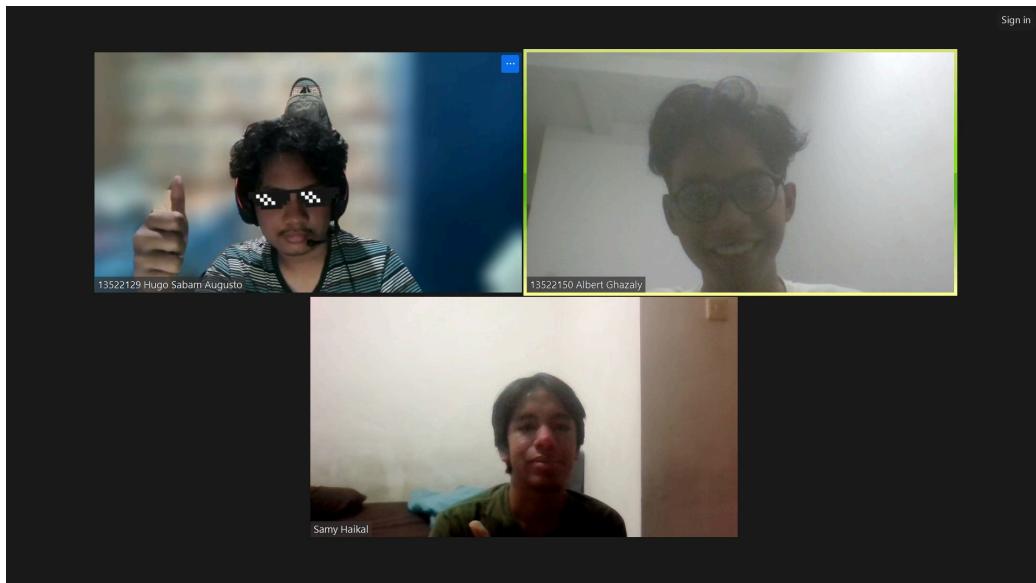


## **Tugas Besar 3 IF2211 Strategi Algoritma**

**Semester II tahun 2023/2024**

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



Oleh:

Hugo Sabam Augusto (13522129)

Albert Ghazaly (13522150)

Samy Muhammad Haikal (13522151)

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

2023

## DAFTAR ISI

<b>DAFTAR ISI</b>	<b>2</b>
<b>BAB I</b>	<b>3</b>
<b>DESKRIPSI TUGAS</b>	<b>3</b>
1.1 Deskripsi Masalah	3
<b>BAB II</b>	<b>4</b>
<b>LANDASAN TEORI</b>	<b>4</b>
2.1 Algoritma KMP	4
2.2 Algoritma BM	4
2.3 Regular Expression	5
2.4 Pengukuran Persentase Kemiripan	6
2.5 Aplikasi Desktop	6
<b>BAB III</b>	<b>7</b>
<b>ANALISIS PEMECAHAN MASALAH</b>	<b>7</b>
3.1 Langkah-langkah pemecahan masalah	7
3.1.2 KMP	8
3.1.3 BM	9
3.1.4 Hamming Distance	10
3.1.5 Regex	10
3.1.6 Basis Data	11
3.2 Fitur-fitur dari Aplikasi Desktop	11
3.3 Contoh Ilustrasi Kasus	12
3.3.1 Ilustrasi Kasus dengan Algoritma KMP dan BM	12
3.3.2 Ilustrasi Kasus dengan Hamming Distance	13
<b>BAB IV</b>	<b>15</b>
<b>IMPLEMENTASI DAN PENGUJIAN</b>	<b>15</b>
4.1 Implementasi program	15
4.2 Tata cara penggunaan program	30
4.3 Hasil Pengujian	31
4.4 Analisis Hasil Pengujian	35
<b>BAB V</b>	<b>37</b>
<b>Kesimpulan dan Saran</b>	<b>37</b>
Kesimpulan	37
Saran	37
<b>LAMPIRAN</b>	<b>38</b>
<b>DAFTAR PUSTAKA</b>	<b>39</b>

## **BAB I**

### **DESKRIPSI TUGAS**

#### **1.1 Deskripsi Masalah**

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. IF2211 Strategi Algoritma – Tugas Besar 3 1

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.

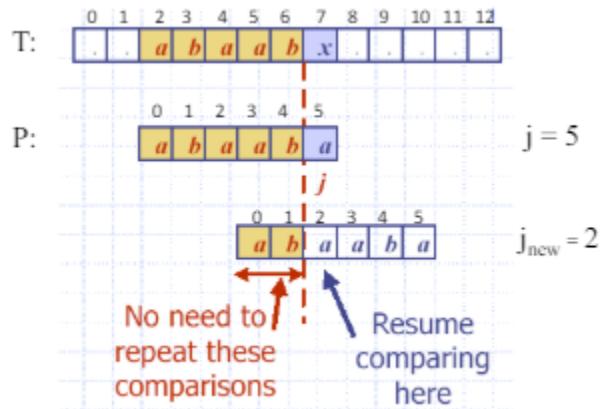
## BAB II

### LANDASAN TEORI

#### 2.1 Algoritma KMP

Algoritma Knuth-Morris-Pratt (KMP) merupakan metode algoritma pencarian string yang efisien, digunakan untuk menemukan kemunculan sebuah pola dalam teks. Algoritma ini mencari pola dalam text dari kanan ke kiri, mirip dengan brute-force namun lebih pintar. Algoritma ini lebih ‘pintar’ daripada Brute Force karena algoritma ini menghindari pencocokan ulang karakter yang telah dibandingkan sehingga lebih cepat dibanding metode Brute Force.

KMP terdiri dari dua fase utama: pertama, membangun tabel "KMP Border-Function" yang menyimpan informasi tentang panjang prefiks yang juga merupakan sufiks untuk setiap posisi dalam pola; kedua, menggunakan tabel fungsi pinggiran KMP untuk melangkah maju dalam teks utama ketika terjadi ketidakcocokan, sehingga memungkinkan pencocokan berikutnya dimulai dari posisi yang tepat tanpa mengulang perbandingan sebelumnya. Dengan cara ini, KMP mencapai kompleksitas waktu pencarian sebesar  $O(n + m)$ , di mana  $n$  adalah panjang teks dan  $m$  adalah panjang pola.

