

TUGAS BESAR 3

IF2211 STRATEGI ALGORITMA

SEMESTER II TAHUN 2023/2024

**Pemanfaatan *Pattern Matching* dalam Membangun Sistem
Deteksi Individu Berbasis Biometrik Melalui Citra Sidik Jari**



OLEH:

Akbar Al Fattah 13522036

Diero Arga Purnama 13522056

Andhita Naura Hariyanto 13522060

**PROGRAM STUDI TEKNIK INFORMATIKA
SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG**

2024

BAB I

DESKRIPSI TUGAS

Pada Tugas Besar ini, buatlah sebuah sistem yang dapat melakukan identifikasi individu berbasis biometrik dengan menggunakan sidik jari dengan detail sebagai berikut.

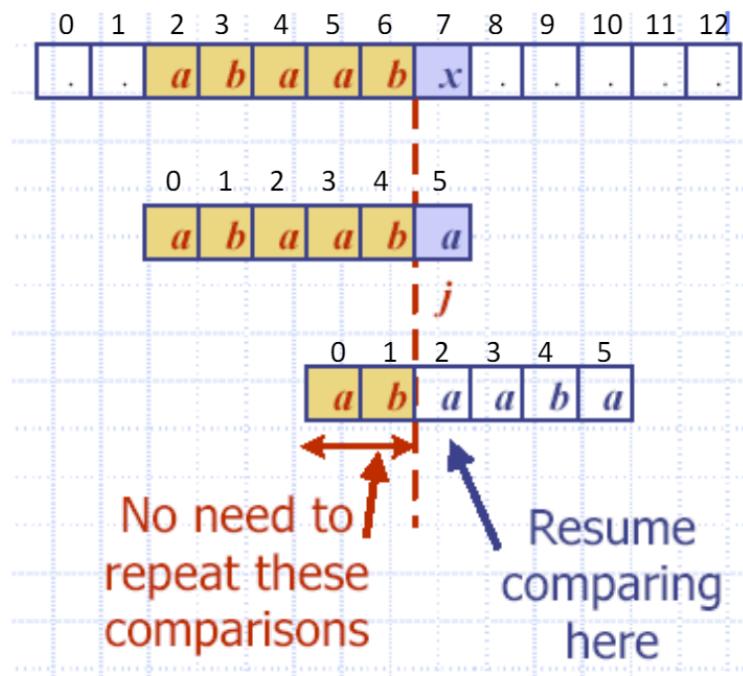
1. Sistem dibangun dalam bahasa C# yang mengimplementasikan algoritma KMP, BM, dan Regular Expression dalam mencocokkan sidik jari dengan biodata yang berpotensi rusak.
2. Program dapat memiliki basis data SQL yang telah mencocokkan berkas citra sidik jari yang telah ada dengan seorang pribadi. Basis data yang digunakan dibebaskan asalkan bukan No-SQL (sebagai contoh, MySQL, PostgreSQL, SQLite).
3. Program dapat menerima masukan sebuah citra sidik jari yang ingin dicocokkan. Apabila citra tersebut memiliki kecocokan di atas batas tertentu (silakan lakukan tuning nilai yang tepat) dengan citra yang sudah ada, maka tunjukkan biodata orang tersebut. Apabila di bawah nilai yang telah ditentukan tersebut, memunculkan pesan bahwa sidik jari tidak dikenali.
4. Program memiliki keluaran yang minimal mengandung seluruh data yang terdapat pada contoh antarmuka pada bagian penggunaan program.
5. Pengguna dapat memilih algoritma yang ingin digunakan antara KMP atau BM.
6. Biodata yang ditampilkan harus biodata yang memiliki nama yang benar (gunakan Regex untuk memperbaiki nama yang rusak dan gunakan KMP atau BM untuk mencari orang yang paling sesuai).
7. Program memiliki antarmuka yang user-friendly. Anda juga dapat menambahkan fitur lain untuk menunjang program yang Anda buat (unsur kreativitas).

BAB II

LANDASAN TEORI

2.1. Algoritma Knuth-Morris-Pratt

Algoritma *Knuth-Morris-Pratt* (KMP) merupakan sebuah algoritma pencocokan *string* yang digunakan untuk mencari sebuah pola dalam teks yang melakukan pencocokan dengan arah *left-to-right*. Berbeda dengan algoritma *brute force* yang mengecek setiap kemungkinan, algoritma KMP menggunakan informasi dari kegagalan sebelumnya untuk menghindari pengujian yang tidak diperlukan, sehingga algoritma ini akan lebih cepat jika dibandingkan dengan algoritma *brute force*.

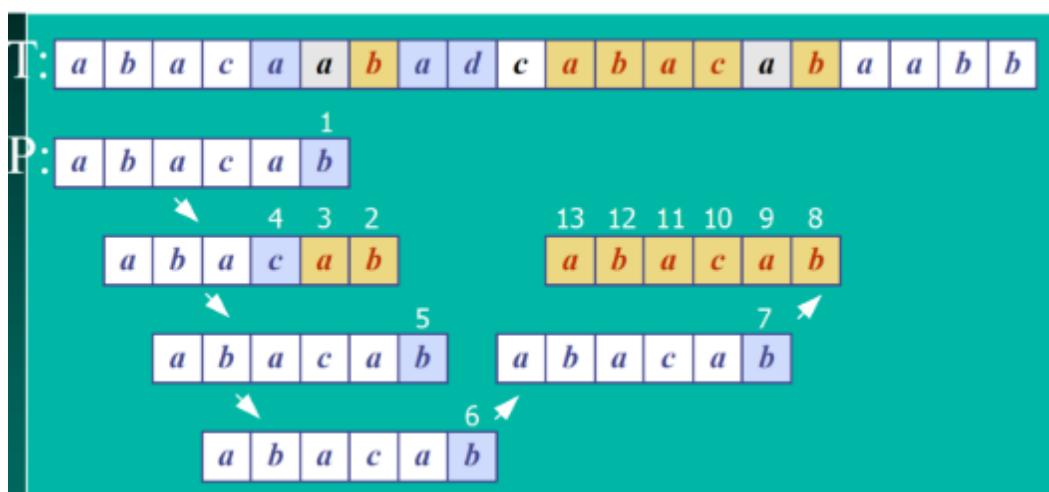


Gambar 2.1 Algoritma KMP

Algoritma ini menggunakan sebuah *border function* untuk mempercepat pencarinya, fungsi ini mencatat sejauh apa pola cocok dengan dirinya sendiri pada semua posisi dalam pola. Ketika terjadi *mismatch* dalam pencarian, algoritma menggunakan fungsi ini untuk menentukan besar lompatan yang akan diambil.

2.2. Algoritma Boyer-Moore

Algoritma Boyer-Moore (BM) merupakan sebuah algoritma pencocokan *string* yang digunakan untuk mencari sebuah pola dalam teks. Algoritma ini menggunakan dua teknik untuk mempercepat pencarinya, yaitu teknik *looking-glass* yaitu pencarian pola dimulai dari akhir teks, dan teknik *character-jump*. Algoritma ini juga menggunakan *last occurrence function*, yang digunakan untuk melakukan mencatat index terakhir kemunculan sebuah huruf dalam pola. Algoritma ini menggunakan *last occurrence function* bersama dengan teknik *character-jump* untuk menentukan besar lompatan yang akan diambil oleh algoritma ketika terjadinya sebuah *mismatch* dalam pencarian.



Gambar 2.2 Algoritma BM

2.3. Regular Expression

Regular expression (regex) adalah notasi standar yang mendeskripsikan suatu pola (pattern) berupa urutan karakter atau string. Regex digunakan untuk pencocokan string (string matching) dengan efisien.

