

**TUGAS BESAR 3**  
**STRATEGI ALGORITMA**

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



**Disusun Oleh Kelompok Oh\_hmm:**

Maria Flora Renata S. (13522010)

Evelyn Yosiana (13522083)

Julian Caleb S. (13522099)

**Teknik Informatika**

**Sekolah Teknik Elektro dan Informatika**

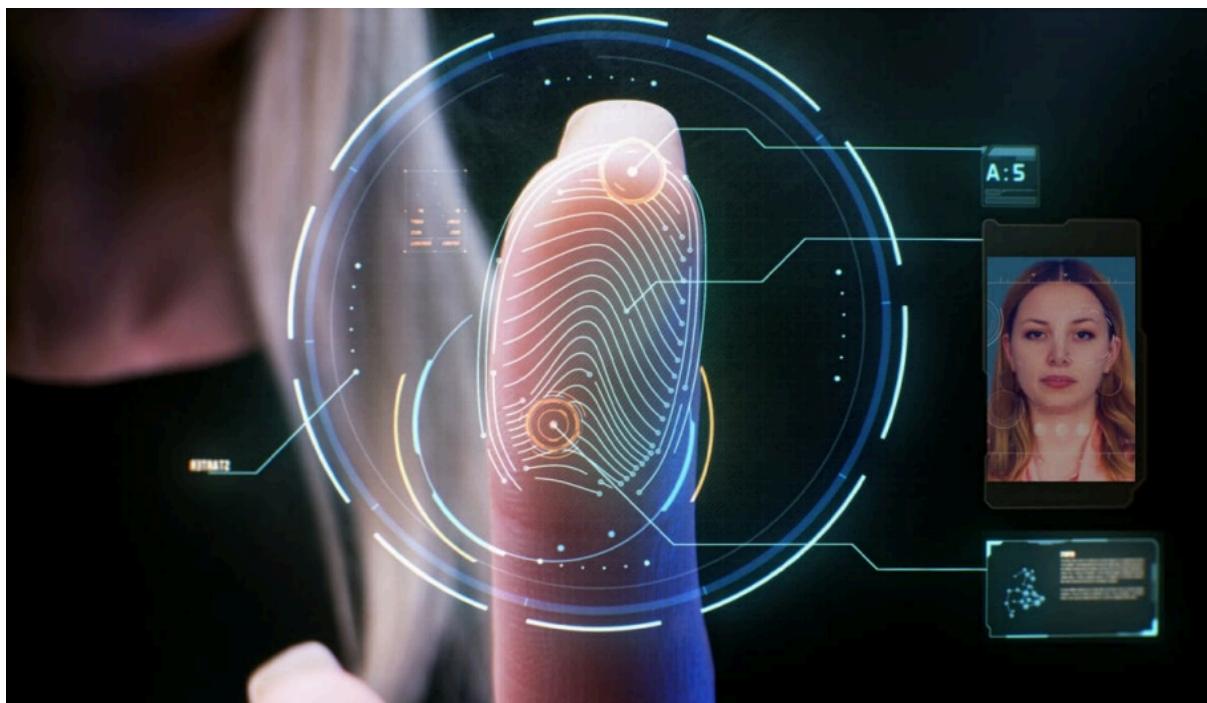
**Institut Teknologi Bandung**

## DAFTAR ISI

DAFTAR ISI.....	2
BAB 1.....	4
BAB 2.....	6
2.1. Algoritma String Matching.....	6
2.2. Teknik Pengukuran Persentase Kemiripan.....	7
2.3. Penjelasan Aplikasi.....	8
BAB 3.....	9
3.1. Langkah-Langkah Pemecahan Masalah.....	9
3.2. Proses penyelesaian masalah dengan algoritma KMP dan BM.....	9
3.3. Fitur Fungsional dan Arsitektur Aplikasi Desktop.....	9
3.4. Contoh Ilustrasi Kasus.....	9
BAB 4.....	10
4.1. Spesifikasi Teknis Program.....	10
4.2. Tata Cara Penggunaan Program.....	10
4.3. Hasil Pengujian.....	10
4.4. Analisis Hasil Pengujian.....	11
BAB 5.....	12
5.1. Kesimpulan.....	12
5.2. Saran.....	12
5.3. Refleksi.....	12
Lampiran.....	13
Daftar Pustaka.....	14

## BAB 1

### Deskripsi Tugas



Gambar 1. Ilustrasi fingerprint recognition pada deteksi berbasis biometrik.

Sumber: <https://www.aratek.co/news/unlocking-the-secrets-of-fingerprint-recognition>

Di era digital ini, keamanan data dan akses menjadi semakin penting. Perkembangan teknologi membuka peluang untuk berbagai metode identifikasi yang canggih dan praktis. Beberapa metode umum yang sering digunakan seperti kata sandi atau pin, namun memiliki kelemahan seperti mudah terlupakan atau dicuri. Oleh karena itu, biometrik menjadi alternatif metode akses keamanan yang semakin populer. Salah satu teknologi biometrik yang banyak digunakan adalah identifikasi sidik jari. Sidik jari setiap orang memiliki pola yang unik dan tidak dapat ditiru, sehingga cocok untuk digunakan sebagai identitas individu. Pattern matching merupakan teknik penting dalam sistem identifikasi sidik jari. Teknik ini digunakan untuk mencocokkan pola sidik jari yang ditangkap dengan pola sidik jari yang terdaftar di database. Algoritma pattern matching yang umum digunakan adalah Bozorth dan Boyer-Moore. Algoritma ini memungkinkan sistem untuk mengenali sidik jari dengan cepat dan akurat, bahkan jika sidik jari yang ditangkap tidak sempurna. Dengan menggabungkan teknologi identifikasi sidik jari dan pattern matching, dimungkinkan untuk membangun sistem identifikasi biometrik yang aman, handal, dan mudah digunakan. Sistem ini dapat

diaplikasikan di berbagai bidang, seperti kontrol akses, absensi karyawan, dan verifikasi identitas dalam transaksi keuangan. Di dalam Tugas Besar 3 ini, Anda diminta untuk mengimplementasikan sistem yang dapat melakukan identifikasi individu berbasis biometrik dengan menggunakan sidik jari. Metode yang akan digunakan untuk melakukan deteksi sidik jari adalah Boyer-Moore dan Knuth-Morris-Pratt. Selain itu, sistem ini akan dihubungkan dengan identitas sebuah individu melalui basis data sehingga harapannya terbentuk sebuah sistem yang dapat mengenali identitas seseorang secara lengkap hanya dengan menggunakan sidik jari.

## BAB 2

### Landasan Teori

#### 2.1. Algoritma String Matching

##### 2.1.1. KMP

Algoritma Knuth–Morris–Pratt (KMP) adalah sebuah algoritma pencocokan teks yang ditemukan oleh James H. Morris, Donald Knuth, dan Vaughan Pratt. Algoritma ini melakukan pencocokan dengan mencari dari kiri ke kanan. Saat terjadi ketidakcocokan karakter antara teks dan pola yang dicari, algoritma KMP akan menggunakan sebuah fungsi batas (*border function*) untuk menentukan besar pergeseran pola yang paling efektif.

Fungsi batas pada KMP bertugas untuk menentukan pergeseran pola paling banyak supaya tidak terjadi pengecekan yang tidak diperlukan (pengecekan pada bagian teks yang sudah pasti sama dengan pola). Fungsi batas akan mencari prefiks terpanjang yang juga merupakan sudiks pada awal pola hingga indeks sebelum terjadi ketidakcocokan. Dengan fungsi batas tersebut, pergeseran pada KMP menjadi lebih efektif dibandingkan dengan pergeseran pada algoritma *brute-force*.

##### 2.1.2. BM

Algoritma Boyer-Moore (BM) adalah sebuah algoritma pencocokan teks yang ditemukan oleh Robert S. Boyer dan J Strother Moore. Algoritma ini memiliki dua teknik utama, yaitu *looking-glass* dan *character jump*. Teknik *looking-glass* berarti mencocokkan pola secara terbalik/mulai dari belakang.

Teknik *character jump* mendefinisikan tiga kemungkinan saat terjadi sebuah ketidakcocokan antara pola dan teks pada suatu karakter. Ketiga kemungkinan tersebut dicek secara berurutan. Misalkan terjadi ketidakcocokan pada karakter c di lokasi T[i] pada teks,

1. Jika pola mengandung c, geser pola ke kanan ke indeks terakhir munculnya x, sehingga c dan T[i] selaras.

2. Jika pola mengandung c, tetapi tidak bisa digeser ke kanan untuk sampai ke c, maka pola akan digeser ke kanan sebanyak satu indeks.
3. Jika kedua kondisi tersebut tidak dapat diaplikasikan (pola tidak mengandung c), maka geser pola sehingga awal pola tersebut selaras dengan  $T[i+1]$ .