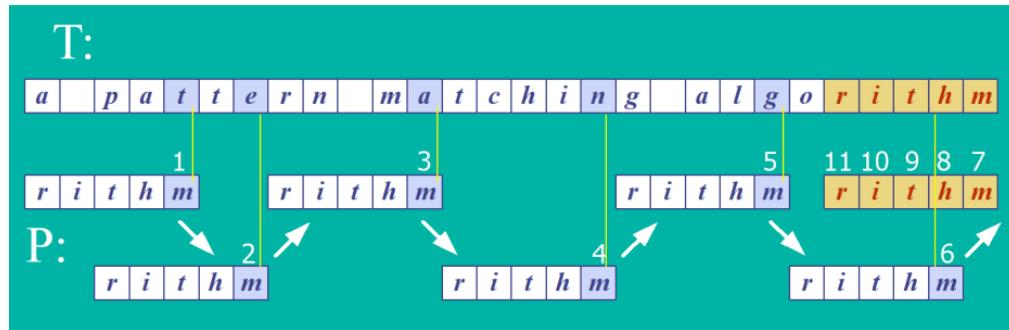


Gambar 2.1.1 Ilustrasi Algoritma KMP

#### 2.2 Algoritma BM

Algoritma Boyer-Moore adalah salah satu algoritma pencarian string yang sangat efisien, terutama digunakan untuk mencari pola dalam teks yang besar. Algoritma ini dikembangkan oleh Robert S. Boyer dan J Strother Moore pada tahun 1977. Algoritma ini dikenal karena kemampuannya untuk melompati sejumlah besar karakter dalam teks saat pencarian, yang membuatnya lebih cepat dibandingkan beberapa algoritma pencarian string lainnya seperti algoritma Naive atau Knuth-Morris-Pratt (KMP).

Berbeda dengan algoritma KMP, algoritma ini akan mencocokkan karakter dari belakang. Pertama-tama akan dibuat tabel last occurrence untuk mencatat posisi terakhir dari setiap karakter yang muncul dalam pola. Tabel ini membantu menentukan seberapa jauh pola harus digeser ketika terjadi ketidakcocokan antara karakter dalam pola dan karakter dalam teks.



Gambar 2.1.2 Ilustrasi Algoritma BM

### 2.3 Regular Expression

Regular expressions (regex) adalah alat yang sangat berguna untuk mengolah teks secara efektif dan efisien, terutama dalam hal pencarian dan manipulasi string. Regex memungkinkan pengguna untuk mendefinisikan pola pencarian yang rumit dan mencari teks yang sesuai dengan pola tersebut dalam data yang lebih besar. Berikut adalah beberapa komponen dan operator dasar dalam regex:

1. Karakter Literal: Mencocokkan karakter yang tepat dalam teks.  
Contoh: a mencocokkan 'a' dalam teks.
2. Metakarakter: Karakter khusus yang memiliki makna tertentu dalam regex.
  - Contoh: . mencocokkan sembarang karakter tunggal kecuali newline
3. Kuantifier: Menentukan berapa kali elemen sebelumnya harus muncul.
  - Contoh: a\* mencocokkan nol atau lebih kemunculan 'a'.
  - Contoh: a+ mencocokkan satu atau lebih kemunculan 'a'.
  - Contoh: a{2,4} mencocokkan antara dua hingga empat kemunculan 'a'.
4. Kelas Karakter: Sekumpulan karakter yang ingin dicocokkan.
  - Contoh: [abc] mencocokkan 'a', 'b', atau 'c'.
  - Contoh: [0-9] mencocokkan digit dari 0 hingga 9.
5. Anchor: Menentukan posisi dalam teks.
  - Contoh: ^ mencocokkan awal dari teks.
  - Contoh: \$ mencocokkan akhir dari teks.
6. Grup dan Alternasi: Mengelompokkan bagian dari regex dan mencocokkan salah satu dari beberapa pola.

- Contoh: (abc) mencocokkan 'abc' sebagai grup.
- Contoh: a|b mencocokkan 'a' atau 'b'.

## 2.4 Pengukuran Persentase Kemiripan

Untuk pengukuran persentase kemiripan, kami menggunakan metode Hamming Distance. Metode Hamming Distance digunakan untuk mengukur kemiripan dua string dengan panjang yang sama dengan cara menghitung jumlah posisi di mana karakter yang bersesuaian berbeda. Dalam konteks pengenalan sidik jari, Hamming Distance mengukur perbedaan antara dua representasi biner dari sidik jari yang dipindai. Jika hasil dari algoritma KMP atau Boyer-Moore (BM) tidak menemukan kecocokan eksak, Hamming Distance dapat memberikan ukuran seberapa dekat dua sidik jari tersebut satu sama lain, membantu menentukan kemiripan parsial atau kemungkinan kecocokan yang tidak sempurna. Dengan demikian, semakin kecil nilai Hamming Distance antara dua sidik jari, semakin besar tingkat kemiripan mereka.

## 2.5 Aplikasi Desktop

Aplikasi desktop yang kami gunakan dalam program ini merupakan aplikasi Windows Form App Project dan menggunakan bahasa pemrograman C# atau C-sharp. Aplikasi tersebut merupakan aplikasi utama yang mengandung tiga tampilan atau halaman utama, yakni MainPage (halaman utama), KMPPage (halaman pencarian metode KMP), dan BMPPage (halaman pencarian metode BM). Halaman-halaman tersebut bukan hanya program yang ada di dalam project Windows Form App ini, tetapi masih ada file-file lain yang mendukung program di balik layar. Aplikasi pada tampilan KMBPage dan BMPPage membutuhkan input gambar dari *user* dan akan menghasilkan output berupa gambar sidik jari yang termirip dan beberapa informasi lainnya yang akan dibahas dalam bagian 3.2 (Fitur-fitur dalam Aplikasi Desktop).

## BAB III

### ANALISIS PEMECAHAN MASALAH

#### 3.1 Langkah-langkah pemecahan masalah

Permasalahan utama dari tugas besar ini adalah mencari biodata seseorang berdasarkan kemiripan sidik jari yang ada di basis data dengan input yang diberikan ketika program run-time. Namun, di dalam basis data tidak ada keterhubungan yang valid antar relasi dikarenakan adanya kerusakan data sehingga nama-nama orang di dalam relasi ‘biodata’ mengalami *data corruption* menjadi nama alay.