/ (ha)+ | (he)+ | (hi)+ /g

Test String

haha hehehehe hoho hihi

Gambar 2.3 Regex (Sumber:

<https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2022-2023/String-Matching-dengan-Regex-2019.pdf>

2.4. *Hamming Distance*

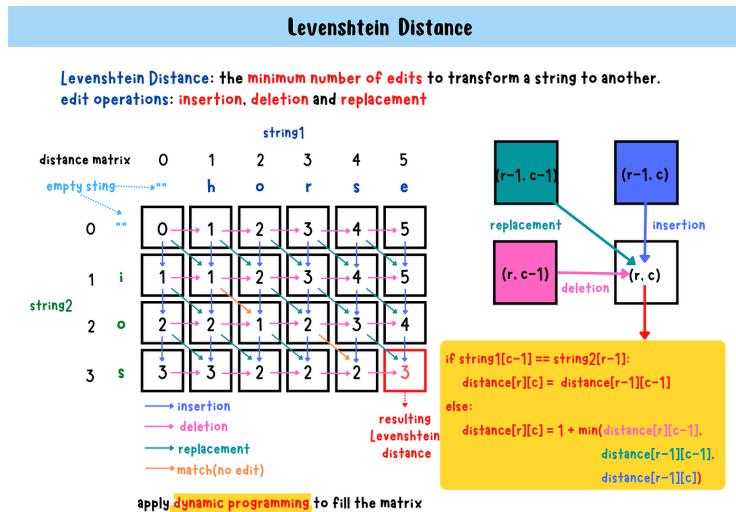
Teknik yang digunakan untuk mengukur persentase kemiripan dalam program ini adalah *Hamming Distance*. *Hamming Distance* adalah sebuah metode pengukuran perbedaan antara dua *string* dengan panjang yang sama. Nilai *Hamming Distance* antara dua *string* adalah jumlah karakter yang berbeda yang terletak pada posisi yang sama.



Gambar 2.4 Hamming Distance

2.5. *Levenhstein Distance*

Levenshtein distance digunakan untuk mengukur perbedaan dua string dengan menghitung jumlah minimum operasi yang diperlukan untuk mentransformasi suatu string menjadi bentuk string lainnya dengan menggunakan bantuan struktur data matriks. Operasi-operasi yang dipertimbangkan jumlahnya untuk menentukan perbedaan kedua string adalah operasi penyisipan (*insertion*), penghapusan (*deletion*), atau penukaran (*substitution*). Levenshtein distance dapat digunakan dalam mendeteksi kemiripan antara dua string yang berpotensi melakukan tindak plagiarisme. Nilai kemiripan antara dua string yang dapat ditentukan Levenshtein distance membuat Levenshtein distance kerap digunakan untuk beberapa keperluan seperti deteksi plagiarisme.



Gambar 2.5 Hamming Distance (Sumber :

<https://yuminlee2.medium.com/levenshtein-distance-1080038a4d9>)

2.6. Tentang Aplikasi Desktop

FINGFIND adalah aplikasi desktop yang bertujuan untuk mendapatkan data seperti NIK, nama, tempat dan tanggal lahir, jenis kelamin, golongan darah, alamat, agama, status perkawinan, pekerjaan, kewarganegaraan, dan sidik jari dari pemiliknya. Cara menggunakan aplikasi ini adalah pengguna akan memberikan masukan berupa gambar sidik jari dalam format .bmp, kemudian pengguna akan memilih algoritma yang digunakan di antara **Knuth-Morris-Pratt (KMP)** atau **Boyer-Moore (BM)**. Setelah pengguna memilih algoritma, pengguna akan menekan tombol cari dan program akan melakukan pencarian data dan sidik jari berdasarkan sidik jari yang diinput. Jika tidak ada sidik jari yang memiliki tingkat kemiripan di atas 0,6, program akan menganggap tidak ada sidik jari yang ditemukan di dalam database. Jika sidik jari ditemukan (ada sidik jari yang memiliki tingkat kemiripan di atas 0,6), seluruh data yang berkaitan dengan sidik jari tersebut akan ditampilkan di layar aplikasi.

BAB III

ANALISIS PEMECAHAN MASALAH

3.1 Langkah Penyelesaian Masalah

3.1.1. Pencocokkan Pola Sidik Jari

Masalah yang ingin diselesaikan oleh program adalah mencari entri dalam *database* yang memiliki sidik jari dengan tingkat kemiripan yang paling tinggi dengan sidik jari yang dimasukkan oleh pengguna. Pertama-tama, gambar tersebut akan diubah menjadi gambar hitam putih, setelah itu, akan dibuat sebuah representasi gambar dalam bentuk larik yang berisi 1 (*pixel* hitam) dan 0 (*pixel* putih). Setelah didapatkan larik tersebut, tiap 32 elemen dalam larik biner akan diubah menjadi 4 karakter ASCII dan disimpan dalam sebuah larik baru. Setelah didapatkan larik *string* ASCII yang merepresentasikan gambar sidik jari, akan diambil satu-per-satu elemen dari larik tersebut (32 *pixel*) dan dilakukan pencarian pola tersebut di dalam representasi ASCII sidik jari yang ada di dalam *database*, alasan pemilihan ukuran 32 *pixel* adalah karena ukuran tersebut dapat diubah dengan mudah menjadi 4 karakter ASCII. Setelah dilakukan pencarian, akan dikembalikan sidik jari dengan tingkat kemiripan tertinggi jika terdapat sidik jari dengan tingkat kemiripan lebih dari 0,6 alasan pemilihan 0,6 sebagai *threshold* tingkat kemiripan adalah untuk memberikan fleksibilitas kepada sistem untuk mengidentifikasi variasi-variasi dari sebuah sidik jari tanpa kehilangan terlalu banyak akurasi.

Langkah-langkah penyelesaian solusi dengan menggunakan algoritma KMP adalah sebagai berikut:

- 1) Lakukan *pre-processing* terhadap pola untuk mendapatkan *border function* atau $b(k)$ yaitu ukuran dari *prefix* terbesar dari $pola[0..k]$ yang juga merupakan *suffix* dari $pola[1..k]$.
- 2) Mulai pencocokan *string* dari ujung kiri.
- 3) Gunakan dua variabel, i yaitu indeks untuk *string*, dan j yaitu indeks untuk pola.
- 4) Bandingkan karakter $string[i]$ dan $pola[j]$.
- 5) Jika kedua karakter adalah sama, tambahkan nilai i dan j dengan satu.
- 6) Jika nilai j mencapai panjang pola dikurangi dengan satu, berarti terdapat pola dalam *string* tersebut, masukkan indeks awal pola tersebut dan simpan dalam sebuah larik.

- 7) Jika terjadi *mismatch* antara karakter *string*[*i*] dengan *pola*[*j*] dan nilai *j* lebih dari nol, ubah nilai *j* menjadi $b(j - 1)$, jika nilai *j* adalah nol, tambah nilai *i* dengan satu.
- 8) Ulangi proses ini hingga nilai *i* mencapai panjang *string* dikurang satu.
- 9) Kembalikan larik yang berisi indeks awal pola-pola yang berada di dalam *string*.

Langkah pencocokan dengan menggunakan algoritma BM adalah sebagai berikut:

- 1) Buat sebuah *dictionary* yang berisi semua huruf dalam alfabet dan indeks kemunculan terakhirnya dalam pola, huruf yang tidak muncul dalam pola akan memiliki nilai -1.
- 2) Mulai pencocokan *string* dari ujung kanan.
- 3) Gunakan dua variabel, *shift* untuk menentukan besar lompatan, dan *idx* yaitu indeks untuk pola.
- 4) Selama nilai *idx* lebih dari sama dengan nol, bandingkan *pola*[*idx*] dan *string*[*shift* + *idx*].
- 5) Jika kedua karakter adalah sama, kurangi nilai *idx* dengan satu.
- 6) Jika nilai *idx* mencapai -1, berarti terdapat pola dalam *string*, masukkan indeks awal pola tersebut ke dalam sebuah larik.
- 7) Jika terjadi *mismatch* antara karakter *pattern*[*idx*] dengan *string*[*shift* + *idx*], tambahkan nilai *shift* dengan indeks terakhir kemunculan karakter *string*[*shift* + panjang pola] yang disimpan dalam *dictionary*.
- 8) Ulangi proses ini hingga nilai *shift* ditambah dengan panjang *pattern* melebihi panjang *string*.
- 9) Kembalikan larik yang berisi indeks awal pola-pola yang berada di dalam *string*.

Jika semua elemen dalam larik ditemukan di dalam representasi ASCII sidik jari dari *database*, program mengembalikan tingkat kemiripan 1 (*exact match*). Jika terdapat elemen yang tidak ditemukan dalam representasi ASCII sidik jari yang ada di dalam *database*, program akan mengembalikan nilai tingkat kemiripan yang dihitung menggunakan *Hamming Distance*.