Untuk menentukan indeks terakhir setiap karakter pada pola, pola akan diproses sebelum memulai pencarian. Proses tersebut akan menghasilkan sebuah tabel yang menunjukkan indeks terakhir setiap karakter dalam alfabet yang digunakan pada pola.

### 2.1.3. Regex

Regex (Regular Expression) adalah sebuah pola yang dibentuk dari karakter-karakter. Pola tersebut akan digunakan untuk mencari semua bagian yang cocok dari sebuah teks. Regex ditulis seperti teks biasa, dengan beberapa karakter khusus untuk merepresentasikan pilihan, *range*, pengulangan, dan hal lain.

## 2.2. Teknik Pengukuran Persentase Kemiripan

Algoritma String Matching yang digunakan mungkin tidak bisa sama persis menemukan pattern pada text yang digunakan, sehingga diperlukan algoritma untuk pengukuran persentase kemiripan, sehingga ketika tidak ada yang cocok, akan diambil bagian dengan persentase kemiripan tertinggi.

Algoritma pengukuran persentase kemiripan yang digunakan adalah Levenshtein Distance, yaitu fungsi yang mengukur kemiripan antara dua string, dengan memperhitungkan jumlah operasi penyisipan, penghapusan, dan substitusi yang diperlukan untuk mengubah satu string ke string lainnya. Terdapat beberapa jenis mengimplementasikan Levenshtein Distance, seperti menggunakan pendekatan rekursif atau menggunakan matrix n kali m dengan n dan m adalah panjang kedua string yang masing-masing ditambah 1. Hasil dari Levenshtein Distance dibagi dengan panjang string terpanjang dikali 100 persen untuk menghasilkan persentase.

### **2.3. Penjelasan Aplikasi**

Aplikasi untuk tugas besar 3 ini adalah sebuah desktop app yang dibuat dengan menggunakan .NET (dibaca Dotnet), sebuah framework pembuatan software komputer yang menggunakan bahasa pemrograman C#. Pada aplikasi, akan terdapat GUI yang berisi setidaknya tombol untuk memasukkan sidik jari yang ingin dicari identitasnya, opsi memilih algoritma string matching yang digunakan (BM atau KMP), tombol untuk memulai pencarian, area untuk menampilkan sidik jari yang dicari dan area untuk menampilkan hasil pencarian.

## BAB 3

### Analisis Pemecahan Masalah

#### 3.1. Langkah-Langkah Pemecahan Masalah

1. Pertama, dari sidik jari yang diupload oleh pengguna, diambil 32 pixel sebagai pattern yang akan dicari. Pixel diambil dari bagian tengah gambar sidik jari dikarenakan ujung dari gambar biasanya tidak terdapat bagian dari sidik jari.
2. Pixel tersebut akan diubah menjadi representasi binarynya, dengan pixel yang gelap menjadi 1, dan pixel yang terang menjadi 0.
3. Binary akan diubah menjadi string ASCII 4 karakter.
4. String ASCII tersebut akan dijadikan pola untuk dicocokkan menggunakan algoritma KMP atau BM. Pola tersebut akan dicocokkan dengan seluruh sidik jari di dalam basis data. Jika algoritma pencocokan string gagal, maka akan dicari jarak antara sidik jari dengan pola menggunakan Levenshtein Distance
5. Nama dari tabel biodata akan diperbaiki menggunakan regex. Angka-angka yang sering digunakan untuk mensubstitusi huruf akan diubah kembali menjadi huruf, dan kapitalisasi semua nama akan disamakan.
6. Nama dari sidik jari yang paling mendekati akan menjadi pola berikutnya. Pola ini akan dicocokkan dengan cara yang sama dengan semua nama di basis data, pada tabel biodata.
7. Biodata dari nama yang paling cocok akan ditunjukkan.

#### 3.2. Proses penyelesaian masalah dengan algoritma KMP dan BM

1. Pengguna memilih algoritma string matching yang digunakan dalam pencarian pemilik sidik jari.
2. Pattern yang digunakan adalah 32 pixel dari sidik jari seperti yang dijelaskan pada 3.1, sedangkan text yang digunakan adalah translasi setiap bitmap sidik jari yang ada pada database menjadi ASCII. Satu pattern akan dibandingkan dengan setiap text ASCII, dan mengembalikan text yang mengandung pattern (atau yang memiliki persentase kemiripan paling tinggi).
3. Jika memilih KMP, dibuat sebuah border array, di mana untuk setiap posisi dalam pattern (kecuali karakter terakhir), akan disimpan panjang prefix terpanjang yang

juga merupakan suffix. Kemudian, akan dilakukan string matching dari awal teks ke akhir teks, dari awal pattern ke akhir pattern, menggunakan algoritma KMP, memanfaatkan border array yang sudah dibuat.

4. Jika memilih BM, dibuat sebuah last occurrence array, sepanjang jumlah karakter ASCII, di mana untuk setiap karakter ASCII, jika muncul pada pattern akan memasukkan index terakhir kemunculan karakter pada pattern ke dalam array sesuai nomor karakter ASCII, dan jika tidak, akan diberi nilai -1. Kemudian, akan dilakukan string matching dari awal text ke akhir text, dari akhir pattern ke awal pattern, menggunakan algoritma BM,
5. Jika setelah menggunakan KMP dan BM tidak ditemukan exact match, akan dibandingkan seluruh bitmap sidik jari yang diupload (dalam bentuk ASCII) dengan masing-masing gambar sidik jari pada database dengan menggunakan Levenshtein Distance dan akan diambil gambar sidik jari dengan persentase tertinggi.

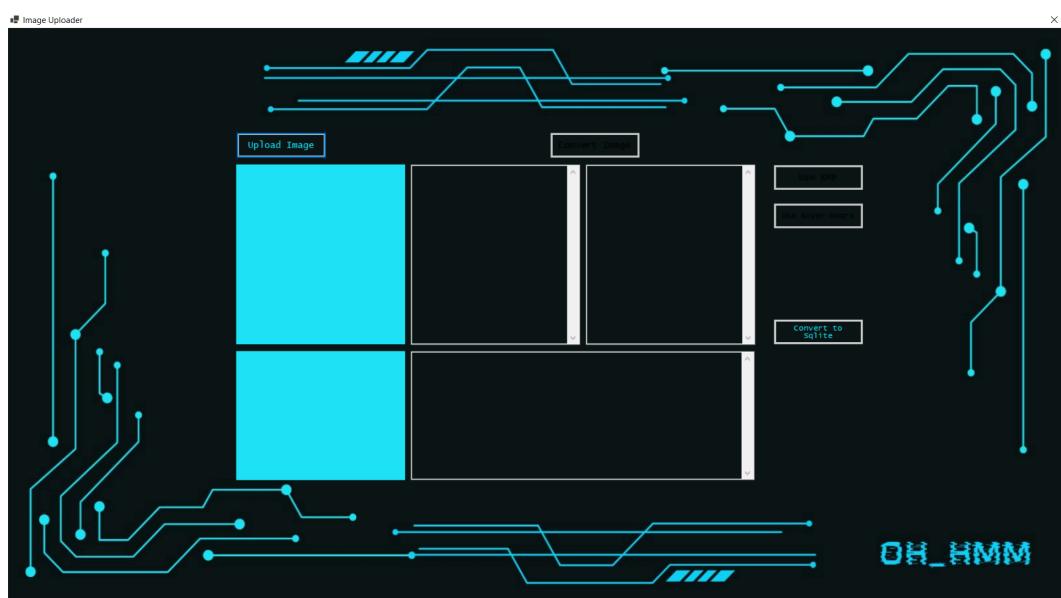
### **3.3. Fitur Fungsional dan Arsitektur Aplikasi Desktop**

Fitur fungsional yang terdapat pada aplikasi:

No	Fitur fungsional
F01	Pengguna dapat mengunggah sebuah gambar (sidik jari)
F02	Pengguna dapat melihat gambar yang diunggah
F03	Pengguna dapat memilih algoritma yang ingin digunakan (BM atau KMP)
F04	Pengguna dapat melakukan pencarian berdasarkan algoritma yang ingin digunakan
F05	Pengguna dapat melihat sidik jari pada database yang paling mirip dengan yang diunggah beserta persentase tingkat kemiripannya
F06	Pengguna dapat melihat lama waktu eksekusi program
F07	Pengguna dapat melihat biodata dari pemilik sidik jari

