrics : String // lirik lagu
genre : String // genre lagu
keyword : String // kata kunci

playlist = createPlaylist(songs, keyword)

If playlist then
   Playlist berhasil dibuat
Else
   Tidak ada lagu yang cocok dengan kriteria
playlist
```

```
function createPlaylist (songs : array of
String, keyword : String) -> Array of String

for song in songs:
    if bmMatch(genre, keyword) then
         playlist.append(song)
    if kmpMatch(lyrics, keyword) then
         playlist.append(song)

return playlist
```

Program utama akan menggunakan fungsi createPlaylist untuk membuat playlist lagu berdasarkan lirik lagu dan genre yang sering didengar. Fungsi createPlaylist akan menerima masukan berupa array of string yang merupakan daftar komponen lagu dan string yang merupakan keyword atau kata kunci dari lirik sebuah lagu. Pada fungsi createPlaylist akan dilakukan iterasi terhadap setiap kata yang berada di array of string. Pada setiap iterasi, akan digunakan fungsi bmMatch dan kmpMatch dengan masukan genre berupa string dan keyword berupa string. Apabila keyword ditemukan di dalam lirik lagu, maka iterasi akan dilanjutkan dan fungsi akan mengembalikan playlist berupa array of String. Bila keyword tidak ditemukan di dalam lirik lagu, maka fungsi akan mengembalikan array kosong.

IV. Implementasi

Implementasi dari algoritma pembuatan rekomendasi playlist dibuat oleh penulis menggunakan bahasa Python.

A. Fungsi bmMatch

Fungsi bmMatch akan menerima masukan sebuah teks dan sebuah pattern kemudian memeriksa apakah pattern yang dimasukkan berada di dalam teks yang dimasukkan. Fungsi bmMatch akan mengembalikan indeks pertama kemunculan pattern dalam teks, apabila pattern tidak ditemukan dalam teks, fungsi bmMatch akan mengembalikan -1.

```python
def bmMatch(text, pattern):
    n = len(text)
    m = len(pattern)
    if m == 0:
        return 0

    # Inisialisasi tabel geser karakter
    shift = {}
    for i in range(m):
        shift[pattern[i]] = max(1, m - i - 1)

    # Proses pencarian
    i = m - 1
    while i < n:
        k = 0
        while k < m and pattern[m - 1 - k] == text[i - k]:
            k += 1
        if k == m:
            return i - m + 1
        else:
            i += shift.get(text[i], m)

    return -1
```

Gambar 4.1 Fungsi bmMatch
Sumber: Dokumen Pribadi

B. Fungsi kmpMatch

Fungsi kmpMatch akan menerima masukan sebuah teks dan sebuah pattern kemudian memeriksa apakah pattern yang dimasukkan berada di dalam teks yang dimasukkan. Fungsi kmpMatch akan mengembalikan indeks terakhir kemunculan pattern dalam teks, apabila pattern tidak ditemukan dalam teks, fungsi kmpMatch akan mengembalikan -1.

```python
def kmpMatch(text, pattern):
    n = len(text)
    m = len(pattern)

    # Buat tabel lompatan (failure function)
    lps = [0] * m
    j = 0
    i = 1
    while i < m:
        if pattern[i] == pattern[j]:
            j += 1
            lps[i] = j
            i += 1
        else:
            if j != 0:
                j = lps[j - 1]
            else:
                lps[i] = 0
                i += 1

    # Proses pencarian
    i = 0
    j = 0
    while i < n:
        if pattern[j] == text[i]:
            i += 1
            j += 1
            if j == m:
                return i - j
        else:
            if j != 0:
                j = lps[j - 1]
            else:
                i += 1

    return -1
```

Gambar 4.2 Fungsi kmpMatch
Sumber: Dokumen Pribadi

C. Kelas Song

Penggunaan kelas dalam program ditujukan untuk merepresentasikan sebuah lagu dengan atribut-atribut seperti judul (title), lirik (lyrics), dan genre. Setiap lagu diwakili sebagai objek dari kelas ini.