3.1.2. Penanganan Data Korup

Masalah lainnya yang ingin diselesaikan oleh program adalah menangani kemungkinan terdapatnya data korup pada nilai atribut nama di tabel sidik_jari. Penanganan data korup dilakukan dengan memanfaatkan regular expression dan juga Levenshtein Distance. Pertama-tama, data nama yang didapat dari tabel sidik_jari akan diaplikasikan

fungsi Translate yang menerima nilai nama berupa string dan mengembalikan hasil translasi dari kakas bahasa alay yang melibatkan konversi angka ke huruf dan juga konversi huruf kapital ke huruf kecil. Kemudian, seluruh data nama dari tabel biodata yang telah dikonversi menjadi huruf kecil semua dan string hasil kembalian fungsi Translate akan dibandingkan jumlah katanya. Jika jumlah kata sama, akan diaplikasikan fungsi SimilarityRate yang membandingkan nilai kemiripan text dan pattern dengan LevenhsteinDistance. Apabila nilai kemiripan ($1 - \text{LevenhsteinDistance}/\text{TextLength}$) yang dihasilkan lebih besar atau sama dengan 0,6, fungsi SimilarityRate akan mengembalikan nilai boolean True yang berarti string nama dari tabel sidik_jari yang telah dikonversi dinilai mirip dengan string nama pada tabel biodata yang sedang ditelusuri. Nilai 0,6 dipilih sebagai threshold tingkat kemiripan karena dinilai memberikan fleksibilitas yang baik kepada sistem untuk mengidentifikasi string asli yang mirip dengan string korupnya setelah dilakukan eksplorasi pencarian hasil kemiripan string yang disimulasikan korup dengan string aslinya.

Langkah penanganan data korup adalah sebagai berikut:

- 1) Translasikan karakter-karakter angka yang ada pada atribut nama dari tabel biodata (string pattern) menjadi huruf dengan rincian data angka yang dikonversi menjadi huruf :
 - 0 (nol) menjadi huruf o
 - 1 (satu) menjadi huruf i
 - 2 (dua) menjadi huruf z
 - 3 (tiga) menjadi huruf e
 - 4 (empat) menjadi huruf a
 - 5 (lima) menjadi huruf s
 - 6 (enam) menjadi huruf g
- 2) Konversi seluruh huruf yang ada pada string pattern tersebut menjadi huruf kecil dengan bantuan fungsi toLower()
- 3) Konversi seluruh huruf pada string nama di tabel biodata (string text) yang sedang ditelusuri menjadi huruf kecil dengan bantuan fungsi toLower().
- 4) Bandingkan string pattern dan string text yang telah dilakukan konversi formatnya.
- 5) Apabila jumlah kata yang ada pada string text dan string pattern tidak sama, akan kedua string dinyatakan tidak mirip.

- 6) Apabila jumlah kata yang ada pada string text dan string pattern sama, akan dilakukan pemeriksaan apakah seluruh karakter yang ada pada pattern merupakan karakter yang juga terkandung pada string text.
- 7) Apabila ada karakter pada string pattern yang tidak terkandung dalam string text, kedua string akan dinyatakan tidak mirip.
- 8) Apabila seluruh karakter pada string pattern terkandung dalam string text, akan dilakukan pemeriksaan kemiripan dengan fungsi SimilarityRate yang mengimplementasikan penghitungan Levenhstein Distance.
- 9) Apabila fungsi SimilarityRate mengembalikan nilai boolean true, maka string text dinyatakan mirip dengan string pattern.

Algoritma Levenhstein Distance dan regular expression dapat dimanfaatkan untuk menangani adanya kemungkinan-kemungkinan korup pada nilai atribut nama di tabel sidik_jari sehingga data lengkap biodata bisa didapat pada pencarian sidik jari yang mirip.

3.2 Pemetaan Masalah menjadi Elemen-elemen Algoritma

Permasalahan-permasalahan yang ada adalah sebagai berikut:

- 1) Gambar sidik jari masukan.
- 2) Gambar sidik jari *database*.

Algoritma *Knuth-Morris-Pratt* (KMP) memiliki elemen-elemen berikut:

- 1) Pola.
- 2) Teks, tempat terjadinya pencarian pola.
- 3) *Border function*, larik yang menyimpan panjang *prefix* terpanjang yang juga merupakan *suffix* untuk setiap elemen dari pola, digunakan untuk menghindari perbandingan ulang setelah terjadi *mismatch*.

Sedangkan algoritma pencarian *Boyer-Moore* (BM) memiliki elemen-elemen berikut:

- 1) Pola.
- 2) Teks, tempat terjadinya pencarian pola.
- 3) *Last occurrence function*, larik yang berisi indeks kemunculan terakhir karakter dalam pola.

Sehingga dapat dipetakan permasalahan-permasalahan menjadi elemen-elemen algoritma KMP dan BM sebagai berikut:

No	Permasalahan	Elemen KMP	Elemen BM
1.	Gambar sidik jari masukan	Dipetakan menjadi pola yang akan di cari di dalam teks dan untuk mencari <i>border function</i> .	Dipetakan menjadi pola yang akan di cari di dalam teks dan untuk mencari <i>Last occurrence function</i> .
2.	Gambar sidik jari <i>database</i>	Dipetakan menjadi teks.	Dipetakan menjadi teks.

3.3 Fitur Fungsional & Arsitektur Aplikasi Desktop

Aplikasi yang kami buat memiliki fungsionalitas berikut:

1. Aplikasi dapat menerima masukan citra sidik jari dan mencari sidik jari dalam basis data dengan tingkat kemiripan yang tertinggi yang melebihi *threshold* 0,6 dan menampilkan biodata yang terkait dengan sidik jari tersebut.
2. Aplikasi dapat melakukan pencarian sidik jari menggunakan algoritma Knuth-Morris-Pratt (KMP) atau algoritma Boyer-Moore (BM).
3. Aplikasi dapat menampilkan waktu yang dibutuhkan untuk melakukan pencarian dan tingkat kemiripan dari sidik jari yang ditemukan.

Arsitektur aplikasi desktop ini terdiri atas 2 bagian utama, yaitu frontend dan backend.

Untuk bagian frontend dari aplikasi ini, kami menggunakan WPF. WPF menggunakan 2 komponen utama, yaitu XAML dan C#.

Untuk bagian backend, kami menggunakan bahasa C# dan komponen-komponen berikut:

1. .NET versi 8.0
2. Server MySQL versi 8.0.37
3. Framework SixLabors.ImageSharp 3.1.4 (untuk menghandle gambar)
4. Library MySql.Data versi 8.4.0
5. Library Dapper versi 2.1.35
6. System.Drawing.Common versi 8.0.6

3.4 Contoh Ilustrasi Kasus



Gambar 3.4.1 Contoh Ilustrasi Kasus

Alur keberjalanannya program adalah sebagai berikut:

- 1) Program mengonversi gambar masukan sidik jari ke dalam bentuk ASCII.
- 2) Program mengambil seluruh gambar sidik jari dari tabel sidik_jari dalam basis data.
- 3) Program mengonversi gambar sidik jari dari basis data ke dalam bentuk ASCII.
- 4) Program membandingkan gambar sidik jari masukan dengan seluruh gambar sidik jari dari database dan mencari sidik jari dengan tingkat kemiripan tertinggi yang lebih dari 0,6.
- 5) Program menggunakan regex untuk mencocokkan nama pada tabel sidik_jari dengan tabel biodata.
- 6) Program menampilkan biodata yang memiliki tingkat kemiripan tertinggi jika ada.

BAB IV

IMPLEMENTASI DAN PENGUJIAN

4.1 Spesifikasi Teknis Program

4.1.1 ConvertImage.cs:

No.	Struktur Data	Fungsi
1	class ImageConverter	Digunakan untuk mengkonversi sebuah gambar menjadi representasi ASCII-nya.
2	convertImage(string path)	Mengkonversi sebuah gambar menjadi array ASCII.
3	ConvertImageToBnW(Bitmap image)	Mengkonversi sebuah gambar menjadi gambar hitam putih.
4	ConvertBinaryToASCII(bool[,] binaryArr)	Mengkonversi larik biner dua dimensi menjadi list ASCII.