##### 3.1.1 Konversi BMP

Tahap awal untuk menyelesaikan permasalahan ini adalah menentukan pola yang mana untuk dijadikan sebagai pola string matching yang nantinya akan digunakan di algoritma KMP maupun BM. Pertama program akan *parsing* file bitmap yang sudah terdaftar di basis data, tahapan untuk mendapatkan string yang akan dicari polanya sebagai berikut:

1. Program mengambil 1 file dengan format ekstensi bitmap (.BMP).
2. Ambil lebar dan tinggi dari image bitmap tersebut.
3. Setelah itu, tiap file bitmap dijadikan *greyscale*. Untuk tiap file bitmap yang dibuat, akan disediakan sebuah array untuk menyimpan nilai binary dari file tersebut. Lalu, iterasi tinggi dan lebar dari bitmap tersebut. Untuk tiap pixel yang diiterasi ambil R G B nya bila ada dan dihitung luminansinya dengan rumus berikut:

$$Y = 0.29 \times R + 0.587 \times G + 0.114 \times B$$

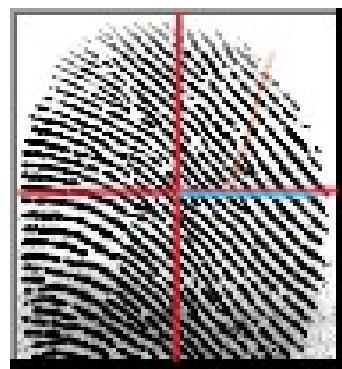
Apabila nilai Y dibawah 128, maka akan append nilai ‘1’, dan bila diatas 128 maka akan diappend nilai ‘0’.

4. Setelah didapatkan data binary untuk file bitmap tersebut. Dipotong 8 binary dan dikonversi menjadi ASCII.
5. Data ASCII dan nama path ke file bitmap tersebut akan disimpan di kelas ‘FingerString’ lalu dimasukkan ke sebuah struktur data map.

Setelah didapatkan *Map* dari kelas *FingerString* (fase ini hanya sekali saja dilakukan ketika program diinisialisasi), selanjutnya program akan *parsing* file bitmap dari input user. Proses ini akan mengambil pixel mana yang akan dijadikan pattern untuk matching sidik jari nantinya. Berikut tahapannya:

1. Program menerima file dengan format ekstensi bitmap (.BMP) dari user.

2. Ambil lebar dan tinggi dari file bitmap tersebut lalu bagi 2 untuk masing masing lebar dan tingginya
3. Konversi menjadi *greyscale* lalu ambil nilai binary nya.(Tahap ini sama dengan tahap pengambilan nilai binary seperti yang sudah dijelaskan di atas)
4. Setelah itu, akan diambil 32 binary dari file bitmap tersebut, diambil 32 agar menjadi kelipatan 8 sehingga dihasilkan 4 karakter ASCII.
5. Cek apakah hasil bagi  $2(p/2 * l/2)$  nya merupakan kelipatan dari 8 atau tidak, apabila iya maka lanjut ke tahap selanjutnya, apabila tidak maka harus dibuat offset hingga mulainya dari kelipatan 8 (Contoh dapat hasil nya 50, maka dibuat ke 48 alias kelipatan 8 terdekat). Hal ini bertujuan agar tidak ada string yang terpotong sehingga pencocokan string bisa berjalan dengan baik.
6. Terakhir ambil nilai 32 pixel dari offset yang sudah dikalkulasi lalu dikonversi menjadi 4 karakter ASCII yang nantinya digunakan sebagai pattern untuk pencocokan string di algoritma KMP maupun BM.
7. Hasil ASCII dan nama file pathnya diinstansiasi kan menjadi objek *FingerString*.



Gambar 3.1.1.1 Ilustrasi pengambilan 32 bit pattern

### 3.1.2 KMP

Setelah Pre-processing seluruh file bitmap yang tersedia dan file bitmap dari input, User bisa memilih algoritma KMP untuk pencocokan sidik jari. Tahap pencocokan string dengan KMP diawali dengan memasukkan map *FingerString* yang sudah dikonversi dan *FingerString* pattern dari input.

1. KMP dimulai dengan inisialisasi fungsi border untuk pola pencarian. Ini dilakukan dengan mengiterasi melalui pola dan mengidentifikasi prefix yang juga merupakan suffix terpanjang pada setiap posisi dalam pola. Proses ini dilakukan

menggunakan dua indeks, idx dan j, yang bertanggung jawab untuk memeriksa kesamaan karakter di antara prefix dan suffix.

2. Setelah inisialisasi fungsi border, algoritma melakukan pencarian kemiripan antara sidik jari input dan setiap sidik jari dalam basis data. Ini dilakukan dengan menggunakan indeks i untuk sidik jari input dan j untuk sidik jari dalam basis data. Saat melakukan perbandingan karakter, jika terjadi ketidakcocokan, indeks j akan digeser mundur sesuai dengan nilai dari fungsi border yang telah dihitung sebelumnya. Proses ini akan berlanjut hingga entri dalam basis data habis diperiksa atau sidik jari input cocok dengan pola sidik jari dalam basis data.
3. Jika terdapat kemiripan antara sidik jari input dan salah satu sidik jari dalam basis data, sidik jari yang cocok tersebut akan ditambahkan ke dalam daftar hasil pencarian. Selain itu, boolean *isFound* akan diatur menjadi true untuk menandakan bahwa setidaknya ada satu kemiripan telah ditemukan.

### 3.1.3 BM

User juga bisa memilih algoritma BM untuk pencocokan sidik jari. Tahap pencocokan string dengan BM diawali dengan memasukkan map *FingerString* yang sudah dikonversi dan *FingerString* pattern dari input.

