Pencocokan Pola Musik Monofonik dengan Memanfaatkan Algoritma Pattern Matching

Arleen Chrysantha Gunardi - 13521059
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
E-mail (gmail): 13521059@std.stei.itb.ac.id

***Abstract*—Musik merupakan salah satu bentuk karya seni berupa serangkaian nada yang dirangkai sedemikian rupa sehingga dapat menciptakan harmoni yang indah. Seiring dengan berkembangnya teknologi, musik dapat dikenali berdasarkan suatu potongan tema atau pola, sehingga musik dapat dicari berdasarkan tema atau pola dari musik tersebut. Prinsip dari pengenalan pola musik sederhana adalah dengan algoritma pencocokan pola (*pattern matching*) musik monofonik, yaitu salah satu tekstur musik paling sederhana yang terdiri dari sebuah rangkaian melodi tanpa iringan musik lainnya. Dengan demikian, dua buah musik dapat dicocokkan berdasarkan setiap not dan nada yang merangkai musik tersebut.**

Keywords—musik; musik monofonik; pattern matching; boyer moore; levensthein

I. Introduction

Musik merupakan salah satu bentuk karya seni berupa serangkaian nada yang dirangkai sedemikian rupa sehingga dapat menciptakan suatu harmoni yang indah. Seiring berkembangnya zaman dan peradaban, musik tetap menjadi salah satu karya seni yang menjadi media untuk merepresentasikan berbagai emosi, kebudayaan, dan kisah yang ingin disampaikan komposernya. Adapun bentuk musik kini beragam, mulai dari musik yang dimainkan oleh alat musik klasik, tradisional, maupun kontemporer, atau musik yang diproduksi secara digital.

Seiring dengan berkembangnya teknologi, kini musik dapat dikenali berdasarkan potongan tema atau pola dari musik tersebut. Terdapat berbagai perangkat lunak untuk mengenali pola musik, misalnya fitur pencarian suara pada Google atau aplikasi Shazam. Pada dasarnya, pengenalan pola musik ini dilakukan dengan algoritma pencocokan pola (*pattern matching*), sehingga pengguna dapat memberikan masukan potongan rangkaian melodi atau lagu dan mendapatkan judul lagu yang cocok dengan potongan melodi dari pengguna.

Program pengenalan pola musik sederhana merupakan program yang melakukan algoritma pencocokan pola antara dua musik monofonik. Musik monofonik adalah salah satu tekstur musik yang paling sederhana karena hanya terdiri dari serangkai nada tanpa iringan nada lain pada waktu bersamaan. Dengan kata lain, dalam satu waktu, hanya terdapat satu buah nada yang terdengar, tidak ada musik iringan atau harmoni lain yang mengiringi.

II. Landasan Teori

A. Definisi Pattern Matching

Pencocokan pola (*pattern matching*) merupakan algoritma untuk mencari pola yang sama antara dua buah objek sekuensial, misalnya string. Pada dasarnya, *pattern matching* mencari apakah suatu objek mengandung suatu pola yang ingin dicocokkan, lalu mengembalikan lokasi di mana pola tersebut ditemukan.

Umumnya, terdapat beberapa komponen dalam algoritma *pattern matching*, yaitu objek sekuensial utama yang ingin dilakukan pencocokan dan juga objek sekuensial pola yang ingin dicocokkan dengan objek sekuensial utama. Adapun panjang dari objek sekuensial pola diasumsikan lebih pendek dibandingkan objek sekuensial utama yang ingin dilakukan pencocokan.

Terdapat beberapa cara untuk menyelesaikan persoalan *pattern matching*, yaitu algoritma Brute Force, algoritma Knuth-Morris-Pratt, algoritma Boyer-Moore. Ketiganya memiliki tingkat efisiensi yang berbeda dalam mencocokkan pola. Pencocokan pola dengan cara Brute Force dilakukan dengan cara mencocokkan tiap karakter sekuensial objek, sehingga kurang efisien. Algoritma Knuth-Morris-Pratt merupakan algoritma yang lebih efisien dibandingkan dengan Brute Force karena algoritma ini melakukan pencocokan dengan menentukan karakter sekuensial mana saja yang harus dibandingkan, sehingga tidak semua karakter dicocokkan. Namun, terdapat algoritma yang lebih efisien dibandingkan kedua algoritma tersebut, yaitu algoritma Boyer-Moore.

B. Algoritma Boyer-Moore

Pencocokan pola dapat dilakukan dengan algoritma Boyer-Moore. Pada algoritma Boyer-Moore, pencocokan pola dilakukan berdasarkan dua macam teknik, yaitu *the looking-glass technique* dan juga *the character-jump technique*. Kedua teknik inilah yang membuat algoritma Boyer-Moore lebih efisien dibandingkan dengan algoritma pencocokan pola lainnya.