Arsitektur aplikasinya adalah sebagai berikut:

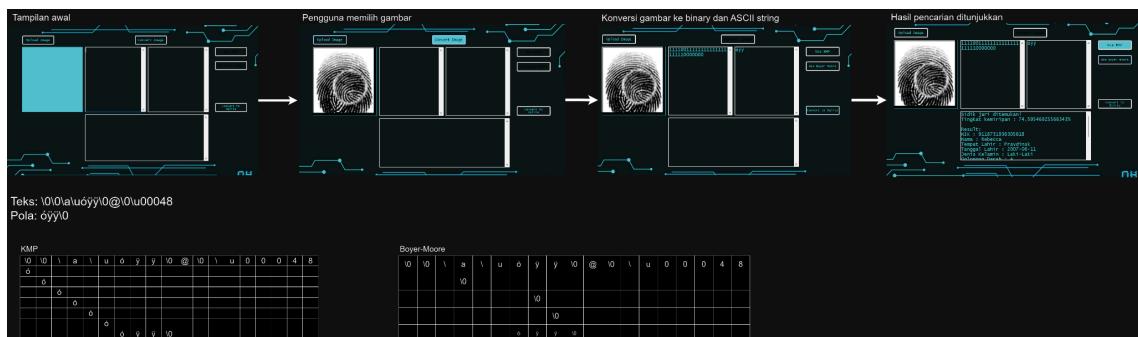
- Secara lingkungan implementasi,
  - Dotnet sebagai framework,
  - C# sebagai bahasa pemrograman
  - Visual Studio Code sebagai text editor
  - SQLite sebagai database
- Secara UI, terdapat judul aplikasi (nama kelompok) di kanan bawah, tombol bertulisan “Upload Image” untuk mengupload gambar, gambar sidik jari yang diupload dan sidik jari hasil pencarian di bawah tombol, tombol “Convert Image” di tengah, dengan hasil biner dan ASCII dibawahnya, diikuti area biodata hasil pencarian, dan terdapat tombol “BM” dan “KMP” untuk memilih algoritma serta tombol “Convert to SQLite” di sebelah kanan.





### 3.4. Contoh Ilustrasi Kasus

Berikut adalah ilustrasi kasus, untuk gambar sidik jari yang diunggah, dilakukan string matching baik dengan algoritma KMP atau algoritma BM, beserta hasilnya.



## BAB 4

### Implementasi dan Pengujian

#### 4.1. Spesifikasi Teknis Program

Aplikasi ditulis dan dibangun menggunakan bahasa C#. Aplikasi dapat dijalankan pada sistem operasi Windows dengan melakukan command “dotnet run” pada folder src. Dump sql yang digunakan oleh basis data aplikasi berada di src/newConverted\_sqlite.sql, sedangkan foto di dalam basis data terletak di test/real.

Gambar sidik jari yang di-upload maupun yang dimasukkan ke dalam basis data harus memiliki warna yang gelap dan warna latar belakang yang terang agar dapat terbaca dengan benar oleh aplikasi.

##### 4.1.1. *Function ToNormal*

*Function* ini menerima parameter berupa nama (string) kemudian mengubah karakter-karakter alay menjadi karakter asli (misalnya ‘4’ diubah menjadi ‘a’) menggunakan regex. Fungsi ini mengembalikan nama yang sudah dinormalisasi (string). Berikut implementasi dari *function* ToNormal.

```
public static string ToNormal(string name) {  
    string capitalization = @"(\w+ )";  
  
    string a = @"4";  
  
    string b = @"(13)";  
  
    string d = @"(17)";  
  
    string e = @"3";  
  
    string g = @"[69]";  
  
    string i = @"1";  
  
    string o = @"0";  
  
    string r = @"(12)";  
  
    string s = @"5";  
  
  
    string normal;
```

```

        normal = Regex.Replace(name, b, "b");

        normal = Regex.Replace(normal, d, "d");

        normal = Regex.Replace(normal, r, "r");

        normal = Regex.Replace(normal, a, "a");

        normal = Regex.Replace(normal, e, "e");

        normal = Regex.Replace(normal, g, "g");

        normal = Regex.Replace(normal, i, "i");

        normal = Regex.Replace(normal, o, "o");

        normal = Regex.Replace(normal, s, "s");

        normal = Regex.Replace(normal, capitalization, new
MatchEvaluator(CapitalizeNames));

        return normal;

    }

```

#### 4.1.2. *Function CapitalizeNames*

*Function* ini menerima parameter berupa nama (string) yang sudah dinormalisasi, kemudian setiap karakternya dikapitalisasi. Fungsi ini mengembalikan nama yang sudah dikapitaliasasi (string). Berikut implementasi dari *function CapitalizeNames*.

```

private static string CapitalizeNames(Match m) {

    string str = m.ToString();

    if (char.ToLower(str[0])) return char.ToUpper(str[0]) + str.Substring(1,
str.Length - 1);

    else return str;

}

```

#### 4.1.3. *Function borderFunction*

*Function* ini menerima parameter berupa *pattern* (string) kemudian mengembalikan border dari *pattern* tersebut. Berikut implementasi dari *function borderFunction*.

```

private static int[] borderFunction(string pattern) {

    // Inisialisasi variable

    int m = pattern.Length - 1;

    int[] border = new int[m];

    int length = 0;

    int i = 1;

    // Elemen pertama border selalu 0

    border[0] = 0;

    // Iterasi setiap elemen pada pattern, mulai elemen kedua

    while (i < m) {

        // Jika cocok, set border[i] menjadi length

        // (panjang prefix yang cocok dengan suffix hingga char index ke-i)

        if (pattern[i] == pattern[length]) {

            length++;

            border[i] = length;

            i++;

            // Jika tidak cocok, lihat lengthnya

        } else {

            // Jika ada yang cocok sebelumnya, set length menjadi

            // nilai border sebelumnya

            if (length != 0) {

                length = border[length - 1];

                // Jika tidak ada yang cocok sebelumnya, set 0

            } else {

                border[i] = 0;

                i++;

            }

        }

    }

}

```

```
// Kembalikan border

return border;

}
```

#### 4.1.4. Function KMPSearch

*Function* ini menerima parameter berupa *pattern* (*string*) dan *text* (*string*) kemudian mengembalikan nilai boolean berupa *true* jika *pattern* ditemukan dalam *text* dengan menggunakan algoritma KMP dan mengembalikan *false* jika sebaliknya. Berikut implementasi dari *function* KMPSearch.

```
public static bool KMPSearch(string pattern, string text) {

    // Inisialisasi variabel

    int m = pattern.Length;

    int n = text.Length;

    bool found = false;

    // Membuat border array

    int[] border = KMPAlgorithm.borderFunction(pattern);

    // Iterasi index

    int i = 0;

    int j = 0;

    // Selama text belum habis,

    while ((i < n) && !found) {

        // Jika karakter cocok, majukan kedua index

        if (pattern[j] == text[i]) {

            j++;

            i++;

        }

    }

}
```

```

        // Jika sudah mencapai akhir pattern, pattern ditemukan dalam text,
break dari loop

        if (j == m) {

            Console.WriteLine("Found pattern at index " + (i - j));

            found = true;

            // Jika tidak, cek apakah text belum habis dan apakah karakter tidak
            // cocok.

        } else if (i < n && pattern[j] != text[i]) {

            // Jika index untuk pattern j != 0, pindah index pattern sesuai
            dengan

            // index ke j - 1 pada border, memungkinkan kecocokan parsial

            if (j != 0) {

                j = border[j - 1];

                // Jika j = 0, iterasi i

            } else {

                i++;

            }

        }

    }

    if ((i == n) && !found) {

        Console.WriteLine("Pattern not found.");

        return false;

    }

    return true;

}