ConvertImage.cs

```
using System.Drawing;
using System.Text;

/**
 * Kelas untuk konversi gambar menjadi array ASCII.
 */
public class ImageConverter {
    /**
     * Konversi gambar menjadi array ASCII.
     *
     * @param {string} path - path ke gambar yang ingin dikonversi.
     * @param {string} filename - nama file gambar yang ingin dikonversi.
     * @returns {List<string>} Hasil konversi gambar menjadi array ASCII.
     */
    public static List<string> convertImage(string path)
    {
```

```

// Load gambar.
Bitmap image = new Bitmap(path);

// Konversi gambar menjadi gambar hitam putih.
Bitmap bnw = ConvertImageToBnW(image);

// Konversi gambar hitam putih menjadi array biner dua dimensi.
bool[,] binaryArr = ConvertBnWToBinary(bnw);

// Konversi array biner menjadi array ASCII.
List<string> asciiArr = ConvertBinaryToASCII(binaryArr);

return asciiArr;
}

/**
 * Konversi gambar menjadi gambar hitam putih.
 *
 * @param {Bitmap} image - Gambar yang ingin dikonversi.
 * @returns {Bitmap} Hasil konversi image menjadi gambar hitam putih.
 */
public static Bitmap ConvertImageToBnW(Bitmap image)
{
    Bitmap bnw = new Bitmap(image.Width, image.Height);
    for (int y = 0; y < image.Height; y++)
        for (int x = 0; x < image.Width; x++)
    {
        Color colorAwal = image.GetPixel(x, y);

        // Konversi ke grayscale menggunakan luminosity method.
        int grayScale = (int) (colorAwal.R * 0.21 +
colorAwal.G * 0.72 + colorAwal.B * 0.05);

        // Konversi ke hitam putih dg threshold 128 (0 -> hitam, 255 -> putih).
        bnw.SetPixel(x, y, grayScale < 128 ? Color.Black : Color.White);
    }
}

```

```

        return bnw;
    }

    /**
     * Konversi gambar hitam putih menjadi array biner dua dimensi.
     *
     * @param {Bitmap} bnw - Gambar hitam putih yang ingin dikonversi.
     * @returns {bool[,]}} Hasil konversi gambar menjadi array dua dimensi.
     */
    public static bool[,] ConvertBnWToBinary(Bitmap bnw)
    {
        bool[,] binaryArr = new bool[bnw.Height, bnw.Width];
        for (int y = 0; y < bnw.Height; y++)
            for (int x = 0; x < bnw.Width; x++)
                // Pixel hitam direpresentasikan dengan True pada array.
                binaryArr[y, x] = bnw.GetPixel(x, y).R == 0;
        return binaryArr;
    }

    /**
     * Konversi array biner dua dimensi menjadi list ASCII. Setiap elemen pada list ASCII merepresentasikan 32 elemen pada array biner.
     *
     * @param {bool[,]}} binaryArr - array biner dua dimensi yang ingin dikonversi.
     * @returns {List<string>} Hasil konversi array biner menjadi array ASCII.
     */
    public static List<string> ConvertBinaryToASCII(bool[,] binaryArr)
    {
        // Konversi list 32 karakter menjadi string ASCII.
        string convert(List<char> binaryGroup) {
            StringBuilder sb = new StringBuilder();

```

```

        sb.Append((char)Convert.ToInt32(new
string(binaryGroup.GetRange(0, 8).ToArray()), 2));
        sb.Append((char)Convert.ToInt32(new
string(binaryGroup.GetRange(8, 8).ToArray()), 2));
        sb.Append((char)Convert.ToInt32(new
string(binaryGroup.GetRange(16, 8).ToArray()), 2));
        sb.Append((char)Convert.ToInt32(new
string(binaryGroup.GetRange(24, 8).ToArray()), 2));

    return sb.ToString();
}

List<string> asciiArr = [];
for (int y = 0; y < binaryArr.GetLength(0); y++)
{
    List<char> binaryGroup = [];
    for (int x = 0; x < binaryArr.GetLength(1); x++)
    {
        if (binaryGroup.Count == 32)
        {
            asciiArr.Add(convert(binaryGroup));
            binaryGroup.Clear();
        }
        binaryGroup.Add(binaryArr[y, x] ? '1' : '0');
    }

    if (binaryGroup.Count > 0)
    {
        while (binaryGroup.Count != 32)
            binaryGroup.Add('0');
        asciiArr.Add(convert(binaryGroup));
        binaryGroup.Clear();
    }
}
return asciiArr;
}
}

```

4.1.2 PatternMatching.cs:

No.	Struktur Data	Fungsi
1	abstract class AlgoritmaPatternMatching	Menyediakan kerangka untuk algoritma KMP dan BM
2	Match(string text, string pattern)	Mencari pola <i>pattern</i> di dalam string <i>text</i> menggunakan algoritma KMP atau BM, mengembalikan larik integer yang berisi indeks awal pola dalam teks.
3	class KMP	Mendefinisikan metode abstrak Match pada kelas AlgoritmaPatternMatching dengan algoritma KMP.
4	calcBorderFunction(string pattern)	Mencari <i>border function</i> dari pattern.
5	class BM	Mendefinisikan metode abstrak Match pada kelas AlgoritmaPatternMatching dengan algoritma BM.
6	calcLastOccurrenceFunction(string pattern)	Mencari <i>last occurrence function</i> dari pattern.

PatternMatching.cs

```
/**
 * Kelas abstrak untuk algoritma pencocokan string.
 */
public abstract class AlgoritmaPatternMatching
{
    /**
     * Mencari pola dalam sebuah teks.
     *
     * @param {string} text - tempat dilakukannya pencarian pola.
     * @param {string} pattern - pola yang ingin dicari dalam text.
     * @returns {List<int>} array yang berisi indeks awal kemunculan pola dalam text.
    */
}
```

```

    */
    public abstract List<int> Match(string text, string pattern);
}

/***
 * Kelas untuk algoritma pencocokan string Knuth-Morris-Pratt
(KMP) .
*/
public class KMP : AlgoritmaPatternMatching {
    /**
     * Mencari border function dari pattern.
     *
     * @param {string} pattern - teks yang ingin diproses
menjadi border function.
     * @returns {List<int>} border function dari pattern.
     */
    public List<int> calcBorderFunction(string pattern)
    {
        List<int> borderFunction = new List<int>(pattern.Length);
        for (int k = 0; k < pattern.Length; k++)
            borderFunction.Add(0);

        int i = 1, j = 0;
        while (i < pattern.Length)
        {
            if (pattern[j] != pattern[i])
            {
                if (j > 0)
                    j = borderFunction[j - 1];
                else
                    i++;
            }
            else
                borderFunction[i++] = ++j;
        }
        return borderFunction;
    }

    /**
     * Mencari pola dalam sebuah teks menggunakan algoritma
Knuth-Morris-Pratt.

```

```

    *
    * @param   {string} text - tempat dilakukannya pencarian
pola.
    * @param   {string} pattern - pola yang ingin dicari dalam
text.
    * @returns {List<int>} array yang berisi indeks awal
kemunculan pola dalam text.
    */
public override List<int> Match(string text, string pattern)
{
    // Isi border function.
    List<int> borderFunction = calcBorderFunction(pattern);

    // Cari pattern dalam text.
    List<int> matches = new List<int>();
    int i = 0, j = 0;
    while (i < text.Length)
    {
        if (pattern[j] != text[i])
        {
            if (j > 0)
                j = borderFunction[j - 1];
            else
                i++;
        }
        else
        {
            if (j == pattern.Length - 1)
            {
                matches.Add(i - j);
                j = 0;
            }
            else
            {
                i++;
                j++;
            }
        }
    }
    return matches;
}

```

```

}

/**
 * Kelas untuk algoritma pencocokan string Boyer-Moore (BM) .
 */
public class BM : AlgoritmaPatternMatching
{
    /**
     * Mencari last occurrence function dari pattern.
     *
     * @param {string} pattern - teks yang ingin diproses
     * menjadi last occurrence function.
     * @returns {Dictionary<char, int>} last occurrence function
     * dari pattern.
     */
    public Dictionary<char, int> calcLastOccurrenceFunction(string
pattern)
    {
        Dictionary<char, int> lof = new Dictionary<char, int>();
        for (int i = 0; i < pattern.Length; i++)
            lof[pattern[i]] = i;
        return lof;
    }

    /**
     * Mencari pola dalam sebuah teks menggunakan algoritma
     * Boyer-Moore.
     *
     * @param {string} text - tempat dilakukannya pencarian
     * pola.
     * @param {string} pattern - pola yang ingin dicari dalam
     * text.
     * @returns {List<int>} array yang berisi indeks awal
     * kemunculan pola dalam text.
     */
    public override List<int> Match(string text, string pattern)
    {
        // Isi last occurrence function.
        Dictionary<char, int> charLastIdx =
calcLastOccurrenceFunction(pattern);