Misalkan T adalah objek sekuensial utama yang ingin dilakukan pencocokan dan P adalah objek sekuensial pola yang ingin dicocokkan dengan T. Pencocokan pada algoritma Boyer-

Moore dilakukan dengan membandingkan karakter terakhir dari P, berbeda dengan kedua algoritma lainnya yang memulai pencocokan dengan membandingkan karakter pertama dari P terlebih dahulu. Apabila karakter yang sedang dibandingkan sama / cocok, maka karakter selanjutnya yang dibandingkan adalah karakter sebelum karakter saat ini. Dengan kata lain, pencocokan pola dengan algoritma Boyer-Moore dilakukan secara "mundur".

Ketika karakter yang sedang dibandingkan tidak cocok, maka ada beberapa macam cara pendekatan untuk menangani beberapa kasus yang berbeda. Misalkan x merupakan karakter tak cocok pada T. Jika P mengandung x, maka geser P sehingga kemunculan terakhir elemen x pada P kini sejajar dengan posisi x pada T. Kasus lainnya adalah ketika P mengandung x, tetapi tidak memungkinkan untuk menggeser P, maka P digeser sebanyak satu karakter ke kanan. Selain kedua kasus tersebut, maka geser P sehingga elemen pertama P sejajar dengan elemen pertama T.

Untuk mempermudah perhitungan dalam algoritma Boyer-Moore, terdapat fungsi untuk mencatat indeks kemunculan terakhir dari setiap elemen T pada P. Fungsi ini dapat disebutkan sebagai Last Occurence Function. Fungsi ini mencatat setiap indeks terakhir dari seluruh elemen T pada P. Jika suatu karakter pada T tidak ditemukan pada P, maka nilai fungsi Last Occurence dari elemen tersebut bernilai -1.

C. Jarak Levensthein dan Persentase Kemiripan

Jarak Levensthein antara dua buah objek sekuensial merepresentasikan jarak atau kemiripan antara dua buah objek sekuensial. Jarak Levensthein merupakan banyaknya operasi penambahan (*insertion*), penghapusan(*deletion)*, dan/atau penggantian (*substitution*) yang harus dilakukan untuk mengganti suatu objek sekuensial menjadi objek sekuensial lainnya. Semakin besar suatu jarak Levensthein, maka semakin berbeda pula kedua objek sekuensial yang dibandingkan.

Misalkan S adalah objek sekuensial asal dan T adalah objek sekuensial target yang ingin dikalkulasikan jaraknya. Pertama-tama, perhitungan jarak Levensthein dilakukan dengan menginisialisasi sebuah matriks berukuran baris sejumlah panjang S dan kolom sejumlah panjang T. Matriks tersebut dievaluasi dengan cara mengiterasi baris yang direpresentasikan sebagai i dan kolom yang direpresentasikan sebagai j. Sel pada matriks bernilai nol (0) jika nilai pada baris ke-i kolom ke-0 adalah sama dengan nilai pada baris ke-0 kolom ke-j. Sebaliknya, jika tidak sama, maka sel pada matriks bernilai satu (1). Lalu, nilai suatu sel diganti menjadi nilai terkecil di antara nilai sel di atasnya ditambahkan dengan satu, nilai sel di kirinya ditambahkan dengan satu, dan nilai sel di kiri atasnya ditambahkan dengan nilai sel yang sedang dievaluasi saat itu. Iterasi dilakukan hingga mencapai baris dan kolom terakhir matriks. Jarak Levensthein tersebut dapat diambil dari nilai yang tersimpan pada baris terakhir dan kolom terakhir matriks.

Setelah mendapatkan jarak Levensthein antara dua buah objek sekuensial, maka persentase kemiripan antara kedua objek tersebut dapat dikalkulasikan. Persentase kemiripan direpresentasikan sebagai rasio antara selisih jarak Levensthein dengan jarak terpanjang di antara kedua objek, dengan jarak terpanjang di antara kedua objek. Semakin besar persentase kemiripan, maka kedua objek tersebut semakin mirip. Ini berarti jarak semakin besar jarak Levensthein antara kedua objek, maka akan semakin kecil persentase kemiripan antara kedua objek tersbut.

D. Musik dan Teori Musik Dasar

Musik merupakan rangkaian dari nada yang dirangkai dengan irama, melodi, dan harmoni, sehingga menghasilkan suatu karya seni. Musik memiliki tekstur. Tekstur musik membedakan bagaimana musik terdengar. Salah satu tekstur musik adalah musik monofonik, yaitu musik yang hanya terdiri atas satu suara melodi, tanpa diikuti oleh musik pengiring atau harmoni lainnya. Tekstur musik monofonik merupakan tekstur musik paling sederhana karena hanya terdiri dari satu rangkaian melodi.