```

#### 4.1.5. Function BMSearch

*Function* ini menerima parameter berupa *pattern* (*string*) dan *text* (*string*) kemudian mengembalikan nilai boolean berupa *true* jika *pattern* ditemukan dalam *text* dengan menggunakan algoritma BM dan mengembalikan *false* jika sebaliknya. Berikut implementasi dari *function* BMSearch.

```

public static bool BMSearch(string pattern, string text) {

    // Instansiasi variabel

    int[] last = BMAlgorithm.lastOccurrenceFunction(pattern);

    int m = pattern.Length;

    int n = text.Length;

    bool found = false;

    // Iterasi index, jalan dari kiri ke kanan text

    // Dicek dari kanan ke kiri.

    int i = m - 1;

    int j = m - 1;

    // Melakukan iterasi

    while ((i < n) && !found) {

        // Jika match

        if (pattern[j] == text[i]) {

            // Jika sudah karakter pertama

            if (j == 0) {

                Console.WriteLine("Found pattern at index " + (i));

                found = true;

            // Jika belum, terapkan looking-glass technique

            // Looking-glass technique: mencari P di T dengan berjalan mundur

            // dari ujung akhir

            } else {

                i--;

                j--;

            }

        // Jika mismatch / tidak cocok, terapkan character-jump technique

        // Character-jump technique: Jika ada mismatch, dengan T[i] = x,

        // dan P[j] != x, atasi diantara 3 kasus berikut:

        // 1. Jika P ada x di kiri j, shift P ke kanan untuk align dengan last

        index x
    }
}

```

```

        // 2. Jika p ada x di kanan j, shift P ke kanan 1 karakter ke T[i+1]

        // 3. Jika x tidak ada di P, shif P sehingga P[0] = T[i+1]

    } else {

        int lo = last[text[i]];

        i = i + m - Math.Min(j, 1 + lo);

        j = m - 1;

    }

}

// Jika tidak ada yang match

if ((i > n - 1) && !found) {

    Console.WriteLine("Pattern not found.");

    return false;

}

return true;

}

```

#### 4.1.6. *Function LevenshteinDistance*

*Function* ini menerima parameter berupa dua *string* yang akan dicari tingkat kemiripannya. Fungsi ini mengembalikan persentase kemiripan dua string dari parameter dengan menggunakan algoritma *Levenshtein Distance*, Berikut implementasi dari *function* LevenshteinDistance.

```

public static double LevenshteinDistance(string s, string t) {

    // Membuat matrix untuk menyimpan perbedaan karakter

    int[,] matrix = new int[s.Length + 1, t.Length + 1];

    // Inisialisasi matrix

    // Untuk kolom

    for (int i = 0; i <= s.Length; i++) {

        matrix[i, 0] = i;
    }
}

```

```

    }

    // Untuk baris

    for (int j = 0; j <= t.Length; j++) {

        matrix[0, j] = j;

    }

    // Menghitung Levenshtein matrix

    for (int j = 1; j <= t.Length; j++) {

        for (int i = 1; i <= s.Length; i++) {

            // Jika karakter sama, tidak perlu ada perubahan

            if (s[i - 1] == t[j - 1])

                matrix[i, j] = matrix[i - 1, j - 1];

            else

                // Jika tidak, cari minimum antara deletion, insertion, atau
                substitution

                matrix[i, j] = Math.Min(Math.Min(
                    matrix[i - 1, j] + 1, // Deletion
                    matrix[i, j - 1] + 1), // Insertion
                    matrix[i - 1, j - 1] + 1); // Substitution

        }

    }

    // Menghitung persentase nilai terakhir matrix yang merupakan Levenshtein
    Distance antar kedua string

    // Persentase = (1 - (Levenshtein Distance / Max Length)) * 100%

    double distance = (1 - ((double)matrix[s.Length, t.Length] /
    (double)Math.Max(s.Length, t.Length))) * 100;

    return distance;
}

```

#### **4.1.7. Function convertImageToBinary**

*Function* ini menerima parameter berupa *path* dari gambar yang akan dikonversi, panjang pixel, dan lebar pixel, kemudian mengembalikan nilai binary dari gambar tersebut (*string*). Berikut implementasi dari *function* convertImageToBinary.

```
private string convertImageToBinary(string imagePath, int widthPixels, int heightPixels)

{
    StringBuilder binaryBuilder = new StringBuilder();

    try
    {
        Bitmap bitmap = new Bitmap(imagePath);

        int bitmapWidth = bitmap.Width;
        int bitmapHeight = bitmap.Height;

        // Jika 6 x 5 pixels

        int startWidth = Math.Max((bitmapWidth - widthPixels) / 2, 0);
        int startHeight = Math.Max((bitmapHeight - heightPixels) / 2, 0);
        int width = Math.Min(widthPixels, bitmapWidth);
        int height = Math.Min(heightPixels, bitmapHeight);

        for (int y = startHeight; y < startHeight + height; y++)
        {
            for (int x = startWidth; x < startWidth + width; x++)
            {
                Color pixelColor = bitmap.GetPixel(x, y);

                int grayscaleValue = (pixelColor.R + pixelColor.G +
pixelColor.B) / 3;

                binaryBuilder.Append(grayscaleValue < 128 ? "1" : "0");
            }
        }
    }
    catch (Exception ex)
    {

```

```

        MessageBox.Show("Error: " + ex.Message);

        return string.Empty;
    }

    return binaryBuilder.ToString();
}

```

#### 4.1.8. Function convertBinaryToAscii

*Function* ini menerima parameter berupa binary dari gambar (*string*) kemudian mengonversinya menjadi ascii. Berikut implementasi dari *function* convertBinaryToAscii.

```

private string convertBinaryToAscii(string binaryString)
{
    StringBuilder asciiString = new StringBuilder();

    for (int i = 0; i < binaryString.Length; i += 8)
    {
        z
    }
}

```

#### 4.1.9. Function convertImageToAscii

*Function* ini menerima parameter berupa *path* dari gambar yang akan dikonversi, kemudian memanggil fungsi convertImageToBinary dan mengembalikan string berisi nilai ascii dari gambar tersebut. Berikut implementasi dari *function* convertImageToAscii.

```

private string convertImageToAscii(string path)
{
    string binaryImage = convertImageToBinary(path, 6, 5);
    string asciiImage = convertBinaryToAscii(binaryImage);
    string res = "";
    foreach (char c in asciiImage)

```

```

    {
        res += c;
    }

    return res;
}

```

#### 4.1.10. Prosedur matchingHandler

*Function* ini menerima parameter berupa ascii dari gambar yang dicari dan algoritma yang dipilih. Dengan membandingkannya dengan seluruh gambar dari basis data, fungsi ini mencari gambar yang paling mirip dengan gambar yang dicari dengan menggunakan algoritma yang dipilih. Kemudian, nama dari pemilik gambar target dibandingkan dengan seluruh nama dari basis data untuk mencari biodata dari pemilik gambar sidik jari yang dicari. Setelah biodata ditemukan, program akan menampilkan biodata tersebut pada GUI. Berikut implementasi dari *function* matchingHandler.

```

private void matchingHandler(string asciiString, string algo)
{
    // Time exe

    Stopwatch stopwatch = new Stopwatch();

    stopwatch.Start();

    // TODO:

    // 1. Query ke database(?) 

    List<string> paths = Query.getAllPaths();

    // 2. Get all fingerprints

    List<Dictionary<string, string>> convertedPathsList = new
    List<Dictionary<string, string>>();

    foreach (string path in paths)

    {
        string basePath = Path.Combine(AppDomain.CurrentDomain.BaseDirectory,
        "...", "...", "...", "...");

```

```

        basePath = Path.Combine(basePath, path);

        string asciiImage = convertImageToAscii(basePath);

        Dictionary<string, string> pathMap = new Dictionary<string, string>

        {
            { "originalPath", path },
            { "convertedPath", asciiImage }
        };

        convertedPathsList.Add(pathMap);
    }

    double distance = 99999;
    string closest = null;

    // 3. Cari path yang mirip
    foreach (var pathMap in convertedPathsList)
    {
        string originalPath = pathMap["originalPath"];
        string convertedPath = pathMap["convertedPath"];

        if (algo.Equals("KMP"))
        {
            if (KMPAlgorithm.KMPSearch(asciiString, convertedPath))
            {
                closest = originalPath;
                distance = 0;
                break;
            }
            else
            {

```

```

        double dist =
TingkatKemiripan.LevenshteinDistance(asciiString, convertedPath);

        if (distance > dist)

        {

            distance = dist;

            closest = originalPath;

        }

    }

}

else

{

    if (BMAlgorithm.BMSearch(asciiString, convertedPath))

    {

        closest = originalPath;

        distance = 0;

        break;

    }

    else

    {

        double dist =
TingkatKemiripan.LevenshteinDistance(asciiString, convertedPath);

        if (distance > dist)

        {

            distance = dist;

            closest = originalPath;

        }

    }

}

// Sampai sini dapet closest = path yang paling mirip

string oriName = Query.getNameBasedOnPath(closest); // -> nama yg benar