1. BM dimulai dengan inisialisasi tabel last occurrence untuk pola pencarian. Ini dilakukan dengan mengiterasi melalui pola dan mencatat kemunculan terakhir dari setiap karakter pada pola.
2. Setelah inisialisasi tabel last occurrence, algoritma melakukan pencarian kemiripan antara sidik jari input dan setiap sidik jari dalam basis data. Ini dilakukan dengan menggunakan indeks i untuk sidik jari input dan j untuk sidik jari dalam basis data. Saat melakukan perbandingan karakter, jika terjadi ketidakcocokan, indeks j akan digeser mundur sesuai dengan nilai dari tabel last occurrence yang telah dihitung sebelumnya. Proses ini akan berlanjut hingga entri dalam basis data habis diperiksa atau sidik jari input cocok dengan pola sidik jari dalam basis data.
3. Jika terdapat kemiripan antara sidik jari input dan salah satu sidik jari dalam basis data, sidik jari yang cocok tersebut akan ditambahkan ke dalam daftar hasil pencarian. Selain itu, boolean *isFound* akan diatur menjadi true untuk menandakan bahwa setidaknya ada satu kemiripan telah ditemukan.

### 3.1.4 Hamming Distance

Metode Hamming Distance digunakan apabila algoritma KMP maupun BM tidak menemukan pola yang cocok dari string ASCII sidik jari. Cara kerja metode Hamming Distance disini adalah :

1. cek dulu apakah jumlah ASCII sama antara sidik jari input dengan sidik jari yang dibandingkan. Apabila tidak, maka next ke sidik jari selanjutnya(di map *FingerString*).
2. Apabila panjang sama, maka akan iterasi sebanyak panjang salah satu sidik jari *FingerString* lalu increment nilai perbedaan apabila karakter pada iterasi tersebut sama
3. Setelah iterasi sebanyak n kali, total perbedaan dibagi dengan n lalu kali 100 %.
4. Apabila hasil persentase diatas 70%, *FingerString* akan dimasukkan ke dalam list (List yang berisi semua yang melewati *threshold* 70%).

Karena di program ini hanya menampilkan yang paling mirip, maka apabila list hasil memiliki hasil lebih dari 1, list akan di sort dan didapatkan hasil dengan persentase terbesar. Alasan mengapa kami memasang threshold diatas 70%, karena setelah berkali kali percobaan, kami menggunakan beberapa sample sidik jari yang ‘korup’ dan 70% merupakan hasil yang memiliki output valid terbanyak dibandingkan persentase lain



Gambar 3.1.4.1 Contoh sidik jari rusak/korup

### 3.1.5 Regex

Regex digunakan untuk mencocokkan nama pada tabel biodata dan sidik jari. Hal ini perlu dilakukan karena data pada biodata memiliki kemungkinan untuk korup. Sebuah data yang korup dapat memiliki berbagai macam bentuk. Pada tugas ini, jenis data korup adalah bahasa alay Indonesia. Pada tugas ini kita diminta untuk menangani kombinasi dari tiga buah variasi bahasa alay, yaitu kombinasi huruf besar-kecil, penggunaan angka, dan penyingkatan. Contoh kasus bahasa alay dijelaskan dengan detail sebagai berikut.

Kata Orisinal	Variasi Alay
---------------	--------------

Edwin Carlson	eDw1n crlSN
Rebecca Green	rb3cc 6rn
Tristan May	tr5Tn m4y

### 3.1.6 Basis Data

Database yang dibuat adalah MySQL Database yang berisi data yang dihasilkan menggunakan pustaka Faker dari Python. Data palsu yang realistik ini disimpan dalam database dengan nama "stima3". Server database berada di localhost, memungkinkan aplikasi untuk mengakses data secara lokal. Autentikasi untuk mengakses database, seperti username, password, dan port, dapat disesuaikan sesuai kebutuhan pengguna dan dapat diatur melalui program yang akan mengaksesnya. Integrasi dengan MySQL Client memastikan pengguna dapat dengan mudah mengelola database, menjalankan perintah SQL, dan melakukan operasi basis data lainnya dengan fleksibilitas yang tinggi. Ini memberikan pengguna kendali penuh atas konfigurasi dan penggunaan database sesuai kebutuhan mereka.

## 3.2 Fitur-fitur dari Aplikasi Desktop

Fitur-Fitur di dalam aplikasi desktop milik kami menggunakan tiga tampilan utama yang menunjang fungsionalitas program ini. Fitur-fitur tersebut mengimplementasikan fungsionalitas berupa program-program parsial kecil dan disatukan menjadi sebuah fitur yang siap dijalankan. Tiga halaman utama tersebut antara lain

1. MainPage
  - Button penghubung dari MainPage ke BMPPage
  - Button penghubung dari MainPage ke KMBPage
  - Input informasi database user
  - Update informasi database berdasarkan input dari user
  - *Load* seluruh gambar dari folder *sample* dan memprosesnya
2. BMPPage
  - Button penghubung dari BMPPage ke MainPage
  - Button *input* gambar dari user untuk dicari
  - Button *clear* gambar dari user jika tidak jadi dicari
  - Button *search* untuk mencari gambar berdasarkan *input* gambar dengan metode BM atau Hamming
  - Button *restart* untuk me-refresh dari awal

- Panel gambar sidik jari hasil pencarian
  - Box informasi mengenai identitas pemilik sidik jari
  - Durasi pencarian dalam ms
  - Kemiripan pencarian dalam persen
3. KMBPage
- Button penghubung dari MBPage ke MainPage
  - Button *input* gambar dari user untuk dicari
  - Button *clear* gambar dari user jika tidak jadi dicari
  - Button *search* untuk mencari gambar berdasarkan *input* gambar dengan metode KMB atau Hamming
  - Button *restart* untuk me-refresh dari awal
  - Panel gambar sidik jari hasil pencarian
  - Box informasi mengenai identitas pemilik sidik jari
  - Durasi pencarian dalam ms
  - Kemiripan pencarian dalam persen

### 3.3 Contoh Ilustrasi Kasus

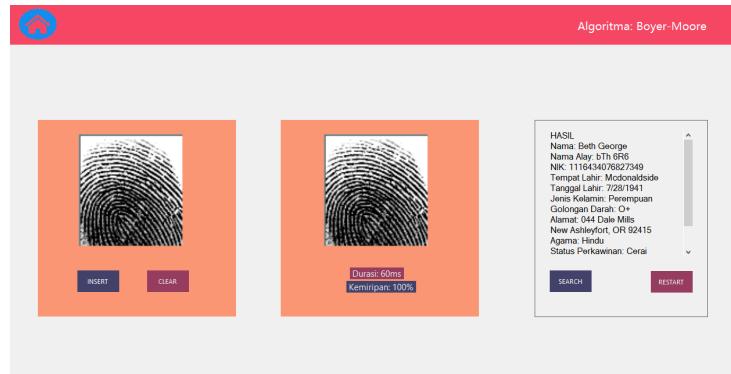
Sebagai contoh kasus, berikut merupakan ilustrasi pencocokan sidik jari pada program kami. Kami akan mengambil 1 contoh sidik jari yang akan dijadikan sample percobaan. Berikut sidik jarinya