Musik dapat direpresentasikan dalam berbagai notasi, misalnya notasi not angka atau not balok. Setiap not, baik dalam notasi not angka maupun not balok, diidentifikasikan berdasarkan tiga hal, yaitu waktu mulai, bobot / durasi, dan nada (*pitch*). Waktu mulai suatu not merepresentasikan kapan not tersebut dimulai. Waktu mulai ini merupakan ketukan suatu not berada, dapat direpresentasikan sebagai bilangan real dimulai dari 1. Bobot / durasi suatu not merepresentasikan seberapa lama not tersebut dimainkan atau dinyanyikan, dengan kata lain, seberapa besar bobot dari not tersebut. Durasi suatu not berupa berapa ketuk not tersebut dimainkan atau dinyanyikan. Durasi ini dapat direpresentasikan sebagai bilangan real yang lebih dari nol. Nada (*pitch*) dari suatu not merepresentasikan frekuensi yang dihasilkan dari not tersebut. Adapun standard dari frekuensi yang menjadi acuan yang umum dipakai adalah frekuensi 440 Hz, yaitu nada A pada oktaf keempat (A4).

Adapun hubungan antara frekuensi dengan nada dapat dihitung dengan rumus berikut.

$$freq = note\ 2^{\frac{N}{12}} \quad (1)$$

$$N = -12 \log_2 (\frac{freq}{note}) \quad (2)$$

Frekuensi (freq) mewakilkan frekuensi nada yang ingin dicari (Hz), note mewakilkan frekuensi acuan yang dipakai, yaitu A4 440 Hz, dan N mewakilkan nilai jarak / interval antara dua nada acuan A4 dengan nada yang ingin dicari frekuensinya. N berupa bilangan bulat yang dapat bernilai negatif maupun positif. Nilai N ditentukan dengan menggunakan prinsip interval nada.

III. PEMBAHASAN

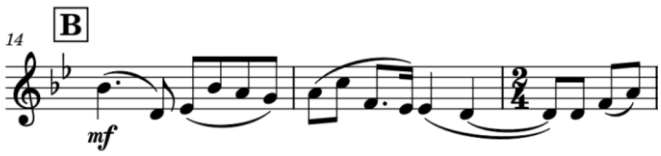
A. Konsep Representasi Musik dalam Program

Program pengenalan pola musik monofonik sederhana dilakukan dengan menerapkan algoritma pencocokan pola (*pattern matching*). Karena tekstur musik yang digunakan sebagai sampel dalam program ini adalah musik monofonik, maka artinya musik dalam program ini hanya berupa rangkaian melodi tanpa iringan. Ini berarti musik monofonik hanya memiliki satu not dalam satu waktu. Karena hanya terdiri dari satu not dalam satu waktu dan not-not tersebut tersusun secara sekuensial, maka musik monofonik dapat diperlakukan sebagai objek sekuensial tunggal (satu dimensi), layaknya sebuah String.

Dengan demikian, algoritma pencocokan string (*string matching*), seperti algoritma Boyer-Moore, dapat diterapkan pada program sederhana ini.

Dalam melakukan pencarian pencocokan pola musik, sebuah musik (lagu) direpresentasikan sebagai kumpulan dari not. Adapun sebuah not direpresentasikan dengan komponen waktu mulai, durasi, serta nadanya. Kumpulan not tersebut membentuk lagu, dengan asumsi bahwa kumpulan not sekuensial berdasarkan waktu mulai dan durasinya, tidak ada dua not dalam waktu yang bersamaan.

Misalkan pada potongan partitur musik "Love Theme from Cinema Paradiso" karya Ennio & Andrea Morricone pada Gambar 3.1, setiap not dapat diidentifikasikan berdasarkan waktu mulai, durasi, dan nadanya. Misalkan ketukan pertama pada bar ke-14 dianggap sebagai ketukan ke-1, maka nada Bb4 (not pertama pada potongan partitur) dimulai pada ketukan ke-1 dengan durasi 1.5 ketukan dan frekuensi 466.2 Hz , dilanjutkan dengan nada D4 (293.6 Hz) yang dimulai pada ketukan ke-2.5 dengan durasi 0.5 ketukan. Kemudian, terdapat nada Eb4 (311.2 Hz) pada ketukan ke-3 dengan durasi 0.5, dan seterusnya.