```python
class Song:
    def __init__(self, title, lyrics, genre):
        self.title = title
        self.lyrics = lyrics
        self.genre = genre
```

Gambar 4.3 Kelas Song
Sumber: Dokumen Pribadi

D. Fungsi createPlaylist

Fungsi createPlaylist akan menerima masukan sebuah daftar lagu (songs) dan sebuah kata kunci (keyword) kemudian membuat playlist lagu berdasarkan lirik lagu dan genre yang cocok dengan kata kunci. Fungsi ini menggunakan fungsi bmMatch untuk mencocokkan genre dan fungsi kmpMatch untuk mencocokkan lirik lagu. Fungsi createPlaylist akan mengembalikan daftar lagu yang cocok dengan kriteria playlist.

```python
def createPlaylist(songs, keyword):
    playlist = []

    for song in songs:
        title = song.title
        lyrics = song.lyrics
        genre = song.genre

        # Menggunakan algoritma Boyer-Moore untuk mencocokkan genre
        if bmMatch(genre.lower(), keyword.lower()) != -1:
            playlist.append(song)

        # Menggunakan algoritma KMP untuk mencocokkan lirik lagu
        if kmpMatch(lyrics.lower(), keyword.lower()) != -1:
            playlist.append(song)

    return playlist
```

Gambar 4.4 Fungsi createPlaylist
Sumber: Dokumen Penulis

V. Uji Coba

Penulis membuat sebuah database terlebih dahulu untuk menyimpan lima daftar lagu sering didengarkan yang berisi judul, nama artis, lirik reff, dan genre sebagai berikut.

Title	Lyrics	Genre
A Thousand Years - Christina Perri	I have died everyday waiting for you Darling don't be afraid I have loved you For a thousand years I'll love you for a thousand more	Pop
One Last Time - Ariana Grande	So one last time I need to be, the one, who takes you home One more time	Pop

Stronger - Kelly Clarkson	What doesn't kill you makes you stronger Stand a little taller Doesn't mean I'm lonely when I'm alone	Pop
Skyscraper - Demi Lovato	You can take everything I have You can break everything I am Like I'm made of glass Like I'm made of paper	Pop
Bad Liar - Imagine Dragons	But I'm a bad liar, bad liar Now you know Now you know	Pop

Kemudian akan dilakukan pengujian pembuatan playlist dengan pengecekan keyword yang berbeda-beda sebagai berikut.

- Keyword = “i”

Terlihat bahwa playlist menampilkan semua lagu yang terdapat dalam database karena semua lirik lagu tersebut mengandung kata “i”.

```
PS C:\Makalah Stima> python -u "c:\Makalah Stima\playlist.py"
Playlist berhasil dibuat

================== Playlist ==================
Title: A Thousand Years - Christina Perri
Lyrics: One step closer
I have died everyday waiting for you
Darling don't be afraid I have loved you
For a thousand years
I'll love you for a thousand more
Genre: Pop

Title: One Last Time - Ariana Grande
Lyrics: So one last time
I need to be, the one, who takes you home
One more time
Genre: Pop

Title: Stronger - Kelly Clarkson
Lyrics: What doesn't kill you makes you stronger
Stand a little taller
Doesn't mean I'm lonely when I'm alone
Genre: Pop

Title: Skyscraper - Demi Lovato
Lyrics: You can take everything I have
You can break everything I am
Like I'm made of glass
Like I'm made of paper
Genre: Pop

Title: Bad Liar - Imagine Dragons
Lyrics: But I'm a bad liar, bad liar
Now you know
Now you know
Genre: Pop
```

Gambar 5.1 Uji coba 1
Sumber: Dokumen Pribadi

- Keyword = “fire”

Terlihat bahwa tidak dapat membuat playlist karena semua lagu yang terdapat dalam database tidak ada kata “fire”

```
PS C:\Makalah Stima> python -u "c:\Makalah Stima\playlist.py"
Tidak ada lagu yang cocok dengan kriteria playlist.
PS C:\Makalah Stima>
```

Gambar 5.2 Uji coba 2
Sumber: Dokumen Pribadi

- Keyword = “one”

Terlihat bahwa playlist menampilkan tiga lagu yang terdapat dalam database karena lirik ketiga lagu tersebut mengandung kata “one”.