```

```

// Cari pattern dalam text.
List<int> matches = new List<int>();
int shift = 0;
while (shift + pattern.Length <= text.Length)
{
    int idx = pattern.Length - 1;
    while (idx >= 0 && (pattern[idx] == text[shift + idx]))
        idx--;

    int val;
    if (idx == -1)
    {
        matches.Add(shift);

        if (shift + pattern.Length < text.Length)
        {
            if (charLastIdx.TryGetValue(text[shift + pattern.Length], out val))
                shift += pattern.Length - val;
            else
                shift += pattern.Length + 1;
        }
    }
    else
    {
        if (charLastIdx.TryGetValue(text[shift + idx], out val))
            shift += idx - val > 1 ? idx - val : 1;
        else
            shift += idx + 1;
    }
}
return matches;
}
}

```

4.1.3 DatabaseManager.cs:

No.	Struktur Data	Fungsi
1	class DatabaseManager	Digunakan untuk mengelola pengambilan data dari basis data biodata yang digunakan oleh program.
2	getBiodataCount()	Mengembalikan jumlah rekord dalam tabel biodata yang ada di basis data.
3	getSidikJariCount()	Mengembalikan jumlah rekord dalam tabel sidik_jari yang ada di basis data.
4	getNameFromSidikJari(string berkas_citra)	Menerima alamat citra dalam format string dan mengembalikan nilai dari atribut nama (dalam format string) yang bersesuaian dengan masukan alamat citra di tabel sidik_jari.
5	getNIKFromName(string Name)	Menerima nama dalam format string dan mengembalikan nilai dari atribut NIK (dalam format string) yang bersesuaian dengan masukan nama di tabel biodata.
6	getNameFromNIK (string NIK)	Menerima NIK dalam format string dan mengembalikan nilai dari atribut nama (dalam format string) yang bersesuaian dengan masukan nama di tabel biodata.
7	getTempatLahirFromNIK (string NIK)	Menerima NIK dalam format string dan mengembalikan nilai dari atribut tempat lahir (dalam format string) yang bersesuaian dengan masukan NIK di tabel biodata.
8	getTanggalLahirFromNIK(string NIK)	Menerima NIK dalam format string dan mengembalikan nilai dari atribut tanggal lahir (dalam format string) yang bersesuaian dengan masukan NIK di tabel biodata.
9	getJenisKelaminFromNIK (string NIK)	Menerima NIK dalam format string dan mengembalikan nilai dari atribut jenis kelamin (dalam format

		string) yang bersesuaian dengan masukan NIK di tabel biodata.
10	getGolonganDarahFromNIK (string NIK)	Menerima NIK dalam format string dan mengembalikan nilai dari atribut golongan darah (dalam format string) yang bersesuaian dengan masukan NIK di tabel biodata.
11	getAlamatFromNIK (string NIK)	Menerima NIK dalam format string dan mengembalikan nilai dari atribut alamat (dalam format string) yang bersesuaian dengan masukan NIK di tabel biodata.
12	getAgamaFromNIK (string NIK)	Menerima NIK dalam format string dan mengembalikan nilai dari atribut agama (dalam format string) yang bersesuaian dengan masukan NIK di tabel biodata.
13	getStatusPerkawinanFromNIK (string NIK)	Menerima NIK dalam format string dan mengembalikan nilai dari atribut status perkawinan (dalam format string) yang bersesuaian dengan masukan NIK di tabel biodata.
14	getPekerjaanFromNIK (string NIK)	Menerima NIK dalam format string dan mengembalikan nilai dari atribut pekerjaan (dalam format string) yang bersesuaian dengan masukan NIK di tabel biodata.
15	getKewarganegaraanFromNIK (string NIK)	Menerima NIK dalam format string dan mengembalikan nilai dari atribut kewarganegaraan (dalam format string) yang bersesuaian dengan masukan NIK di tabel biodata.
16	printDataFromName(string Name)	Menampilkan nilai dari seluruh atribut dalam tabel biodata berdasarkan masukan string nama yang bersesuaian dengan nilai dari atribut nama pada tabel biodata.
17	printDataFromNIK(string NIK)	Menampilkan nilai dari seluruh

		atribut dalam tabel biodata berdasarkan masukan string NIK yang bersesuaian dengan nilai dari atribut NIK pada tabel biodata.
18	getAllSidikJari()	Mengembalikan List of string yang berisikan seluruh alamat berkas citra yang didapat dari nilai atribut berkas_citra pada seluruh rekord tabel sidik_jari yang ada pada basis data.
19	getAllNamaFromBiodata()	Mengembalikan List of string yang berisikan seluruh nama yang didapat dari nilai atribut nama pada seluruh rekord tabel biodata yang ada pada basis data.
20	insertDataToSidikJari()	Menambahkan rekord baru pada tabel sidik_jari dengan masukan yang bersesuaian.
21	insertDataToBiodata()	Menambahkan rekord baru pada tabel biodata dengan masukan yang bersesuaian.

DatabaseManager.cs

```

using System;
using Dapper;
using MySql.Data.MySqlClient;

public class DatabaseManager
{
    private readonly string _connectionString;

    public DatabaseManager(string connectionString)
    {
        _connectionString = connectionString;
    }

    public int getBiodataCount()
    {
        using (var connection = new

```

```

MySqlConnection(_connectionString))
{
    connection.Open();
    var users = connection.Query("SELECT * FROM biodata");
    return users.Count();
}

public int getSidikJariCount()
{
    using (var connection = new
MySqlConnection(_connectionString))
    {
        connection.Open();
        var users = connection.Query("SELECT * FROM
sidik_jari");
        return users.Count();
    }
}

public string getNameFromSidikJari(string berkas_citra)
{
    using (var connection = new
MySqlConnection(_connectionString))
    {
        connection.Open();
        var users = connection.Query("SELECT * from sidik_jari
WHERE berkas_citra = @berkas_citra", new { berkas_citra });
        if (users.Count() == 0)
        {
            return "No data found with the inserted
fingerprint";
        }
        return users.First().nama;
    }
}

public string getNIKFromName(string Name)
{
    using (var connection = new

```

```

MySqlConnection(_connectionString))
{
    connection.Open();
    var users = connection.Query("SELECT * from biodata
WHERE nama = @nama", new { nama = Name });
    if (users.Count() == 0)
    {
        return "No data found with the inserted name";
    }
    return users.First().NIK;
}

public string getNameFromNIK (string NIK) {
    using (var connection = new
MySqlConnection(_connectionString))
    {
        connection.Open();
        var users = connection.Query("SELECT * from biodata
WHERE NIK = @NIK", new { NIK });
        if (users.Count() == 0)
        {
            return "No data found with the inserted NIK";
        }
        return users.First().nama;
    }
}

public string getTempatLahirFromNIK (string NIK) {
    using (var connection = new
MySqlConnection(_connectionString))
    {
        connection.Open();
        var users = connection.Query("SELECT * from biodata
WHERE NIK = @NIK", new { NIK });
        if (users.Count() == 0)
        {
            return "No data found with the inserted NIK";
        }
        return users.First().tempat_lahir;
    }
}

```

```

    }

    public string getTanggalLahirFromNIK(string NIK)
    {
        using (var connection = new
MySqlConnection(_connectionString))
        {
            connection.Open();
            var result =
connection.QueryFirstOrDefault<DateTime?>(
                "SELECT tanggal_lahir FROM biodata WHERE NIK =
@NIK",
                new { NIK }
            );

            if (result == null)
            {
                return "No data found with the inserted NIK";
            }

            // ubah date type tanggal lahir ke string
            return result.Value.ToString("yyyy-MM-dd");
        }
    }

    public string getJenisKelaminFromNIK (string NIK) {
        using (var connection = new
MySqlConnection(_connectionString))
        {
            connection.Open();
            var users = connection.Query("SELECT * from biodata
WHERE NIK = @NIK", new { NIK });
            if (users.Count() == 0)
            {
                return "No data found with the inserted NIK";
            }
            return users.First().jenis_kelamin;
        }
    }
}

```

```

        public string getGolonganDarahFromNIK (string NIK) {
            using (var connection = new
MySqlConnection(_connectionString))
{
            connection.Open();
            var users = connection.Query("SELECT * from biodata
WHERE NIK = @NIK", new { NIK });
            if (users.Count() == 0)
            {
                return "No data found with the inserted NIK";
            }
            return users.First().golongan_darah;
        }
    }

        public string getAlamatFromNIK (string NIK) {
            using (var connection = new
MySqlConnection(_connectionString))
{
            connection.Open();
            var users = connection.Query("SELECT * from biodata
WHERE NIK = @NIK", new { NIK });
            if (users.Count() == 0)
            {
                return "No data found with the inserted NIK";
            }
            return users.First().alamat;
        }
    }

        public string getAgamaFromNIK (string NIK) {
            using (var connection = new
MySqlConnection(_connectionString))
{
            connection.Open();
            var users = connection.Query("SELECT * from biodata
WHERE NIK = @NIK", new { NIK });
            if (users.Count() == 0)
            {
                return "No data found with the inserted NIK";
            }
}