Gambar 3.1 Potongan Partitur Musik "Love Theme from Cinema Paradiso" Karya Ennio & Andrea Morricone

Berdasarkan waktu mulai, durasi, dan nada, maka diperoleh suatu lagu yang terdiri atas kumpulan not (*list of notes*) yang merepresentasikan potongan partitur pada Gambar 3.1 sebagai berikut:

$$S = \{(1, 1.5, 466.2), (2.5, 0.5, 293.6), (3, 0.5, 311.2),$$
$$(3.5, 0.5, 466.2), (4, 0.5, 440), (4.5, 0.5, 392),$$
$$(5, 0.5, 440), (5.5, 0.5, 523.2), (6, 0.75, 698.5),$$
$$(6.75, 0.25, 311.2), (7, 1, 311.2), (8,1.5, 293.6),$$
$$(9.5, 0.5, 293.6), (10, 0.5, 698.5), (10.5, 0.5, 440)\}$$

B. *Program Sederhana Pengenalan Musik Monofonik*

Program sederhana pengenalan musik monofonik dibuat dengan menggunakan bahasa pemrograman Java dan menerapkan konsep pemrograman berorientasi objek (*object oriented programming*). Program ini merupakan program sederhana berbasis *command line interface* (CLI).

Secara umum, program akan memiliki kumpulan pola / tema lagu yang disimpan, kemudian program akan menerima masukan dari pengguna berupa lagu yang ingin dicari dan dicocokkan. Masukan dari pengguna adalah berupa list dari not-not yang menyusun suatu lagu. Program akan mengiterasi lagu-lagu yang dimiliki dan mencocokkannya dengan lagu dari pengguna. Untuk melakukan pencocokan pola antarlagu, digunakan algoritma Boyer-Moore dan kalkulasi persentase kemiripan dengan jarak Levensthein, sehingga dapat diperoleh lagu yang sesuai dengan pola yang dicari.

C. *Struktur Program: Kelas Note*

Kelas Note merepresentasikan not. Kelas ini memiliki tiga buah atribut bertipe bilangan real, masing-masing merepresentasikan waktu mulai, durasi, dan nada (*pitch*) dari not tersebut. Berikut merupakan potongan kode untuk mendefinisikan kelas Note.

```
public class Note {
    /* attributes */
    private Double start;
    private Double duration;
    private Double pitch;

    /* constructor */
    public Note(Double start, Double duration, Double pitch){
        this.start = start;
        this.duration = duration;
        this.pitch = pitch;
    }

    /* getter and setter */


    /* other methods */
    public void transpose(int numTranspose){
        pitch *= Math.pow(2., (Double.valueOf(numTranspose) /
                12.));
    }

    public int calcNumTranspose(Note note){
        return (int) Math.round(-12 *
                (Math.log(this.pitch / note.pitch) /
                Math.log(2)));
    }
}
```

Metode transpose adalah untuk mengubah frekuensi nada (*pitch*) suatu not berdasarkan interval transpos (numTranspose). Kalkulasi frekuensi ini menggunakan persamaan (1), sehingga diperoleh frekuensi hasil transpos. Metode calcNumTranspose adalah metode untuk menghitung interval transpos dari suatu nada ke nada lainnya. Kalkulasi ini menggunakan persamaan (1) yang diturunkan menjadi persamaan (2).

D. *Struktur Program: Kelas Song*

Kelas Song merepresentasikan sebuah lagu yang terdiri atas kumpulan not. Kelas ini memiliki atribut judul lagu dan juga list of Notes yang berurutan berdasarkan waktu mulainya.

```
public class Song {
    /* attributes */
    private String title;
    private List<Note> notes;

    /* constructor */
    public Song(){
        title = "";
        notes = new ArrayList<Note>();
    }
    public Song(String title){
        this.title = title;
        this.notes = new ArrayList<Note>();
    }
    public Song(String title, List<Note> notes){
        this.title = title;
        this.notes = notes;
    }

    /* getter and setter */


    /* other methods */
    public boolean containsNote(Note note){ ... }
    public void addNote(Note note){ ... }
    public void removeNote(Note note){ ... }
    public void removeNote(int idx){ ... }
    public void setNoteIdx(int idx, Note note){ ... }

    public Double findMinimumDurationGrouping(){
        Double min = notes.stream()
                .min(Comparator.comparing(Note::getDuration))
                .orElseThrow(NoSuchElementException::new)
                .getDuration();
        for (Note note : notes) {
            if (note.getDuration() % min > 0 &&
                    note.getDuration() % min < min){
                min = note.getDuration() % min;
            }
```

```
        }
        return min;
    }

    public String toString(Double minimumDurationGrouping){
        StringBuilder songStr = new StringBuilder();
        for (Note note : notes){
            for (int i = 0; i <
                    note.getDuration()/minimumDurationGrouping;
                    i++){
                songStr.append(String.format(
                        "%.1f ",note.getPitch()).toString());
            }
        }
        songStr.deleteCharAt(songStr.length()-1);
                // remove last whitespace
        return songStr.toString();
    }

    public void transpose(int numTranspose){
        for (Note note : notes){
            note.transpose(numTranspose);
        }
    }

    public void printSong(){
        System.out.println(title);
        System.out.println("(start, duration, pitch)");
        for (Note note : notes) {
            System.out.println(String.format(
                    "(%.2f, %.2f, %.1f)", note.getStart(),
                    note.getDuration(), note.getPitch()));
        }
    }
}
```