```

```

    // 4. query biodata table with original, alay, name.... uh, belum kebayang
    sebenarnya kalo belum ada db. mungkin perlu hashmap nama normalized -> nama di
    db (alay) ??

    List<Dictionary<string, string>> nameList = Query.GetAllNames();

    string nameRes = "";
    double min = 99999;

    foreach (var nameMap in nameList)
    {
        string decryptedName = nameMap["decryptedName"];

        string nameInDB = nameMap["originalName"];
        decryptedName = AlayMatcher.ToNormal(decryptedName);

        if (algo.Equals("KMP"))
        {
            if (KMPAlgorithm.KMPSearch(decryptedName, oriName))
            {
                nameRes = nameInDB;
                break;
            }
        }
        else
        {
            double dist = TingkatKemiripan.LevenshteinDistance(oriName,
decryptedName);

            if (min > dist)
            {
                min = dist;
                nameRes = nameInDB;
            }
        }
    }
}

```

```

    {

        if (BMAlgorithm.BMSearch(decryptedName, oriName))

        {

            nameRes = nameInDB;

            break;

        }

        else

        {

            double dist = TingkatKemiripan.LevenshteinDistance(oriName,
decryptedName);

            if (min > dist)

            {

                min = dist;

                nameRes = nameInDB;

            }

        }

    }

}

// Time exe

stopwatch.Stop();

List<String> identity = Query.getIdentityBasedOnName(nameRes);

if (identity != null && identity.Count > 0)

{

    string message = "Sidik jari ditemukan!" + Environment.NewLine;

    message += "Tingkat kemiripan : " + (1-distance)*100 + "%" +
Environment.NewLine + Environment.NewLine;

    message += "Result:" + Environment.NewLine;

    message += "NIK : " + identity[0] + Environment.NewLine;

    message += "Nama : " + oriName + Environment.NewLine;

    message += "Tempat Lahir : " + identity[2] + Environment.NewLine;

```

```

        message += "Tanggal Lahir : " + identity[3] + Environment.NewLine;

        message += "Jenis Kelamin : " + identity[4] + Environment.NewLine;

        message += "Golongan Darah : " + identity[5] + Environment.NewLine;

        message += "Alamat : " + identity[6] + Environment.NewLine;

        message += "Agama : " + identity[7] + Environment.NewLine;

        message += "Status Perkawinan : " + identity[8] + Environment.NewLine;

        message += "Pekerjaan : " + identity[9] + Environment.NewLine;

        message += "Kewarganegaraan : " + identity[10] + Environment.NewLine +
Environment.NewLine;

        message += "Time execution: " + stopwatch.ElapsedMilliseconds + " ms";

        result.Text = message;
    }

    else
    {
        MessageBox.Show("No identity found!");
    }
}

```

#### 4.1.11. Prosedur convertToSqlite

Prosedur ini mengonversi file sql sehingga formartnya sesuai dengan kebutuhan program menggunakan regex (program ini menggunakan sqlite). Berikut implementasi dari prosedur convertToSqlite.

```

public static string ConvertToSqlite(string mysql)
{
    // Remove enum

    mysql = Regex.Replace(mysql, @"enum\(([^\)]+\)\)\s*DEFAULT NULL", "TEXT");

    // Syntax

    mysql = Regex.Replace(mysql, @"(var)?char\(\d+\)", "TEXT");

    mysql = Regex.Replace(mysql, @"(tiny|small)?int\(\d+\)", "INTEGER");
}

```

```

mysql = Regex.Replace(mysql, @"`(\w+)`", "$1");

// Create Table

mysql = Regex.Replace(mysql, @"ENGINE=InnoDB", "");
mysql = Regex.Replace(mysql, @"DEFAULT CHARSET=\w+", "");
mysql = Regex.Replace(mysql, @"COLLATE=\w+", "");
mysql = Regex.Replace(mysql, @"AUTO_INCREMENT", "AUTINCREMENT");

// Lock Unlock

mysql = Regex.Replace(mysql, @"LOCK TABLES biodata WRITE;", "");
mysql = Regex.Replace(mysql, @"LOCK TABLES sidik_jari WRITE;", "");
mysql = Regex.Replace(mysql, @"UNLOCK TABLES;", "");

// ect

mysql = Regex.Replace(mysql, @"Displaying tubes3_stima24.sql.", "");
mysql = Regex.Replace(mysql, @"tubes3_stima24.sql", "");

return mysql;
}

```

#### 4.1.12. Prosedur LoadSQLFile

Prosedur ini membaca file sql yang sudah disesuaikan kemudian membuat atau menimpa database yang akan digunakan berdasarkan file sql tersebut. Berikut implementasi dari prosedur LoadSQLFile.

```

public static void LoadSQLFile()
{
    string connString = "Data Source=src/tubes3_stima24.db;";

    string sql = File.ReadAllText("new_converted_sqlite.sql");

    using (var conn = new SQLiteConnection(connString))

```

```

{
    try
    {
        conn.Open();

        using (var cmd = new SQLiteCommand(sql, conn))
        {
            cmd.ExecuteNonQuery();
        }
    }

    catch (SQLiteException sqlEx)
    {
        Console.WriteLine(sqlEx.Message);
    }

    catch (Exception ex)
    {
        Console.WriteLine(ex.Message);
    }
}

```

#### 4.1.13. Fungsi-Fungsi Query

Fungsi-fungsi ini bertujuan untuk melakukan query untuk mendapatkan data dari database. Berikut implementasinya.

```

public static List<string> getAllPaths()
{
    string connString = "Data Source=tubes3_stima24.db;";

    List<string> paths = new List<string>();

    using (var conn = new SQLiteConnection(connString))
    {

```

```

try
{
    conn.Open();

    string query = "SELECT berkas_citra FROM sidik_jari;"

    using (var cmd = new SQLiteCommand(query, conn))
    {
        using (var reader = cmd.ExecuteReader())
        {
            if (reader.HasRows)
            {
                while (reader.Read())
                {
                    string berkasCitra = reader.GetString(0);
                    paths.Add(berkasCitra);
                }
            }
            else
            {
                Console.WriteLine("Gaada huhu");
            }
        }
    }
}

catch (SQLiteException sqlEx)
{
    Console.WriteLine(sqlEx.Message);
}

catch (Exception ex)
{
    Console.WriteLine(ex.Message);
}

```

```

        }

    }

    return paths;
}

}

public static string getNameBasedOnPath(string berkasCitra)
{
    string connString = "Data Source=tubes3_stima24.db;";
    string name = null;

    using (var conn = new SQLiteConnection(connString))
    {
        try
        {
            conn.Open();

            string query = "SELECT nama FROM sidik_jari WHERE berkas_citra = @berkasCitra LIMIT 1;";

            using (var cmd = new SQLiteCommand(query, conn))
            {

                cmd.Parameters.AddWithValue("@berkasCitra", berkasCitra);

                using (var reader = cmd.ExecuteReader())
                {
                    if (reader.HasRows)
                    {
                        while (reader.Read())
                        {
                            name = reader.GetString(0);
                        }
                    }
                }
            }
        }
    }
}

```

```

        else

    {

        Console.WriteLine("Gaada huhu");

    }

}

}

catch (SQLiteException sqlEx)

{

    Console.WriteLine(sqlEx.Message);

}

catch (Exception ex)

{

    Console.WriteLine(ex.Message);

}

}

return name;

}

public static List<string> getIdentityBasedOnName(string name)

{

    string connString = "Data Source=tubes3_stima24.db;";

    List<string> identity = new List<string>();




    using (var conn = new SQLiteConnection(connString))

    {

        try

        {

            conn.Open();

            string query = "SELECT NIK, nama, tempat_lahir, tanggal_lahir,
jenis_kelamin, golongan_darah, alamat, agama, status_perkawinan, pekerjaan,
kewarganegaraan FROM biodata WHERE nama = @name LIMIT 1;";


```

```

        using (var cmd = new SQLiteCommand(query, conn))
        {

            cmd.Parameters.AddWithValue("@name", name);

            using (var reader = cmd.ExecuteReader())
            {

                if (reader.HasRows)
                {

                    while (reader.Read())
                    {

                        identity.Add(reader.GetString(0));

                        identity.Add(reader.GetString(1));

                        identity.Add(reader.GetString(2));

                        identity.Add(reader.GetString(3));

                        identity.Add(reader.GetString(4));

                        identity.Add(reader.GetString(5));

                        identity.Add(reader.GetString(6));

                        identity.Add(reader.GetString(7));

                        identity.Add(reader.GetString(8));

                        identity.Add(reader.GetString(9));

                        identity.Add(reader.GetString(10));
                    }
                }
                else
                {
                    Console.WriteLine("Gaada huhu");
                }
            }

            for (int i = 0; i < 11; i++)
            {

                List<BigInteger> encrypted =
ConvertToIntList(identity[i]);