```

```

        return users.First() .agama;
    }

}

public string getStatusPerkawinanFromNIK (string NIK) {
    using (var connection = new
MySqlConnection(_connectionString))
    {
        connection.Open();
        var users = connection.Query("SELECT * from biodata
WHERE NIK = @NIK", new { NIK });
        if (users.Count() == 0)
        {
            return "No data found with the inserted NIK";
        }
        return users.First() .status_perkawinan;
    }
}

public string getPekerjaanFromNIK (string NIK) {
    using (var connection = new
MySqlConnection(_connectionString))
    {
        connection.Open();
        var users = connection.Query("SELECT * from biodata
WHERE NIK = @NIK", new { NIK });
        if (users.Count() == 0)
        {
            return "No data found with the inserted NIK";
        }
        return users.First() .pekerjaan;
    }
}

public string getKewarganegaraanFromNIK (string NIK) {
    using (var connection = new
MySqlConnection(_connectionString))
    {
        connection.Open();
        var users = connection.Query("SELECT * from biodata
WHERE NIK = @NIK", new { NIK });

```

```

        if (users.Count() == 0)
        {
            return "No data found with the inserted NIK";
        }
        return users.First().kewarganegaraan;
    }

}

public void printDataFromName(string Name)
{
    using (var connection = new
 MySqlConnection(_connectionString))
    {
        connection.Open();
        var users = connection.Query("SELECT * FROM biodata
WHERE nama = @nama", new { nama = Name });
        foreach (var user in users)
        {
            Console.WriteLine(user.NIK);
            Console.WriteLine(user.nama);
            Console.WriteLine(user.tempat_lahir);
            Console.WriteLine(user.tanggal_lahir);
            Console.WriteLine(user.jenis_kelamin);
            Console.WriteLine(user.golongan_darah);
            Console.WriteLine(user.alamat);
            Console.WriteLine(user.agama);
            Console.WriteLine(user.status_perkawinan);
            Console.WriteLine(user.pekerjaan);
            Console.WriteLine(user.kewarganegaraan);
        }
    }
}

public void printDataFromNIK(string NIK)
{
    using (var connection = new
 MySqlConnection(_connectionString))
    {
        connection.Open();
        var users = connection.Query("SELECT * FROM biodata
WHERE NIK = @NIK", new { NIK });
        foreach (var user in users)
        {
            Console.WriteLine(user.NIK);
            Console.WriteLine(user.nama);
            Console.WriteLine(user.tempat_lahir);
            Console.WriteLine(user.tanggal_lahir);
            Console.WriteLine(user.jenis_kelamin);
            Console.WriteLine(user.golongan_darah);
            Console.WriteLine(user.alamat);
            Console.WriteLine(user.agama);
            Console.WriteLine(user.status_perkawinan);
            Console.WriteLine(user.pekerjaan);
            Console.WriteLine(user.kewarganegaraan);
        }
    }
}

```

```

WHERE NIK = @NIK", new { NIK });
        foreach (var user in users)
        {
            Console.WriteLine(user.NIK);
            Console.WriteLine(user.nama);
            Console.WriteLine(user.tempat_lahir);
            Console.WriteLine(user.tanggal_lahir);
            Console.WriteLine(user.jenis_kelamin);
            Console.WriteLine(user.golongan_darah);
            Console.WriteLine(user.alamat);
            Console.WriteLine(user.agama);
            Console.WriteLine(user.status_perkawinan);
            Console.WriteLine(user.pekerjaan);
            Console.WriteLine(user.kewarganegaraan);
        }
    }

}

public List<string> getAllSidikJari()
{
    using (var connection = new
 MySqlConnection(_connectionString))
    {
        connection.Open();
        var users = connection.Query("SELECT * FROM
sidik_jari");
        List<string> result = new List<string>();
        foreach (var user in users)
        {
            result.Add(user.berkas_citra);
        }
        return result;
    }
}

public List<string> getAllNamaFromBiodata()
{
    using (var connection = new
 MySqlConnection(_connectionString))
    {

```

```

        connection.Open();
        var users = connection.Query("SELECT * FROM biodata");
        List<string> result = new List<string>();
        foreach (var user in users)
        {
            result.Add(user.nama);
        }
        return result;
    }

    public void insertDataToSidikJari(){
        string insertDataQuery = @""
            INSERT INTO sidik_jari (berkas_citra, nama)
            VALUES (@FilePath, 'DUMMY_NAME');

        string filePath = @"test\1717073090681.bmp";

        using (var connection = new
MySqlConnection(_connectionString))
        {
            connection.Open();
            connection.Execute(insertDataQuery, new { FilePath =
filePath });
        }
    }

    public void insertDataToBiodata(){
        string insertDataQuery = @""
            INSERT INTO biodata (NIK, nama, tempat_lahir,
tanggal_lahir, jenis_kelamin, golongan_darah, alamat, agama,
status_perkawinan, pekerjaan, kewarganegaraan)
            VALUES ('0123456701234567', 'DUMMY_NAME',
'DUMMY_PLACE_OF_BIRTH', '2000-01-01', 'Laki-Laki', 'O',
'DUMMY_ADDRESS', 'DUMMY_RELIGION', 'Belum Menikah', 'DUMMY',
'DUMMY') ";
        using (var connection = new
MySqlConnection(_connectionString))
        {
            connection.Open();

```

```

        connection.ExecuteNonQuery();
    }

}
}

```

4.1.4 PencocokSidikJari.cs:

No.	Struktur Data	Fungsi
1	class PencocokSidikJari	Digunakan untuk melakukan pencocokan antara sidik jari masukan dengan sidik jari dari database.
2	Cocok(string path_sidik_jari_db, AlgoritmaPatternMatching algoritma)	Melakukan pencocokan antara sidik jari masukan dengan salah satu sidik jari dalam database dan mengembalikan tingkat kemiripannya.
3	MulaiPencocokan(string input, List<string> db, AlgoritmaPatternMatching algoritma)	Melakukan pencocokan antara sidik jari masukan dengan seluruh sidik jari dalam database, mengembalikan pasangan sidik jari dengan tingkat kemiripan dengan nilai tingkat kemiripan tertinggi.
4	HammingDistance(string t1, string t2)	Menghitung nilai <i>Hamming Distance</i> antara dua string.
5	HitungTingkatKemiripan(string t1, string t2)	Mengembalikan tingkat kemiripan antara dua string.

PencocokSidikJari.cs

```

/**
 * Kelas untuk melakukan pencocokan sidik jari.
 */
public class PencocokSidikJari
{
    private static List<string> sidik_jari_masukan = new
List<string>();

```

```

    private static List<string> list_sidik_jari_db = new
List<string>();

    /**
     * Melakukan pencocokan antara sidik jari masukan dengan
salah satu sidik jari dalam database.
     *
     * @param {string} path_sidik_jari_db - path menuju ke
sidik jari dalam database
     * @param {AlgoritmaPatternMatching} algoritma - algoritma
yang digunakan untuk pencocokan sidik jari.
     * @returns {double} tingkat kemiripan kedua sidik jari.
     */
    public static double Cocok(string path_sidik_jari_db,
AlgoritmaPatternMatching algoritma)
{
    // Konversi citra sidik jari menjadi ASCII.
    List<string> sidik_jari_db =
ImageConverter.convertImage(path_sidik_jari_db);
    string sidik_jari_db_str = string.Join("", sidik_jari_db);
    string sidik_jari_masukan_str = string.Join("", sidik_jari_masukan);

    // Lempar exception jika panjang string berbeda karena
perhitungan tingkat kemiripan
    // menggunakan hamming distance.
    if (sidik_jari_db_str.Length !=
sidik_jari_masukan_str.Length)
        throw new Exception("Ukuran sidik jari berbeda!");

    // Cari tingkat kemiripan kedua sidik jari, jika "exact
match" maka tingkat kemiripan
    // adalah 1, jika bukan exact match, maka akan dihitung
menggunakan hamming distance.
    double tingkatKemiripan = 1;
    for (int i = 0; i < sidik_jari_masukan.Count &&
tingkatKemiripan == 1; i++)
        if (algoritma.Match(sidik_jari_db_str,
sidik_jari_masukan[i]).Count == 0)
            tingkatKemiripan =
HitungTingkatKemiripan(sidik_jari_db_str, sidik_jari_masukan_str);
}