Metode findMinimumDurationGrouping adalah metode untuk mencari faktor terkecil durasi not dalam suatu lagu, sehingga semua not dalam lagu tersebut merupakan kelipatan dari pengelompokan durasi minimum tersebut. Misalkan pada potongan lagu pada Gambar 3.1, pengelompokan durasi not minimum adalah 0.25 ketukan yang terdapat pada nada Eb4 di ketukan ke-6.75. Pengelompokan durasi minimum ini dipakai sebagai satuan yang dipakai ketika melakukan pencocokan pola, sehingga not dengan nilai kelipatan *n* dari pengelompokan durasi minimum dianggap sebagai suatu objek sekuensial dengan panjang *n*.

Pengelompokan durasi minimum ini akan memengaruhi bagaimana sebuah lagu direpresentasikan sebagai sebuah String. Metode toString merupakan metode yang mengembalikan sebuah String yang merepresentasikan sebuah lagu. String tersebut terdiri dari nada-nada (dalam Hz) yang dipisahkan dengan spasi.

Misalkan terdapat lagu pada Gambar 3.1. Jika diterjemahkan ke dalam bentuk String dengan pengelompokan durasi minimum 0.25, maka lagu tersebut adalah sebagai berikut:

466.2 466.2 466.2 466.2 466.2 466.2 293.6 293.6 311.2 311.2
466.2 466.2 440.0 440.0 392.0 392.0 440.0 440.0 523.2 523.2
698.5 698.5 698.5 311.2 311.2 311.2 311.2 311.2 293.6 293.6
293.6 293.6 293.6 293.6 293.6 293.6 698.5 698.5 440.0 440.0

E. Struktur Program: Kelas Matcher

Kelas Matcher merupakan kelas dengan atribut mainSong sebagai lagu yang ingin dilakukan pencocokan dan atribut patternSong sebagai lagu yang inign dicocokkan dengan mainSong.

```
public class Matcher implements IBoyerMoore, ILevenstein {
    /* attributes */
    private Song mainSong;
    private Song patternSong;

    /* constructor */
    public Matcher(Song mainSong, Song patternSong){
        this.mainSong = mainSong;
        this.patternSong = patternSong;
    }
```

```
    /* getter and setter */

    /* other methods */
    public Double findMinimumDurationGrouping(){
        return (mainSong.findMinimumDurationGrouping() <
                patternSong.findMinimumDurationGrouping()) ?
                mainSong.findMinimumDurationGrouping() :
                patternSong.findMinimumDurationGrouping();
    }

    public String mainSongToString(){
        return mainSong.toString(
                findMinimumDurationGrouping());
    }

    public String patternSongToString(){
        return patternSong.toString(
                findMinimumDurationGrouping());
    }

    /* Boyer-Moore Algorithm */
    public int boyerMooreMatch(){
        String[] mainSongArr = mainSongToString().split(" ");
        String[] patternSongArr = patternSongToString().split(" ");
        Map<String, Integer> lastOccurence = lastOccurence(
                                    mainSongArr, patternSongArr);

        int n = mainSongArr.length;
        int m = patternSongArr.length;
        int i = m-1;
        if (i > n-1){
            return -1;
        }
        int j = m-1;
        do {
            if (patternSongArr[j].equals(mainSongArr[i]) ||
                    Math.abs(Double.parseDouble(mainSongArr[i])
                    - Double.parseDouble(patternSongArr[j])) /
                    Double.parseDouble(patternSongArr[j])
                    >= 0.75){
                if (j==0){
                    return i; // matches
                }
                i--;
                j--;
            } else {
                int last = lastOccurence.get(mainSongArr[i]);
                i += m - Math.min(j, 1+last);
                j = m-1;
            }
        } while (i<=n-1);
        return -1;
    }

    public Map<String, Integer> lastOccurence(
            String[] mainSongArr, String[] patternSongArr){
        Map<String, Integer> last =
                new HashMap<String, Integer>();
        for (String note : mainSongArr){
            if (!last.containsKey(note)){
                last.put(note,
                        searchLastIdx(note, patternSongArr));
            }
        }
        return last;
    }

    public int searchLastIdx(String note, String[] patternSongArr){
        int idx = patternSongArr.length-1;
        while (idx>=0){
            if (patternSongArr[idx].equals(note)){
                return idx;
            } else {
                idx--;
            }
        }
        return -1;
    }

    public int levenstein(String[] mainSongArr,String[] patternSongArr)
    {
        int mainSongLength = mainSongArr.length;
        int patternSongLength = patternSongArr.length;
        int[][] levenstheinMatrix = new
                int[mainSongLength+1][patternSongLength+1];
        Arrays.stream(levenstheinMatrix)
              .forEach(elmt -> Arrays.fill(elmt, 0));

        for (int i=1; i<=mainSongLength; i++){
            levenstheinMatrix[i][0] = i;
        }
        for (int j=1; j<=patternSongLength; j++){
            levenstheinMatrix[0][j] = j;
        }

        for (int i=1; i<=mainSongLength; i++){
            for (int j=1; j<=patternSongLength; j++){
                if (mainSongArr[i-1] == patternSongArr[j-1] ||
                        Math.abs(Double.parseDouble(mainSongArr[i-1]) -
                        Double.parseDouble(patternSongArr[j-1])) /
                        Double.parseDouble(patternSongArr[j-1]) >=
                        0.75){
                    levenstheinMatrix[i][j] =
                            levenstheinMatrix[i-1][j-1];
                } else {
                    levenstheinMatrix[i][j] =
                            Math.min(levenstheinMatrix[i-1][j-1],
                            Math.min(levenstheinMatrix[i-1][j],
                            levenstheinMatrix[i][j-1])) + 1;
                }
            }
        }
```

```
        return levenstheinMatrix[mainSongLength][patternSongLength];
    }

    public Double calcSimilarityPercentage() {
        String[] mainSongArr = mainSongToString().split(" ");
        String[] patternSongArr = patternSongToString().split(" ");

        int levenstheinValue = levensthein(mainSongArr, patternSongArr);
        Double similarity = 0.;
        int maxLength = Math.max(
                mainSongArr.length, patternSongArr.length);
        if (maxLength != 0){
            similarity = Double.valueOf(maxLength - levenstheinValue) /
                    Double.valueOf(maxLength);
        } else {
            similarity = 1.;
        }
        return similarity;
    }
}
```