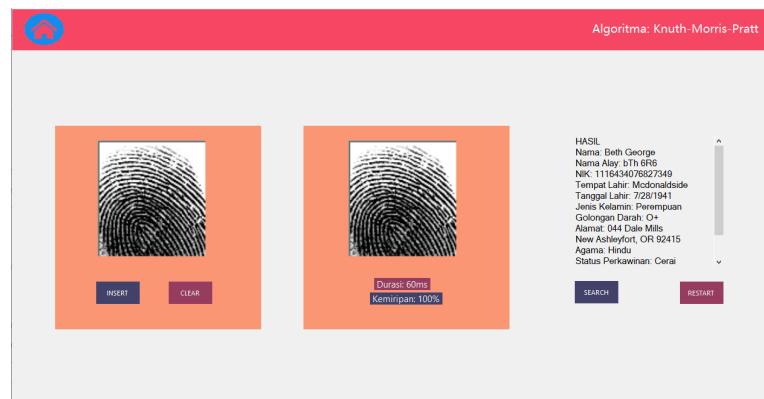
Gambar 3.3.1 Sidik jari sampel

#### 3.3.1 Ilustrasi Kasus dengan Algoritma KMP dan BM

Pertama program akan meminta user algoritma apa yang mau digunakan untuk pencarian sidik jari. Setelah itu, masukkan file bitmap sidik jari sampel di dalam page KMP atau BM dengan menekan tombol ‘insert’ , lalu tekan tombol ‘search’ untuk melakukan pencarian sidik jari yang cocok dengan sampel yang digunakan



Gambar 3.3.1.1 Tangkapan layar hasil pencarian dengan BM

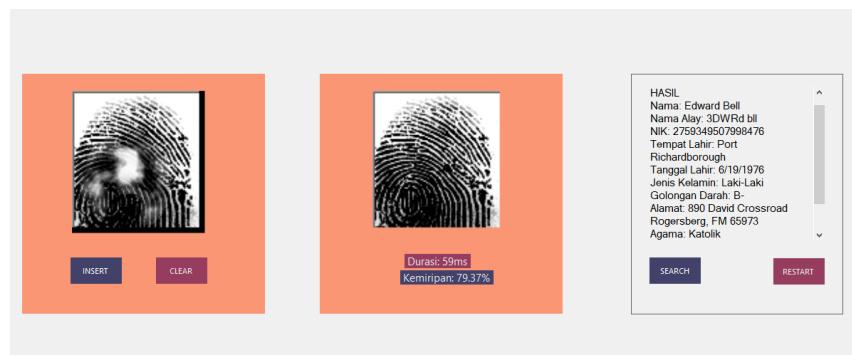


Gambar 3.3.1.2 Tangkapan layar hasil pencarian dengan KMP

Dalam durasi tertentu, program akan menampilkan biodata yang bersangkutan dengan sidik jari sampel. Mulai dari nama alay, nama asli, NIK, tempat lahir, tanggal lahir, jenis kelamin, golongan darah, alamat, agama, status perkawinan, pekerjaan dan kewarganegaraan.

### 3.3.2 Ilustrasi Kasus dengan Hamming Distance

Untuk kasus ini, Metode Hamming Distance akan digunakan apabila kedua algoritma baik KMP maupun BM tidak menemukan hasil yang eksak. Untuk kasus ini, saya akan menggunakan sidik jari ‘rusak’ sebagai input agar hamming distance bisa berjalan



*Gambar 3.3.2.1 Tangkapan layar dengan Hamming Distance*

Dari Tangkapan layar diatas, didapatkan hasil dengan kemiripan 79.37%. Cukup terlihat bahwa sidik jari di atas sama

## BAB IV

### IMPLEMENTASI DAN PENGUJIAN

#### 4.1 Implementasi program

No	Kelas	Deskripsi
1	BitmapParserBuilder	Kelas ini bertanggungjawab untuk mengubah input berupa file bitmap menjadi serangkaian kode ascii
<b>ATTRIBUT</b>		
<pre>private int left_x; private int left_y; private int right_x; private int right_y; private string _directoryPath; private System.Text.StringBuilder _bitStringBuilder; private System.Text.StringBuilder _asciiStringBuilder; private Dictionary&lt;int, FingerString&gt; _asciiMap;</pre>		
<b>METHOD</b>		
<pre>public BitmapParserBuilder(string directoryPath) {     _directoryPath = directoryPath;     left_x = 0;     left_y = 0;      // Inisialisasi ukuran gambar pertama di direktori atau     file     InitializeImageSize(directoryPath);      _bitStringBuilder = new System.Text.StringBuilder();     _asciiStringBuilder = new System.Text.StringBuilder();     _asciiMap = new Dictionary&lt;int, FingerString&gt;();     Console.WriteLine(\$"H : {right_y - left_y} L: {right_x -     left_x}"); }  public BitmapParserBuilder(string directoryPath, int offsetX, int offsetY) {     _directoryPath = directoryPath;      InitializeImageSize(directoryPath);     //Sample selalu diambil height / 2 dan width bagi 2</pre>		

```

        left_x = right_x / 2;
        left_y = right_y / 2;
        int startbit = right_x * (left_y - 1) + left_x;
        Console.WriteLine($"Sample diambil harus kelipatan 8
bit!, sekarang diambil mulai dari bit ke-{startbit} bit atau
{right_x} * {left_y - 1} + {left_x}");
        //Handling kalo image yang diinput mulai samplenya bukan
kelipatan 8
        if (startbit % 8 != 0)
        {
            Console.WriteLine("tidak kelipatan 8 bit!, tidak akan
berhasil"); // harus diubah
            Console.WriteLine($"harus geser {startbit%8} bit");
            left_x -= startbit % 8;
        }
        right_x = left_x + offsetX;
        right_y = left_y + offsetY;
        Console.WriteLine($"Diambil pixel dari Width {left_x} ke
{right_x} dan Height dari {left_y} ke {right_y - 1}");

        _bitStringBuilder = new System.Text.StringBuilder();
        _asciiStringBuilder = new System.Text.StringBuilder();
        _asciiMap = new Dictionary<int, FingerString>();
        Console.WriteLine($"H : {right_y - left_y} L: {right_x -
left_x}");
    }

private void InitializeImageSize(string path)
{
    if (File.Exists(path))
    {
        // The path is a single file
        using (Image<Rgba32> image =
Image.Load<Rgba32>(path))
        {
            right_x = image.Width;
            right_y = image.Height;
        }
    }
    else if (Directory.Exists(path))
    {
        // The path is a directory
        string[] bmpFiles = Directory.GetFiles(path,
"*.BMP");
        if (bmpFiles.Length > 0)
        {
            using (Image<Rgba32> image =
Image.Load<Rgba32>(bmpFiles[0]))
            {
                right_x = image.Width;
                right_y = image.Height;
            }
        }
    }
}