```

```

        string decrypted = DecryptData(encrypted);

        identity[i] = decrypted;

    }

}

}

catch (SQLiteException sqlEx)
{
    Console.WriteLine(sqlEx.Message);
}

catch (Exception ex)
{
    Console.WriteLine(ex.Message);
}

}

return identity;
}

```

#### **4.1.14. Function Encrypt dan EncryptChar**

Fungsi ini mengenkripsi suatu kalimat (*string*) menggunakan algoritma RSA yang dijelaskan pada slide perkuliahan matematika diskrit. Nilai kunci n dan d diselipkan di awal dan akhir kalimat hasil enkripsi. Berikut implementasi dari fungsi Encrypt.

```

public List<BigInteger> Encrypt(string message)
{
    List<BigInteger> encryptedMessage = new
List<BigInteger>();
    encryptedMessage.Add(new BigInteger(d_value));
    foreach (char c in message)
    {

```

```

        BigInteger encryptedChar = EncryptChar((int)c);

        encryptedMessage.Add(encryptedChar);

    }

    encryptedMessage.Add(new BigInteger(n_value));

    return encryptedMessage;

}

```

```

private BigInteger EncryptChar(int message)

{
    BigInteger bigMessage = new BigInteger(message);

    BigInteger bigE = new BigInteger(e_value);

    BigInteger bigN = new BigInteger(n_value);

    BigInteger encrypted = this.ModPow(bigMessage, bigE,
bigN);

    return encrypted;
}

```

#### **4.1.15. Function Decrypt dan DecryptChar**

Fungsi ini mendekripsi suatu kode (*big integer*) menggunakan algoritma RSA yang dijelaskan pada slide perkuliahan matematika diskrit. Berikut implementasi dari fungsi Decrypt dan DecryptChar.

```

public string Decrypt()

{
    StringBuilder decryptedMessage = new StringBuilder();

    decryptedMessage.Append("[");

    for (int i = 0; i < encryptedMessage.Length - 1; i++)
    {
        decryptedMessage.Append((char)decryptedMessage[i]);
    }
}

    encryptedMessage.RemoveAt(0);
}

```

```
        encryptedMessage.RemoveAt(encryptedMessage.Count - 1);

        foreach (BigInteger encryptedChar in encryptedMessage)
        {
            int decryptedChar = DecryptChar(encryptedChar);
            decryptedMessage.Append((char)decryptedChar);
        }

        return decryptedMessage.ToString();
    }
}
```

```
private int DecryptChar(BigInteger encrypted)
{
    BigInteger bigD = new BigInteger(d_value);
    BigInteger bigN = new BigInteger(n_value);

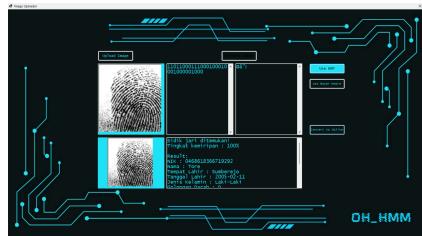
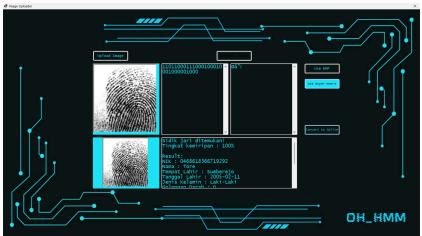
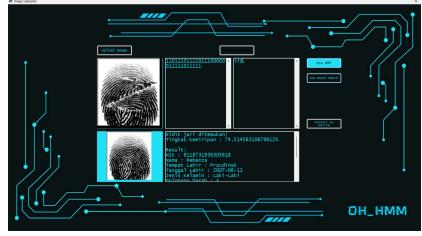
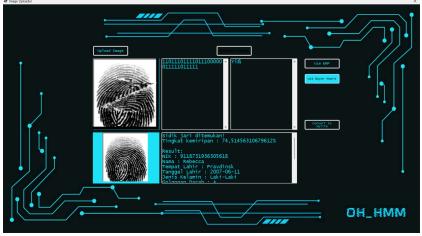
    BigInteger decrypted = this.ModPow(encrypted, bigD,
bigN);

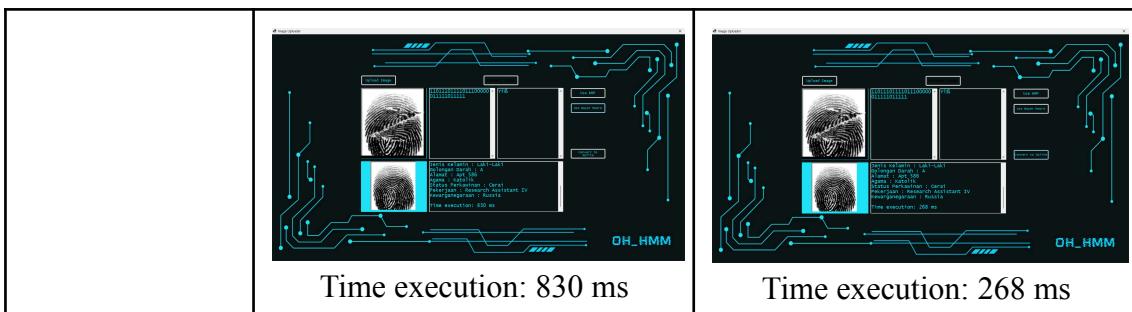
    return (int)decrypted;
}
```

## 4.2. Tata Cara Penggunaan Program

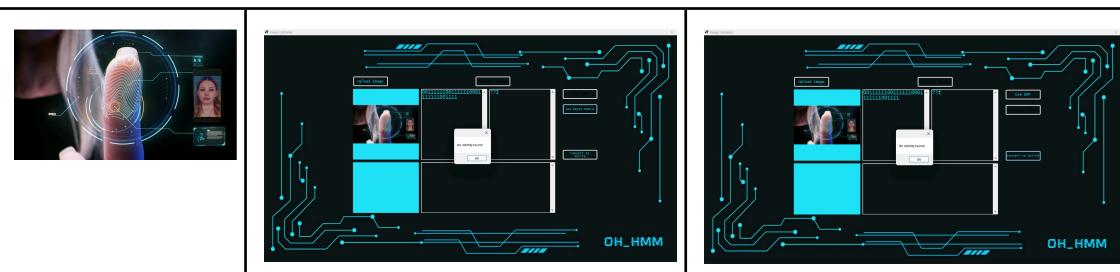
Pertama-tama pengguna dapat menekan tombol “convert to sqlite” jika ingin mengganti database. Kemudian, pengguna menekan tombol “Upload Image” untuk memilih gambar sidik jari yang akan dicari. Setelah pengguna memilih gambar, gambar tersebut akan ditunjukkan pada layar, dan tombol “Convert Image” akan muncul. Selanjutnya, pengguna menekan tombol “Convert Image” untuk mengubah gambar menjadi string binary dan ASCII. Tombol “Use KMP” dan “Use Boyer-Moore” akan muncul. Kedua tombol tersebut akan digunakan untuk memilih algoritma pencocokan string yang ingin digunakan. Aplikasi akan melakukan pencarian dan menunjukkan hasilnya setelah selesai.

## 4.3. Hasil Pengujian

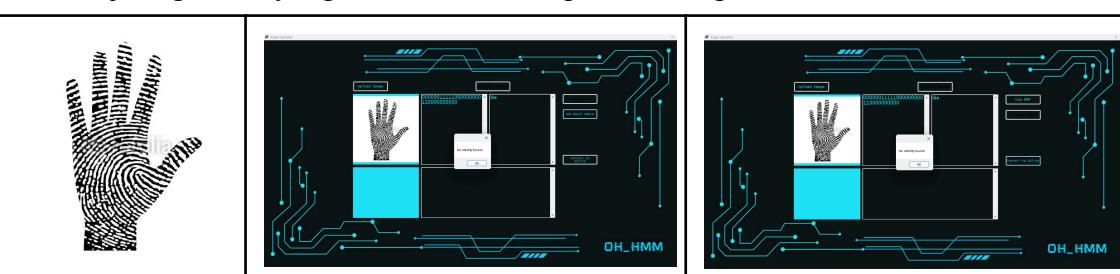
Kasus Uji	KMP	BM
Kasus uji 1: sidik jari yang dijadikan kasus uji diambil dari database sehingga seharusnya tingkat kemiripan 100%.		
	 Time execution: 380 ms	 Time execution: 281 ms
Kasus uji 2: sidik jari yang dijadikan kasus uji alter.		
		