Metode boyerMooreMatch pada kelas Matcher adalah metode yang mengembalikan indeks ditemukannya pola patternSong pada mainSong dengan menggunakan algoritma Boyer-Moore. Jika pola tidak ditemukan, maka indeks yang dikembalikan adalah indeks tak valid, yaitu -1. Pencocokan Boyer-Moore membutuhkan fungsi untuk mencari kemunculan terakhir suatu nada di pola patternSong. Pencarian kemunculan terakhir ini dilakukan dengan metode lastOccurence yang mengembalikan sebuah Map nada dan indeks terakhir ditemukan pada patternSong.

Metode levensthein adalah metode untuk menghitung berapa banyak operasi yang dibutuhkan untuk mengubah mainSong menjadi patternSong. Nilai ini akan dipakai untuk menghitung persentase kemiripan antara mainSong dengan patternSong yang dikalkulasikan melalui metode calcSimilarity.

F. Struktur Program: Kelas Solver

Kelas Solver merupakan kelas dengan atribut daftar lagu yang disimpan pada program dan atribut lagu yang ingin dicocokkan dengan lagu yang tersimpan.

```
public class Solver {
    /* attributes */
    private List<Song> songDatabase;
    private Song songToBeMatched;

    /* constructor */
    public Solver(){
        this.songDatabase = new ArrayList<Song>();
        this.songToBeMatched = new Song();
    }
    public Solver(Song songToBeMatched){
        this.songDatabase = new ArrayList<Song>();
        this.songToBeMatched = songToBeMatched;
    }
    public Solver(List<Song> songDatabase, Song songToBeMatched){
        this.songDatabase = songDatabase;
        this.songToBeMatched = songToBeMatched;
    }

    /* getter and setter */

    /* other methods */
    public boolean containsSongDatabase(Song song){ ... }
    public void addSongDatabase(Song song){ ... }
    public void removeSongDatabase(Song song){ ... }
    public void removeSongDatabase(int idx){ ... }

    public List<Song> findMatch(){
        List<Song> matchedSongs = new ArrayList<Song>();
        for (Song songDB: songDatabase) {
            Matcher matcher = new Matcher(songToBeMatched, songDB);
            if (matcher.boyerMooreMatch() != -1){
                matchedSongs.add(songDB);
            } else {
                if (matcher.calcSimilarityPercentage()>=0.75){
                    matchedSongs.add(songDB);
                }
            }
        }
        return matchedSongs;
    }
}
```

Metode findMatch merupakan metode untuk mencari semua lagu pada songDatabase yang cocok dengan lagu yang ingin dicocokkan (songToBeMatched). Program akan mengiterasi semua lagu pada songDatabase lalu menjalankan algoritma Boyer-Moore untuk mencocokkan polanya. Apabila tidak ditemukan pola yang cocok, maka perhitungan persentase kemiripan dilakukan. Kedua lagu dianggap cocok jika persentase kemiripannya di atas 75%.