```

```

        }
    }
    else
    {
        throw new FileNotFoundException("No BMP files
found in the specified directory.");
    }
}
else
{
    throw new FileNotFoundException("The specified path
is neither a valid file nor a directory.");
}
}

public FingerString getFirstFingerString() {
    return this.AsciiMap[1];
}
public void PrintBinaryAll()
{
    Console.WriteLine(this._directoryPath);

    if (File.Exists(_directoryPath))
    {
        // Process single file
        ProcessSingleFile(_directoryPath);
    }
    else
    {
        // Process directory
        string[] bmpFiles =
Directory.GetFiles(_directoryPath, "*.BMP");

        foreach (string filePath in bmpFiles)
        {
            //Console.WriteLine($"Processing file:
{Path.GetFileName(filePath)}");
            using (Image<Rgba32> image =
Image.Load<Rgba32>(filePath))
            {
                PrintbinaryImage(image);
            }
        }
    }
}

public void ParseMapAscii()
{
    Console.WriteLine(this._directoryPath);
    int fileId = 1; // Nomor identifikasi file
}

```

```

        if (File.Exists(_directoryPath))
        {
            // Process single file
            ProcessSingleFile(_directoryPath, fileId);
        }
        else
        {
            // Process directory
            string[] bmpFiles =
Directory.GetFiles(_directoryPath, "*.BMP");

            foreach (string filePath in bmpFiles)
            {
                ProcessSingleFile(filePath, fileId);
                fileId++;
            }
        }
    }

public void clearNullChar()
{
    foreach (var item in this._asciiMap)
    {
        if (item.Value.AsciiString.Contains('\0'))
        {
            item.Value.AsciiString =
item.Value.AsciiString.Replace('\0', '\x00');
        }
    }
}

private void ProcessSingleFile(string filePath, int fileId = 1)
{
    //Console.WriteLine($"Processing file:
{Path.GetFileName(filePath)}");
    using (Image<Rgba32> image =
Image.Load<Rgba32>(filePath))
    {
        //PrintbinaryImage(image);
        //Console.WriteLine($"H : {right_y - left_y} L:
{right_x - left_x}");
        string bitString = ParseBits(image);
        ConvertBitsToAscii();
        string asciiString = _asciiStringBuilder.ToString();

        // Tambahkan hasil print ASCII ke dalam map
        _asciiMap[fileId] = new
FingerString(Path.GetFileNameWithoutExtension(filePath),
asciiString);
    }
}

```

```

private void PrintbinaryImage(Image<Rgba32> image) // buat
milih pixel yg bagus
{
    for (int x = left_x; x < right_x; x++)
    {
        Console.WriteLine(x % 10);
    }
    Console.WriteLine();
    using (var clone = image.CloneAs<Rgba32>())
    {
        for (int y = left_y; y < right_y; y++)
        {
            Console.Write($"{y} ");
            for (int x = left_x; x < right_x; x++)
            {
                Rgba32 pixel = clone[x, y];
                double luminance = CalculateLuminance(pixel);

                if (luminance > 128)
                {
                    Console.Write("0");
                }
                else
                {
                    Console.Write("1");
                }
            }
            Console.WriteLine();
        }
    }
}

private string ParseBits(Image<Rgba32> image)
{
    using (var clone = image.CloneAs<Rgba32>())
    {
        _bitStringBuilder.Clear(); // Clear existing data
        for (int y = left_y; y < right_y; y++)
        {
            for (int x = left_x; x < right_x; x++)
            {
                Rgba32 pixel = clone[x, y];
                double luminance = CalculateLuminance(pixel);

                if (luminance > 128)
                {
                    _bitStringBuilder.Append("0");
                }
                else
                {

```

```

                _bitStringBuilder.Append("1");
            }
        }

        return _bitStringBuilder.ToString();
    }
}

private void ConvertBitsToAscii()
{
    asciiStringBuilder.Clear(); // Clear existing data
    for (int i = 0; i < _bitStringBuilder.Length; i += 8)
    {
        string byteString =
    _bitStringBuilder.ToString().Substring(i, Math.Min(8,
    _bitStringBuilder.Length - i));
        if (byteString.Length < 8)
        {
            byteString = byteString.PadRight(8, '0');
        }
        int asciiValue = Convert.ToInt32(byteString, 2);
        _asciiStringBuilder.Append((char)asciiValue);
    }
}

private double CalculateLuminance(Rgba32 color)
{
    return 0.299 * color.R + 0.587 * color.G + 0.114 *
color.B;
}

public void PrintAllMap()
{
    Console.WriteLine("Printing ASCII Map:");
    foreach (var entry in _asciiMap)
    {
        Console.Write($"Key : {entry.Key} | ");
        entry.Value.displayData();
        Console.WriteLine();
    }
}

public void PrintMap(int id)
{
    Console.WriteLine("Print ASCII map id " + id);
    if (_asciiMap.ContainsKey(id))
    {
        FingerString value = _asciiMap[id];
        value.displayData();
    }
}