```

```

        return tingkatKemiripan;
    }

    /**
     * Melakukan pencocokan antara sidik jari masukan dengan
     seluruh sidik jari dari database.
     *
     * @param {string} input - path menuju ke file sidik jari
     masukan.
     *
     * @param {List<string>} db - list yang berisi path file
     sidik jari dari database.
     *
     * @param {AlgoritmaPatternMatching} algoritma - algoritma
     yang digunakan untuk pencocokan sidik jari.
     *
     * @returns {KeyValuePair<string, double>} pasangan sidik
     jari dengan tingkat kemiripan dengan nilai
     *
                                         tingkat kemiripan
yang paling tinggi.
     */
    public static KeyValuePair<string, double>
MulaiPencocokan(string input, List<string> db,
AlgoritmaPatternMatching algoritma)
{
    // Inisialisasi
    sidik_jari_masukan = ImageConverter.convertImage(input);
    list_sidik_jari_db = new List<string>(db);

    // Bandingin sidik jari masukan ke semua sidik jari dalam
    database
    Dictionary<string, double> hasil_akhir = new
Dictionary<string, double>();
    foreach (string path_sidik_jari_db in list_sidik_jari_db)
        hasil_akhir.Add(path_sidik_jari_db,
Cocok(path_sidik_jari_db, algoritma));

    // Cari tingkat kemiripan tertinggi
    var maxKvp = hasil_akhir.FirstOrDefault();
    foreach (var kvp in hasil_akhir)
    {
        double tingkatKemiripan = kvp.Value;
        if (tingkatKemiripan > maxKvp.Value)

```

```

        maxKvp = kvp;
    }

    return maxKvp;
}

// ----- Tingkat Kemiripan -----

/**
 * Menghitung nilai hamming distance antara dua string.
 * Berasumsi bahwa kedua string memiliki panjang sama.
 *
 * @param {string} t1 - string pertama.
 * @param {string} t2 - string kedua.
 * @returns {int} nilai hamming distance t1 dengan t2.
 */
public static int HammingDistance(string t1, string t2)
{
    if (t1 == t2)
        return 0;

    int cnt = 0;
    for (int i = 0; i < t1.Length; i++)
        if (t1[i] != t2[i])
            cnt++;

    return cnt;
}

/**
 * Menghitung tingkat kemiripan antara dua string.
 *
 * @param {string} t1 - string pertama.
 * @param {string} t2 - string kedua.
 * @returns {int} tingkat kemiripan t1 dengan t2.
 */
public static double HitungTingkatKemiripan(string t1, string
t2) { return 1 - ((double) HammingDistance(t1, t2) / t2.Length); }
}

```

4.1.5 Regex.cs

No.	Struktur Data	Fungsi
1	class Regex	Digunakan untuk melakukan pencarian data nama yang paling sesuai antara nilai atribut nama dari tabel sidik_jari dengan nilai atribut nama dari tabel biodata.
2	Translate(string input)	Menerima input string untuk dilakukan konversi dari bahasa alay menjadi string yang bersesuaian dan mengembalikan string hasil konversi.
3	MatchName(string text, string pattern)	Menerima input string text dan string pattern untuk dibandingkan kemiripannya kemudian mengembalikan nilai boolean true jika pattern sesuai dengan text dan false jika pattern tidak sesuai dengan text.
4	ContainsAllAlphabets(string text, string pattern)	Menerima input string text dan string pattern untuk diperiksa apakah text mengandung karakter yang dimiliki oleh pattern kemudian mengembalikan nilai boolean true jika text memiliki semua karakter yang dimiliki pattern dan false jika ada karakter pattern yang tidak dimiliki oleh text.
5	ResultText(List<string> dataname, string pattern)	Menerima input list of string berisikan seluruh nama dan string pattern, melakukan operasi MatchName untuk seluruh elemen pada list of string terhadap pattern masukan, dan mengembalikan string nama yang paling sesuai dengan string pattern.
6	isWordLengthSame(string text, string pattern)	Menerima input dua string kemudian mengecek panjang kedua string dan mengembalikan nilai boolean true jika panjang kedua string masukan sama atau false jika panjang kedua string masukan berbeda.

7	LevenhsteinDistance(string text1, string text2)	Menerima dua inputan string dan mengembalikan nilai double yang merepresentasikan jarak minimum antara kedua string dengan menggunakan algoritma <i>Levenhstein Distance</i> .
8	SimilarityRate(string text1, string text2)	Menerima dua inputan string dan mengembalikan nilai boolean true jika setelah diaplikasikan <i>Levenhstein distance</i> , perbandingan jarak hasil <i>Levenhstein distance</i> dengan panjang text terpanjangnya lebih besar dari atau sama dengan 0,6 serta false jika lebih kecil dari 0,6.

Regex.cs

```

using System;
using System.Text.RegularExpressions;

public class Regex
{
    public static string Translate(string input)
    {
        string result = input;
        result = result.Replace("4", "a");
        result = result.Replace("2", "z");
        result = result.Replace("1", "i");
        result = result.Replace("3", "e");
        result = result.Replace("6", "g");
        result = result.Replace("0", "o");
        result = result.Replace("5", "s");

        result = result.ToLower();

        return result;
    }

    public static double LevenshteinDistance (string text1, string
text2)
}

```

```

{
    int n = text1.Length;
    int m = text2.Length;
    int[,] dp = new int[n + 1, m + 1];

    for (int i = 0; i <= n; i++)
    {
        for (int j = 0; j <= m; j++)
        {
            if (i == 0)
            {
                dp[i, j] = j;
            }
            else if (j == 0)
            {
                dp[i, j] = i;
            }
            else if (text1[i - 1] == text2[j - 1])
            {
                dp[i, j] = dp[i - 1, j - 1];
            }
            else
            {
                dp[i, j] = 1 + Math.Min(dp[i, j - 1],
Math.Min(dp[i - 1, j], dp[i - 1, j - 1]));
            }
        }
    }

    return dp[n, m];
}

public static bool SimilarityRate(string text1, string text2)
{
    double distance = LevenshteinDistance(text1, text2);
    double maxLength = Math.Max(text1.Length, text2.Length);
    double rate = 1 - distance / maxLength;

    Console.WriteLine(rate);
    return rate >= 0.6;
}

```

```

public static bool MatchName(string text, string pattern)
{
    string textCompared = text.ToLower();
    string patternCompared = Translate(pattern);
    if (!isWordLengthSame(text, pattern))
    {
        return false;
    }
    else
    {
        if (ContainsAllAlphabets(textCompared,
patternCompared))
        {
            return SimilarityRate(textCompared,
patternCompared);
        }
        else
        {
            return false;
        }
    }
}

private static bool ContainsAllAlphabets(string text, string
pattern)
{
    foreach (char letter in pattern)
    {
        if (!text.Contains(letter.ToString()))
        {
            return false;
        }
    }
    return true;
}

public static string ResultText(List<string> dataname, string
pattern)
{
    foreach (string name in dataname)

```

```

    {
        if (MatchName(name, pattern))
        {
            return name;
        }
    }
    return "Not found";
}

private static bool isWordLengthSame(string text, string
pattern)
{
    static int CountWords(string text)
    {
        int wordCount = 0;
        bool isWord = false;

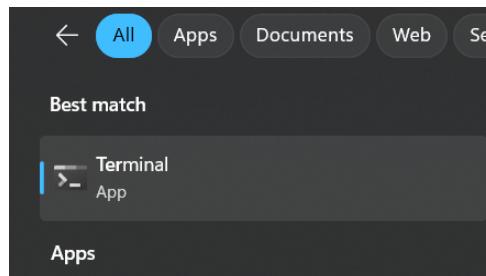
        foreach (char c in text)
        {
            if (char.IsLetterOrDigit(c))
            {
                if (!isWord)
                {
                    wordCount++;
                    isWord = true;
                }
            }
            else if (char.IsNullOrWhiteSpace(c))
            {
                isWord = false;
            }
        }
        return wordCount;
    }
    return CountWords(text) == CountWords(pattern);
}
}

```

4.2 Tata Cara Penggunaan Program

Berikut merupakan cara untuk menjalankan program:

1. Buka terminal



Gambar 4.2.1 Buka terminal

2. Import database yang ingin digunakan

```
C:\Users\DIERO PURNAMA>mariadb -u root -p stima3 < tubes3_stima.sql
```

Gambar 4.2.2 Import database

3. Masuk ke dalam folder *src* dalam terminal

```
|● PS C:\Users\DIERO PURNAMA\Desktop\PR\Tubes3_Panitia-Persiapan-Platform-Based-Development> cd src
○ PS C:\Users\DIERO PURNAMA\Desktop\PR\Tubes3_Panitia-Persiapan-Platform-Based-Development\src>
```