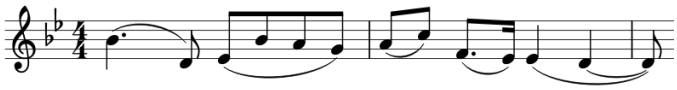
G. Struktur Program: Program Utama

Program utama pertama-tama akan meminta input lagu dari pengguna, kemudian melakukan pencocokan terhadap semua lagu pada database. Metode initSolver merupakan metode untuk melakukan konfigurasi dan menginisialisasi lagu-lagu yang disimpan pada “kamus lagu” program. Jika terdapat lagu yang cocok dengan lagu dari pengguna, maka program akan mencetak judul lagu tersebut.

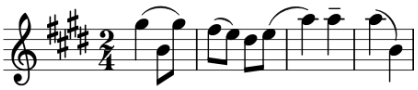
```
public class Main {
    public static Solver initSolver(){ ... }

    public static void main(String[] args) {
        Song inputSong = new Song();
        boolean isInputValid = false;
        List<Note> notes = new ArrayList<Note>();

        while (!isInputValid){
            System.out.println("Please input the song you want to search!");
            System.out.println("<start> <duration> <pitch>");
            System.out.println("Input -1 to stop");

            String note = "";
            notes.clear();
            do {
                Scanner inputScan = new Scanner(System.in);
                note = inputScan.nextLine();
                if (note.equals("-1")){
                    isInputValid = true;
                    break;
                }
                String[] noteComponent = note.split(" ");
                if (noteComponent.length != 3){
                    break;
                }
                Double start = Double.parseDouble(noteComponent[0]);
                Double duration = Double.parseDouble(noteComponent[1]);
                Double pitch = Double.parseDouble(noteComponent[2]);
                if (start<1. || duration<0. || pitch<0.){
                    break;
                }
                notes.add(new Note(start, duration, pitch));
            } while (!note.equals("-1"));
        }

        inputSong.setNotes(notes);

        if (notes.size() > 0){
            Solver solver = initSolver();
            solver.setSongToBeMatched(inputSong);
            List<Song> matchInputSong = solver.findMatch();
            if (matchInputSong.size()>0){
                System.out.println(String.format(
                        "%d matches found!", matchInputSong.size()));
                for (Song match : matchInputSong){
                    System.out.println(match.getTitle());
                }
            } else {
                System.out.println("No matches found!");
            }
        }
    }
}
```

H. Pengujian Program

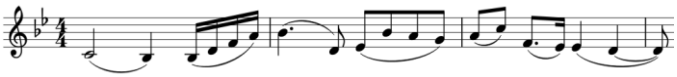
Pengujian program dilakukan dengan memasukkan sebuah lagu yang ingin dicari. Pada pengujian ini, program memiliki dua lagu yang ada pada “kamus lagu” program, yaitu hasil transkrip partitur pada Gambar 3.2, yaitu tema dari “Love Theme from Cinema Paradiso” karya Ennio & Andrea Morricone, serta Gambar 3.3, yaitu tema dari “Salut d’Amour” karya Edward Elgar.



Gambar 3.2 Potongan Partitur Musik "Love Theme from Cinema Paradiso" Karya Ennio & Andrea Morricone dalam Program



Gambar 3.3 Potongan Partitur Musik "Salut d'Amour" Karya Edward Elgar dalam Program



Gambar 3.4 Potongan Partitur Musik Masukan Pengguna

Adapun lagu yang ingin dimasukkan adalah hasil transkrip berdasarkan Gambar 3.4. Dari masukan ini, ingin dicari lagu yang cocok pada kamus lagu program. Apabila diterjemahkan, potongan partitur musik pada Gambar 3.4 adalah sebagai berikut:

```
1. 2. 262.6
3. 1. 246.9
4. 0.25 246.9
4.25 0.25 293.6
4.5 0.25 698.5
4.75 0.25 440.
5. 1.5 466.2
6.5 0.5 293.6
7. 0.5 311.2
7.5 0.5 466.2
8. 0.5 440.
8.5 0.5 392.
9. 0.5 440.
9.5 0.5 523.2
10. 0.75 698.5
10.75 0.25 311.2
11. 1. 311.2
12. 1.5 293.6
```