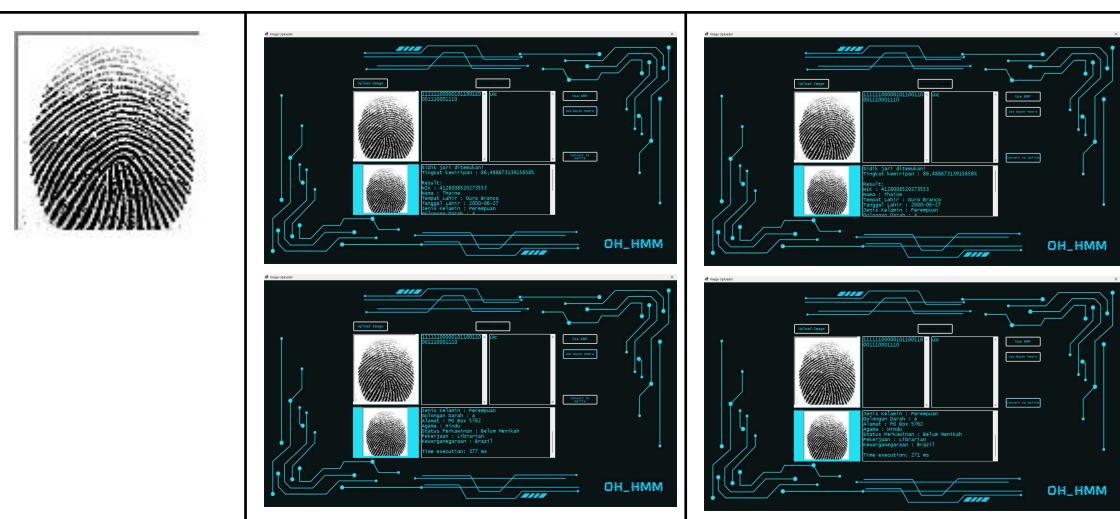
Kasus uji 3: gambar yang dimasukkan bukan gambar sidik jari.



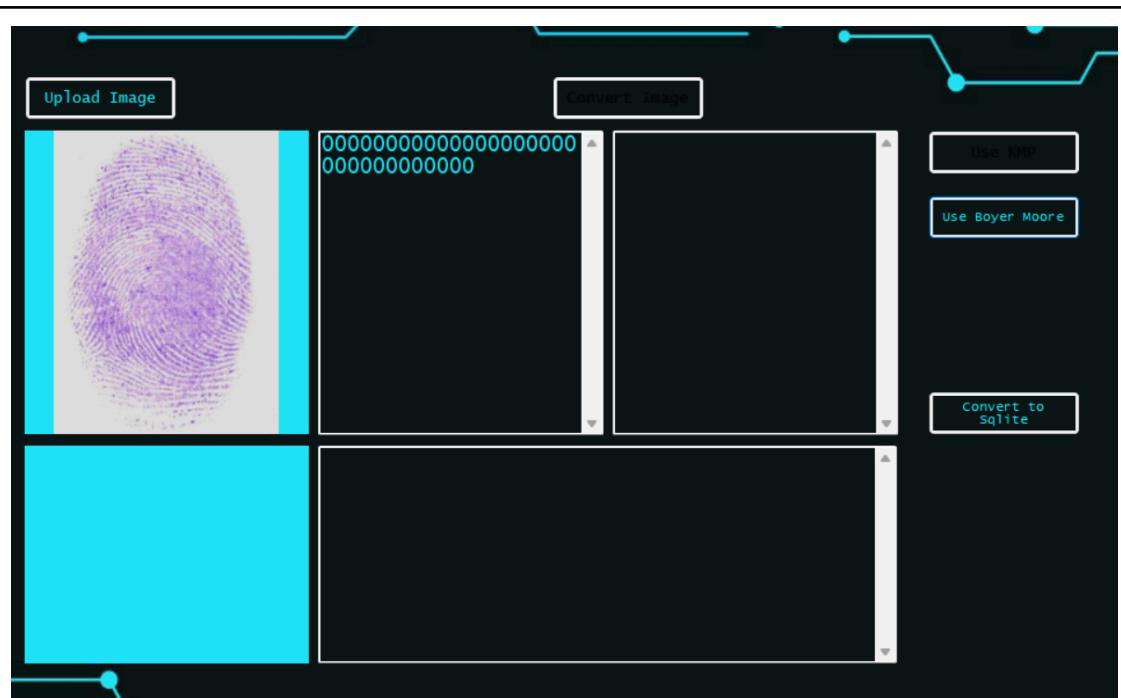
Kasus uji 3: gambar yang dimasukkan berupa sidik tangan.



Kasus uji 4: sidik jari yang dijadikan kasus uji diambil dari database namun dalam bentuk .jpg

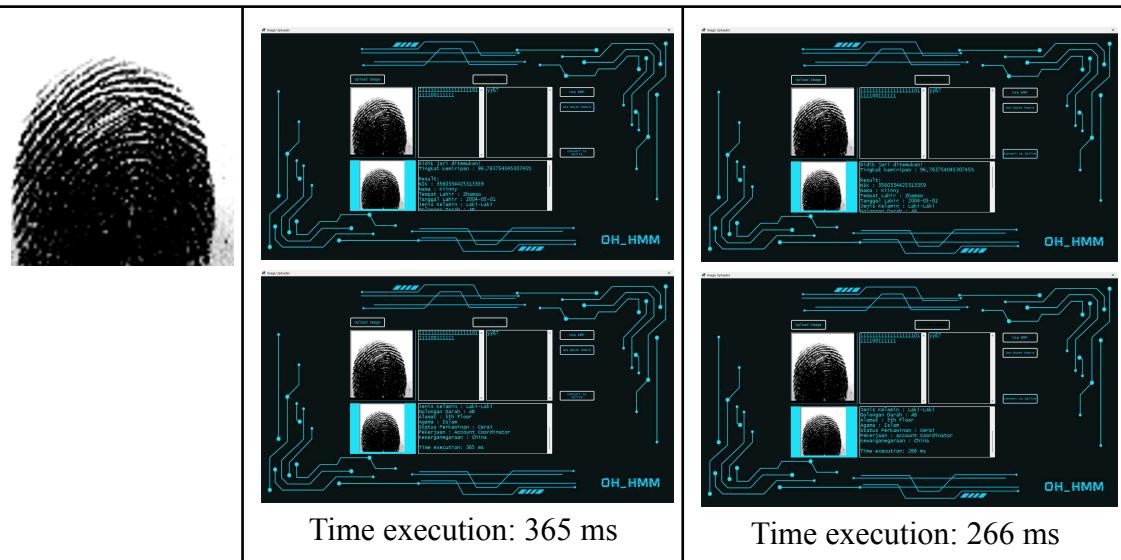


Kasus uji 5: gambar sidik jari terlalu terang

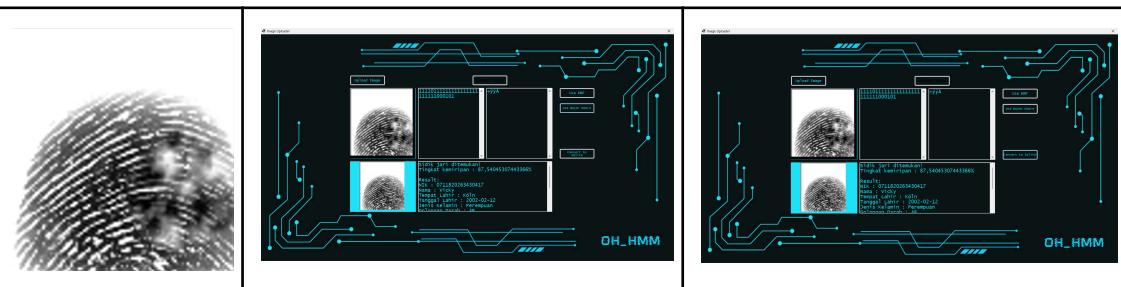


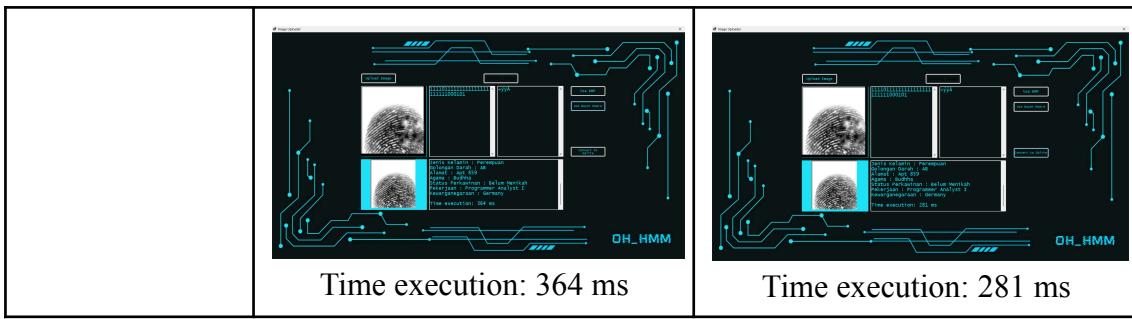
Ascii tidak muncul

Kasus uji 6: sidik jari yang dijadikan kasus uji alter.