```
PS C:\Makalah Stima> python -u "c:\Makalah Stima\playlist.py"
Playlist berhasil dibuat

================== Playlist ==================
Title: A Thousand Years - Christina Perri
Lyrics: One step closer
I have died everyday waiting for you
Darling don't be afraid I have loved you
For a thousand years
I'll love you for a thousand more
Genre: Pop

Title: One Last Time - Ariana Grande
Lyrics: So one last time
I need to be, the one, who takes you home
One more time
Genre: Pop

Title: Stronger - Kelly Clarkson
Lyrics: What doesn't kill you makes you stronger
Stand a little taller
Doesn't mean I'm lonely when I'm alone
Genre: Pop

PS C:\Makalah Stima>
```

Gambar 5.3 Uji Coba 3
Sumber: Dokumen Pribadi

- Keyword = “like”

Terlihat bahwa playlist menampilkan satu lagu yang terdapat dalam database karena hanya lirik lagu tersebut mengandung kata “like”.

```
PS C:\Makalah Stima> python -u "c:\Makalah Stima\playlist.py"
Playlist berhasil dibuat

================== Playlist ==================
Title: Skyscraper - Demi Lovato
Lyrics: You can take everything I have
You can break everything I am
Like I'm made of glass
Like I'm made of paper
Genre: Pop

PS C:\Makalah Stima>
```

Gambar 5.4 Uji Coba 4
Sumber: Dokumen Pribadi

VI. KESIMPULAN

Algoritma *string matching* bermanfaat dalam berbagai bidang, salah satunya di bidang musik yaitu pembuatan rekomendasi playlist lagu di Spotify. Dengan menggunakan algoritma *string matching*, lagu-lagu yang sering diputar dan lagu yang disukai oleh pengguna dapat dideteksi apabila keyword atau kata kunci dalam lirik lagu sudah didefinisikan terlebih dahulu dan terdapat database lagu beserta atribut-atributnya seperti judul, artis, lirik, dan genre. Berdasarkan uji coba yang dilakukan oleh penulis, program penulis yang menggunakan algoritma *string matching* berupa algoritma Boyer-Moore dan algoritma Knuth-Morris-Pratt sudah dapat membuat rekoemndasi playlist lagu berdasarkan lagu yang sering didengar dan pencocokan keyword dengan cukup baik.

LINK VIDEO YOUTUBE

https://youtu.be/CEeYeyiBPKo

LINK GITHUB

https://github.com/ferindya/Makalah_Stima.git

VII. UCAPAN TERIMA KASIH

Terima kasih sebesar-besarnya kepada Tuhan Yang Maha Esa karena berkat rahmat dan karunia-Nya penulis diberikan kelancaran sehingga penulisan makalah IF2211 Strategi Algoritma semester II tahun 2022/2023 dapat terselesaikan dengan baik. Penulis juga ingin menyampaikan terima kasih kepada keluarga yang telah mendukung saya dalam penulisan makalah ini. Penulis juga mengucapkan terima kasih kepada Bapak Dr. Ir. Rinaldi Munir, M.T. selaku dosen pengampu mata kuliah IF2211 Strategi Algoritma yang telah memberikan materi untuk penulisan makalah ini. Penulis juga meminta maaf, apabila terdapat kesalahan dalam penulisan makalah ini, penulis berharap makalah ini dapat berguna bagi para pembaca.

REFERENSI

[1] https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Pencocokan-string-2021.pdf, diakses pada tanggal 15 Mei 2023

[2] https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2018-2019/String-Matching-dengan-Regex-2019.pdf, diakses pada tanggal 15 Mei 2023

[3] https://support.spotify.com/id-id/article/social-recommendations-in-playlists/, diakses pada tanggal 17 Mei 2023

[4] https://support.spotify.com/id-id/article/create-playlists/, diakses pada tanggal 17 Mei 2023

[5] https://www.lifewire.com/spotify-tips-and-tricks-4105795, diakses pada tanggal 18 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

Ferindya Aulia Berlianty - 13521161