```

	<pre>         else         {             Console.WriteLine("Key not found in the map.");         }     } } </pre>	
2	KMP	Kelas ini bertanggungjawab untuk melakukan pencarian sidik jari dengan menggunakan algoritma KMP
<b>ATTRIBUT</b>		
<pre> private Dictionary&lt;int, FingerString&gt; fingermap; private FingerString inputF; private List&lt;int&gt; borderfunc; private List&lt;FingerString&gt; resultmatch; private bool isFound; </pre>		
<b>METHOD</b>		
<pre> public KMP(Dictionary&lt;int, FingerString&gt; fm, FingerString inputF) {     this.fingermap = fm;     this.inputF = inputF;     this.borderfunc = Enumerable.Repeat(0, inputF.AsciiString.Length).ToList();     this.isFound = false;     this.resultmatch = new List&lt;FingerString&gt;();  }  public void searchKMP() {     initBorderFunc();     foreach (var entry in fingermap) {          string source = inputF.AsciiString;         string target = entry.Value.AsciiString;         //Console.WriteLine(\$" Source : {source} Target : {target}");         int i = 0;         int j = 0;         while(j &lt; borderfunc.Count &amp;&amp; i &lt; target.Length)         {             //Console.WriteLine(\$"i = {i} dan j = {j}   banding {source[j]} dengan {target[i]}" ); </pre>		

```
        if (source[j] == target[i])
        {
            i++;
            j++;
        }
        else
        {
            if (source[j] != target[i] && j > 0)
            {
                j = borderfunc[j - 1];
            }
            else// j<=0
            {
                j = 0;
                i++;
            }
        }
    }

    if(j != borderfunc.Count)
    {
    }
    else
    {
        resultmatch.Add(entry.Value);
        isFound = true;
    }
}

printHasil();

}

public void printHasil()
{
    Console.WriteLine("=====KMP=====");
    if (this.isFound)
    {
        Console.WriteLine("Hasil yang ditemukan:");
        foreach (var entry in resultmatch)
        {
            Console.WriteLine($"Filename : {entry.FileName}");
        }
    }
    else
    {
        Console.WriteLine("Tidak ada hasil yang ditemukan!");
    }
}
```

```
        }
    }

public void initBorderFunc()
{
    string word = inputF.AsciiString;
    int idx = 0;
    int streak = 0;
    int j = 1;
    while(j < word.Length)
    {
        if (word[idx] == word[j])
        {
            Console.WriteLine(j);
            idx++;
            j++;
            streak++;
            borderfunc[j-1] = streak;

        }
        else if (word[idx] != word[j] && idx > 0)
        {
            streak = 0;
            idx = borderfunc[idx - 1];

        }
        else
        {
            streak = 0;
            idx = borderfunc[0];
            j++;

        }
    }

}

//PrintBorderFunc();

}

public void PrintBorderFunc()
{
    Console.WriteLine("Border Function Values:");
    for (int i = 0; i < borderfunc.Count; i++)
    {
        Console.WriteLine($"borderfunc[{i}] = {borderfunc[i]}");
    }
}
```

3	<p>BM</p> <p>Kelas ini bertanggungjawab untuk melakukan pencarian sidik jari dengan menggunakan algoritma BM</p>
<b>ATTRIBUTE</b>	
<pre>private Dictionary&lt;int, FingerString&gt; fingermap; private FingerString inputF; private int[] LastOcc; private List&lt;FingerString&gt; resultmatch; private bool isFound;</pre>	
<b>METHOD</b>	
<pre>public void printHasil() {     Console.WriteLine("=====BM=====");     if (this.isFound)     {         Console.WriteLine("Hasil yang ditemukan:");         foreach (var entry in resultmatch)         {             Console.WriteLine(\$"Filename : {entry.FileName}");         }     }     else     {         Console.WriteLine("Tidak ada hasil yang ditemukan!");     } }  public BM(Dictionary&lt;int, FingerString&gt; fm, FingerString inputF) {     this.fingermap = fm;     this.inputF = inputF;     this.LastOcc = BuildLastOccurrence(inputF.AsciiString);     this.resultmatch = new List&lt;FingerString&gt;();     this.isFound = false; }  private int[] BuildLastOccurrence(string source) {     int[] table = new int[256]; // Assume ASCII character set     for (int i = 0; i &lt; table.Length; i++)         table[i] = source.Length;</pre>	

```

        for (int i = 0; i < source.Length; i++)
            table[source[i]] = i;
        return table;
    }

public void searchBM()
{
    // int x =0;
    foreach (var entry in fingermap) {
        // Console.WriteLine(x);
        // x++;
        string source = inputF.AsciiString;
        string target = entry.Value.AsciiString;
        // Console.WriteLine(inputF.FileName);
        // Console.WriteLine(entry.Value.FileName);
        //Console.WriteLine($" Source : {source} Target :
{target}");
        int m = source.Length;
        int n = target.Length;
        int i = m-1;

        if(i > n-1){
            Console.WriteLine("source lebih panjang dari
target!");
            continue;
        }
        int j = m-1;
        // int a = 0;
        do {
            // Console.WriteLine($"Comparing: {source[j]}
{target[i]}");
            // if(a>2){
            //     break;
            // }
            // a++;

            if (source[j] == target[i])
            {
                if (j == 0)
                {
                    //Console.WriteLine($"ditemukan di index
{i}");
                    this.isFound = true;
                    resultmatch.Add(entry.Value);
                    break;
                }
                else
                { // looking-glass technique
                    i--;
                    j--;
                }
            }
        }
    }
}

```

```

        }
        else { // character jump technique
            int lo = LastOcc[target[i]]; //last occ
            i = i + m - Math.Min(j, 1+lo);
            j = m - 1;
        }
    } while (i <= n-1);
    continue; // no matches
}
printHasil();
}

```

4	Hamming	Kelas ini bertanggungjawab untuk melakukan pencocokan string dengan algoritma Hamming jika match masih belum ditemukan saat pencarian dengan metode KMP/BM
---	---------	--

#### ATTRIBUTE

```

private Dictionary<int, FingerString> fingermap;
private FingerString inputF;
private List<Result> goodResults;
private int tuningPersen;

```

#### METHOD

```

public void searchHamming()
{
    foreach (var entry in fingermap)
    {
        if (entry.Value.AsciiString.Length ==
inputF.AsciiString.Length)
        {
            int totlength = entry.Value.AsciiString.Length;
            int different = 0;

            for (int i = 0; i < totlength; i++)
            {
                if (entry.Value.AsciiString[i] ==
inputF.AsciiString[i])
                {
                    different++;
                }
            }

            double percent = ((double)different / totlength)
* 100;
    }
}