Dengan demikian, berikut adalah hasil pengujian.

```
PS D:\kuliah\smt4\IF2211_Strategi_Algoritma\makalah\MusicRecognition\MusicRecognitionProject\out> java Main
Please input the song you want to search!
<start> <duration> <pitch>
Input -1 to stop
1. 2. 262.6
3. 1. 246.9
4. 0.25 246.9
4.25 0.25 293.6
4.5 0.25 698.5
4.75 0.25 440.
5. 1.5 466.2
6.5 0.5 293.6
7. 0.5 311.2
7.5 0.5 466.2
8. 0.5 440.
8.5 0.5 392.
9. 0.5 440.
9.5 0.5 523.2
10. 0.75 698.5
10.75 0.25 311.2
11. 1. 311.2
12. 1.5 293.6
-1
1 matches found!
Love Theme from Cinema Paradiso
PS D:\kuliah\smt4\IF2211_Strategi_Algoritma\makalah\MusicRecognition\MusicRecognitionProject\out>
```

Gambar 3.5 Hasil Pengujian Program

IV. Simpulan

Pengenalan pola musik monofonik sederhana dapat dilakukan dengan menerapkan algoritma pencocokan pola (*pattern matching*), yaitu algoritma Boyer-Moore dan kalkulasi persentase kemiripan dengan jarak Levensthein. Algoritma pencocokan pola (String) dapat dilakukan karena musik monofonik dapat direpresentasikan sebagai objek sekuensia satu dimensi.

Program sederhana ini masih memiliki banyak keterbatasan. Musik yang dapat dicocokkan adalah musik dengan tingkat kemiripan yang tinggi. Musik dengan tema yang sama namun memiliki variasi tema yang berbeda akan dianggap sebagai tidak cocok karena program ini menerapkan prinsip *exact matching*. Adapun representasi not pada program ini masih terbatas pada waktu mulai, durasi, dan nada, tanpa mempertimbangkan artikulasi, instrumen, tempo, dan dinamika.

V. Ucapan Terima Kasih

Puji dan syukur penulis panjatkan kepada Tuhan Yang Maha Esa atas berkat, penyertaan, serta karunia-Nya, sehingga penulis dapat menyelesaikan makalah berjudul "Pencocokan Pola Musik Monofonik dengan Memanfaatkan Algoritma Pattern Matching". Penulis mengucapkan terima kasih atas seluruh dukungan yang diberikan terhadap penulis dalam penulisan makalah ini, yaitu kepada:

1. Bapak Dr. Rinaldi Munir, Ibu Dr. Nur Ulfa Maulidevi, serta Bapak Dr. Rila Mandala, sebagai dosen-dosen pengajar IF2211 Strategi Algoritma, atas bimbingan, pengajaran, dan ilmu yang telah dibagikan kepada penulis serta teman-teman mahasiswa Teknik Informatika,
2. Teman-teman saya yang telah membantu dan mendukung saya dalam menyelesaikan makalah ini,
3. Orang tua saya yang selalu memberikan dukungan hingga saat ini, dan
4. Para penulis jurnal dan artikel mengenai teori dasar musik serta algoritma pencocokan musik yang karyanya penulis jadikan referensi dan acuan dalam penulisan makalah ini.

Akhir kata, penulis berharap semoga makalah ini dapat bermanfaat.

Github Repository Link

https://github.com/arleenchr/MusicRecognition

References

[1] Gilleland, Michael, Merriam Park Software, "Levensthein Distance, in Three Flavors." University of Pittsburgh. https://people.cs.pitt.edu/~kirk/cs1501/Pruhs/Spring2006/assignments/editdistance/Levenshtein%20Distance.htm (Diakses 22 Mei 2023)

[2] Michigan Technological University. "Physics of Music - Notes" Michigan Technological University. https://pages.mtu.edu/~suits/NoteFreqCalcs.html (Diakses 22 Mei 2023)

[3] Rinaldi Munir, “Pencocokan String (String/Pattern Matching).” Homepage Rinaldi Munir. Sekolah Teknik Elektro dan Informatika (STEI) ITB. https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Pencocokan-string-2021.pdf (Diakses 22 Mei 2023)

[4] T. Rainer, G. Panos, V. Remco C., W. Frans, Ren´e van Oostrum, “Using Transportation Distances for Measuring Melodic Similarity.” Institute of Information and Computing Sciences University of Utrecht. https://www.researchgate.net/publication/46648869_Using_Transportation_Distances_for_Measuring_Melodic_Similarity (Diakses 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

Arleen Chrysantha Gunardi (NIM 13521059)