Kasus uji 7: sidik jari yang dijadikan kasus uji alter.





#### 4.4. Analisis Hasil Pengujian

Dalam kasus sidik jari berwarna cukup gelap dan latar belakang cukup terang, sidik jari dapat tetap dicocokkan walaupun terdapat kerusakan seperti *blurring* atau potongan kecil hingga sedang.

Dalam kasus pencocokan sidik jari, algoritma BM cenderung lebih cepat karena sidik jari seringkali memiliki struktur yang kompleks dan tidak teratur. Algoritma BM dapat bekerja dengan maksimal pada kasus karakter sangat bervariasi. Kasus ini terjadi pada pencocokan sidik jari dengan menggunakan karakter ASCII karena jumlah karakter ASCII sendiri dapat dikatakan cukup besar (255 karakter). Algoritma BM melompati sejumlah besar karakter yang tidak relevan sehingga dapat melakukan lebih sedikit perbandingan secara keseluruhan.

## **BAB 5**

### **Kesimpulan**

#### **5.1. Kesimpulan**

Berdasarkan program yang kami buat, dapat disimpulkan bahwa program berjalan dengan lancar. Program dapat mengidentifikasi sidik jari seseorang secara individual kemudian mencari identitas pemilik sidik jari tersebut berdasarkan basis data pada program kami. Program dapat menggunakan algoritma KMP dan BM serta mengalkulasi tingkat kemiripan dengan menggunakan algoritma levenshtein distance. Program dapat menghasilkan database berdasarkan .sql dari pengguna. Program juga dapat mengenkripsi basis data sehingga selain melalui program ini, data yang didapatkan dari basis data tersebut tidak berarti. Dari pengujian yang dilakukan, didapatkan bahwa algoritma BM cenderung lebih cepat daripada algoritma KMP dalam kasus pencarian sidik jari.

#### **5.2. Saran**

Untuk proyek berikutnya, lebih baik jika folder src dibagi-bagi lagi menjadi folder-folder sesuai dengan fungsinya masing-masing.

#### **5.3. Refleksi**

Untuk proyek berikutnya, ada baiknya jika folder src dibagi-bagi lagi menjadi folder-folder sesuai dengan fungsinya masing-masing. Untuk program kecil, memasukkan seluruh kode di dalam folder src mungkin tidak akan menimbulkan masalah, namun jika program yang dibuat cukup besar, hal ini akan cukup membingungkan.

## Lampiran

Tautan repository lama: [https://github.com/Julian-Caleb/Tubes3\\_Oh\\_hmm.git](https://github.com/Julian-Caleb/Tubes3_Oh_hmm.git) (tidak bisa diakses)

Tautan repository baru: [https://github.com/florars/Tubes3\\_Oh\\_hmm](https://github.com/florars/Tubes3_Oh_hmm)

History commit:

Merge pull request #3 from Julian-Caleb/pixel_problem	Verified	b8d14df			
florars committed 50 minutes ago					
Merge branch 'main' of https://github.com/Julian-Caleb/Tubes3_Oh_hmm into pixel_problem		6120b15			
florars committed 52 minutes ago					
fix enkripsi		27bccaa0			
eveeeeeeee committed 1 hour ago					
add persentase		b979b39			
eveeeeeeee committed 1 hour ago					
up img res revisi		2b02a60			
eveeeeeeee committed 1 hour ago					
up img res		79cfda2			
eveeeeeeee committed 1 hour ago					
merging integrated_fixes		ef88cf7			
florars committed 2 hours ago					
fix: showing distance		ea4018c			
florars committed 2 hours ago					
feat: time exe		814bf89			
eveeeeeeee committed 3 hours ago					

Update README.md	Verified	ceca5f6			
evelynno4 committed 3 hours ago					
Update README.md	Verified	8eea09f			
evelynno4 committed 4 hours ago					
Update README.md	Verified	6729d8e			
evelynno4 committed 4 hours ago					
Create README.md	Verified	b417ba0			
evelynno4 committed 4 hours ago					
fix: db, style		24f7423			
eveeeeeeee committed 4 hours ago					
refactor: ternyata matchnya ke seluruh pixel di gambar cuy		665ede9			
florars committed 4 hours ago					
Fix?? Refactor?? entahlah : Add match percentage to results shown		42785a5			
florars committed 7 hours ago					
refactor: change results to show real name instead of alay		12ec0f1			
florars committed 10 hours ago					
refactor: String -> string		2b29ba5			
florars committed 10 hours ago					

refactor: String -> string	2b29ba5
florars committed 10 hours ago	🔗
refactor: absolute pathing to bg.jpg, matching name with chosen algorithm instead of .Equals	7390f5e
florars committed 11 hours ago	🔗
merge integrated	0a2d13b
florars committed 11 hours ago	🔗
grunt: clean uploaded image folder	1f55fb9
florars committed 11 hours ago	🔗
fix: relative pathing in matching handler when converting db paths, db location, boyer moore last occurrence, bm messagebox being shown two times	09b3675
florars committed 11 hours ago	🔗

-o- Commits on Jun 8, 2024	
styling	
eveweeweee committed yesterday	1f2d213
add result	
eveweeweee committed yesterday	b87cd77
integrate	
eveweeweee committed yesterday	8d18da3
-o- Commits on Jun 5, 2024	
fix: database	
eveweeweee committed 4 days ago	8d1f5a0
feat: sql converter (to sqlite)	
eveweeweee committed 5 days ago	949060c
-o- Commits on Jun 1, 2024	
fix: query	
eveweeweee committed last week	54e793f
-o- Commits on May 31, 2024	

-o- Commits on May 31, 2024	
fix: sql	
eveweeweee committed last week	c29b898
-o- Commits on May 25, 2024	
feat: query	
eveweeweee committed 2 weeks ago	5a1dbf3
feat: sql	
eveweeweee committed 2 weeks ago	e9327ec
-o- Commits on May 24, 2024	
feat: regex and matching algorithm buttons	
florars committed 2 weeks ago	a2bd37f
feat: process image	
eveweeweee committed 2 weeks ago	bff5d26

-o- Commits on May 23, 2024	
Merge pull request #2 from Julian-Caleb/feat/batas-persentase-kemiripan	
Julian-Caleb committed 2 weeks ago	(Verified) 4288e1f
feat: LevenshteinDistance	
Julian-Caleb committed 2 weeks ago	12b9ce6

The screenshot shows a GitHub commit history for a repository. It includes the following commits:

- Merge pull request #1 from Julian-Caleb/feat/pattern-matching** (Verified, 25b411e) - Julian-Caleb committed 2 weeks ago.
- feat: KMP dan BM** (1b01521) - Julian-Caleb committed 2 weeks ago.
- feat: RSA** (bbf4dc3) - eveveeeee committed 3 weeks ago.
- feat: Commits on May 21, 2024** (Commits on May 21, 2024)
- feat: gui for ascii text (function need to be fix)** (0823d4a) - Julian-Caleb committed 3 weeks ago.
- feat: image to binary** (d0bed1d) - Julian-Caleb committed 3 weeks ago.
- feat: GUI** (37f2dc0) - Julian-Caleb committed 3 weeks ago.

(Kami melampirkan history commit karena ada masalah dengan akun pemilik repo sehingga repo asli tidak dapat di-*public*. Kami juga memindahkan seluruh kode ke repo lain. Terima kasih atas pengertiannya <3)

## **Daftar Pustaka**

Munir, Rinaldi. 2023. Pencocokan String (String/Pattern Matching): Institut Teknologi Bandung.

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

Munir, Rinaldi. 2023. String Matching dengan Regular Expression: Institut Teknologi Bandung.

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