Gambar 4.2.3 Terminal buka src

4. Jalankan perintah “dotnet run”

```
○ PS C:\Users\DIERO PURNAMA\Desktop\PR\Tubes3_Panitia-Persiapan-Platform-Based-Development\src> dotnet run
```

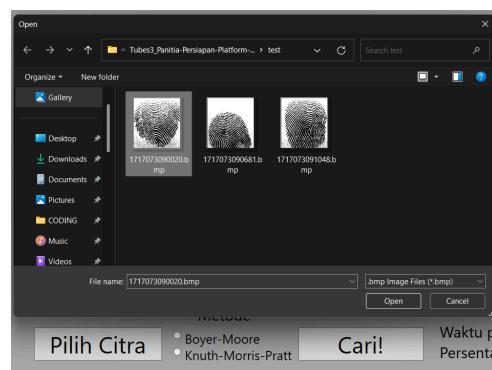
Gambar 4.2.4 Terminal dotnet run

5. Buka aplikasi



Gambar 4.2.5 Aplikasi

6. Klik tombol “Pilih Citra” dan pilih sidik jari yang ingin dicari



Gambar 4.2.6 Pilih sidik jari

7. Pilih algoritma yang ingin digunakan
8. Klik tombol “Cari!”
9. Aplikasi akan menampilkan biodata dan sidik jari dalam basis data yang memiliki tingkat kemiripan paling tinggi jika terdapat sidik jari yang memiliki tingkat kemiripan yang lebih dari 0,6.

4.3 Hasil Pengujian

- a. Test Case 1

Terdapat sidik jari dalam database dengan tingkat kemiripan lebih dari sama dengan 0,6.



Gambar 4.3.1 KMP Test Case 1.1



Gambar 4.3.2 KMP Test Case 1.2



Gambar 4.3.3 KMP Test Case 1.3



Gambar 4.3.4 BM Test Case 1.1



Gambar 4.3.5 BM Test Case 1.2



Gambar 4.3.6 BM Test Case 1.3

b. Test Case 2

Tidak terdapat sidik jari dalam database dengan tingkat kemiripan lebih dari 0,6.



Gambar 4.3.7 KMP Test Case 2



Gambar 4.3.8 BM Test Case 2

c. Test Case 3

Mencari data pemilik sidik jari dengan kondisi data nama yang dicari korup pada tabel sidik_jari.

```

mysql> select * from sidik_jari;
+-----+-----+
| berkas_citra | nama |
+-----+-----+
| 107__M_Right_middle_finger.bmp | Test Nama Normal |
| 105__M_Right_thumb_finger.bmp | t5t N4m 4Ly |
+-----+-----+
2 rows in set (0.08 sec)

```

Gambar 4.3.9 Rekord Tabel sidik_jari pada Basis Data



Gambar 4.3.10 KMP Test Case 3



Gambar 4.3.11 BM Test Case 3

d. Test Case 4

Mencari data pemilik sidik jari dengan kondisi data nama yang dicari normal (tidak korup) pada tabel sidik_jari.

```
mysql> select * from sidik_jari;
+-----+-----+
| berkas_citra | nama |
+-----+-----+
| 107__M_Right_middle_finger.bmp | Test Nama Normal |
| 105__M_Right_thumb_finger.bmp | t5t N4m 4Ly |
+-----+-----+
2 rows in set (0.08 sec)
```

Gambar 4.3.12 Rekord Tabel sidik_jari pada Basis Data



Gambar 4.3.13 KMP Test Case 4



Gambar 4.3.14 KMP Test Case 4

4.4 Analisis Hasil Pengujian

Dari hasil pengujian diatas, diperoleh data mengenai waktu pencarian program dari masing-masing algoritma. Berdasarkan data untuk pencarian yang terdapat sidik jari dalam database dengan tingkat kemiripan lebih dari atau sama dengan 0,6 pada test case 1, dapat dicari rata-rata waktu pencarian dari setiap algoritma, didapatkan hasil sebagai berikut

$$\bar{x}_{KMP} = \frac{1}{3} \sum_{i=1}^3 x_i \approx 70,67 \text{ ms, dan}$$

$$\bar{x}_{BM} = \frac{1}{3} \sum_{i=1}^3 x_i \approx 78,67 \text{ ms}$$

yang berarti bahwa algoritma KMP lebih cepat sedikit jika dibandingkan dengan algoritma BM, walaupun perbedaan rata-rata waktu pencarian kedua algoritma tidak terlalu signifikan. Hal ini disebabkan karena algoritma KMP memiliki kompleksitas algoritma $O(m + n)$ sedangkan algoritma BM memiliki kompleksitas algoritma $O(m * n)$.

Dari hasil pengujian, diperoleh juga bahwa implementasi regular expression dan algoritma Levenshtein Distance dalam melakukan penanganan potensi data korup, utamanya data bertipe string, dapat dilakukan. Regular expression dapat melakukan modifikasi terhadap string. Algoritma Levenshtein Distance yang memiliki kegunaan menghitung jumlah minimum operasi yang diperlukan untuk mentransformasi suatu string

menjadi bentuk string lainnya dengan menggunakan bantuan struktur data matriks dapat dimanfaatkan untuk mendapatkan jarak antara dua string. Algoritma ini memiliki kompleksitas waktu $O(mn)$, dengan m dan n adalah panjang setiap string yang dilakukan perbandingan.

BAB V

KESIMPULAN DAN SARAN

5.1. Kesimpulan

Algoritma *Knuth-Morris-Pratt* (KMP) dan *Boyer-Moore* (BM) merupakan algoritma yang efektif untuk melakukan pencarian pola pada citra sidik jari. Kedua algoritma dapat mencocokkan dua citra sidik jari dengan tingkat akurasi yang tinggi. Dari hasil pengujian, rata-rata waktu pencarian untuk algoritma KMP sedikit lebih rendah dibandingkan algoritma BM. Meskipun perbedaan rata-rata waktu pencarian tidak terlalu signifikan, algoritma KMP menunjukkan sedikit keunggulan dalam segi kecepatan pencarian.

Regular expression dan algoritma Levenshtein Distance dapat dimanfaatkan untuk melakukan *string matching* (pencocokan string) pada dua string yang dimasukkan. Regular expression dapat dimanfaatkan untuk melakukan modifikasi string dan juga pencocokan string. Algoritma Levenshtein Distance yang memiliki kegunaan menghitung jumlah minimum operasi yang diperlukan untuk mentransformasi suatu string menjadi bentuk string lainnya dengan menggunakan bantuan struktur data matriks dapat dimanfaatkan untuk mendapatkan jarak antara dua string. Kemudian, jarak yang dihasilkan akan dibandingkan dengan string terpanjang di antara kedua string tersebut untuk didapatkan tingkat kemiripan di antara kedua string tersebut. Dengan pemanfaatan regular expression dan algoritma Levenshtein Distance, penanganan optimal terhadap adanya potensi data korup, utamanya jika data bertipe string, dapat dilakukan.

5.2. Saran

Saran-saran untuk kelompok kami agar untuk kedepannya dapat mengerjakan tugas besar yang lainnya dengan lebih:

- 1) Mempelajari dengan lebih dalam bahasa pemrograman yang digunakan agar dapat memberikan hasil akhir yang lebih baik dan untuk mengurangi waktu yang dihabiskan untuk *debugging* program.
- 2) Meningkatkan komunikasi antar anggota kelompok.
- 3) Me-manage waktu dengan lebih baik.

5.3. Refleksi

Refleksi yang kami dapatkan setelah menyelesaikan tugas besar ini adalah bahwa mengintegrasikan *backend* program dengan GUI memakan waktu yang jauh lebih banyak dari yang kami pikirkan pada awalnya. Untuk kedepannya, kami merasa bahwa perlu dialokasikan lebih banyak waktu untuk integrasi program agar hasil akhir yang diberikan lebih maksimal.

LAMPIRAN

Repository github dapat diakses melalui pranala berikut:

https://github.com/DeltDev/Tubes3_Panitia-Persiapan-Platform-Based-Development

Link video:

<https://youtu.be/EHHK65JpHCA?si=0n0FXmhoG9dkDhRm>

DAFTAR PUSTAKA

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

<https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2022-2023/String-Matching-dengan-Regex-2019.pdf>