```

```

        if (percent > this.tuningPersen)
        {
            goodResults.Add(new Result(entry.Value,
percent));
        }
    }
else
{
    Console.WriteLine($"Length difference, cannot
calculate Hamming distance between {entry.Value.FileName} and
{inputF.FileName}");
}
}
//sort
goodResults.Sort((x, y) =>
y.Percentage.CompareTo(x.Percentage));
}

public void writeResult()
{
    Console.WriteLine("====RESULTS ABOVE THRESHOLD====");
    foreach (var result in goodResults)
    {
        Console.WriteLine($"FileName:
{result.FingerString.FileName}, Percentage:
{result.Percentage}%");
    }
}
public FingerString getBestResult()
{
    return goodResults.First().FingerString;
}

```

5	<p>AlayMatcher -Static Class-</p> <p><b>ATRIBUT</b></p> <p>-</p> <p><b>METHOD</b></p>	Kelas ini bertanggungjawab untuk melakukan pencocokan nama yang korup menggunakan regex
---	---	---

```

public static bool AlayMatch(string normal, string alay)
{
    string pattern = @"";
    var numberMap = new Dictionary<char, string>
    {
        { 'A', "4" },
        { 'E', "3" },
        { 'G', "6" },
        { 'I', "1" },
        { 'O', "0" },
        { 'S', "5" },
        { 'Z', "2" }
    };
    string vowels = "aeiouAEIOU";
    foreach (char c in normal)
    {
        if(numberMap.ContainsKey(char.ToUpper(c)))
        {
            if(vowels.Contains(c))
            {

pattern+=${: ${c} | ${numberMap[char.ToUpper(c)]}} | ${(char.IsUpper(c) ? char.ToLower(c) : char.ToUpper(c))}?";
            }
            else
            {

pattern+=${: ${c} | ${numberMap[char.ToUpper(c)]}} | ${(char.IsUpper(c) ? char.ToLower(c) : char.ToUpper(c))}";
            }
            else if(vowels.Contains(c)){
                pattern+=${{ ${c}} | ${(char.IsUpper(c) ? char.ToLower(c) : char.ToUpper(c))}}?";
            }
            else
            {
                pattern+=${{ ${c}} | ${(char.IsUpper(c) ? char.ToLower(c) : char.ToUpper(c))}}";
            }
        }
        //Console.WriteLine($"Pattern : {pattern}");

        Regex regex = new Regex(pattern);
        Match match = regex.Match(alay);
        if (match.Success)
        {
            Console.WriteLine($"Found match: {match.Value}");
            return true;
        }
    }
}

```

	<pre>         {             //Console.WriteLine("No match found.");             return false;         }     } </pre>	
6	FingerString	Kelas ini bertanggung jawab untuk menyimpan data nama file path dan ASCII untuk sidik jari bitmap
<b>ATRIBUT</b>		
<pre> public string FileName { get; set; } public string AsciiString { get; set; } </pre>		
<b>METHOD</b>		
<pre> public FingerString(string filename) {     FileName = filename;     AsciiString = filename; } public FingerString(string fileName, string asciiString) {     FileName = fileName;     AsciiString = asciiString; } public void displayData() {     Console.WriteLine(\$" Filename : {FileName} ");     Console.WriteLine(\$" ASCII : {AsciiString} ");     Console.WriteLine(\$"Banyaknya Char : {AsciiString.Length}"); } </pre>		
7	Result	Kelas ini bertanggung jawab untuk menyimpan data Fingerstring dan persentase kecocokan dari hasil pencarian menggunakan hamming distance
<b>ATRIBUT</b>		
<pre> public FingerString FingerString { get; } public double Percentage { get; } </pre>		

	METHOD
	<pre>public Result(FingerString fingerString, double percentage) {     this.FingerString = fingerString;     this.Percentage = percentage; }</pre>

#### 4.2 Tata cara penggunaan program

Program digunakan dengan terlebih dahulu melakukan clone terhadap repository GitHub pada link [berikut](#) (terlampir juga pada Lampiran). Silakan masukkan perintah berikut pada terminal.

```
git clone https://github.com/AlbertGhazaly/Tubes3_Barbarians.git
```

Setelah itu, lakukan import file ‘final2.sql’ di dalam folder ‘db’ ke dalam MySQL localhost anda dengan nama database ‘stima3’

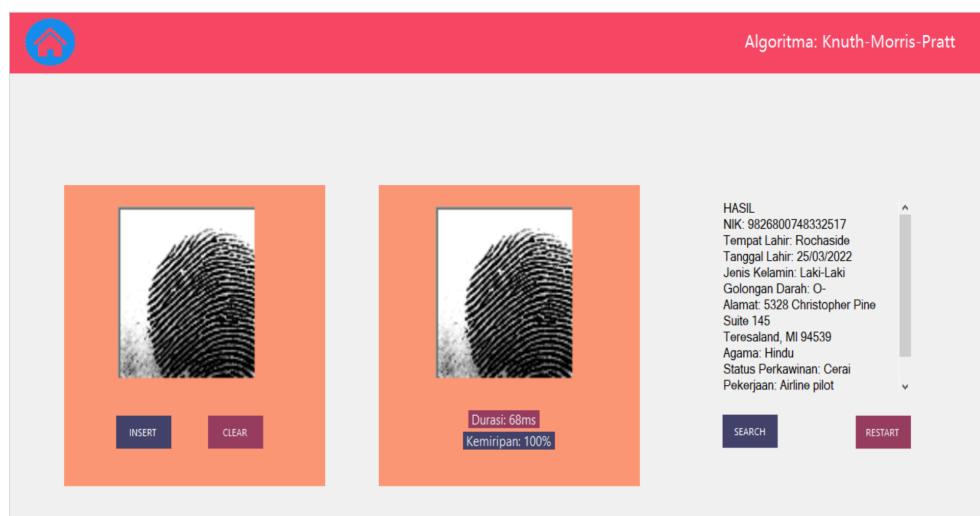
Lalu run ‘Barbarians.exe’ di directory berikut:

```
cd src\Barbarians\bin\Release
```

#### 4.3 Hasil Pengujian

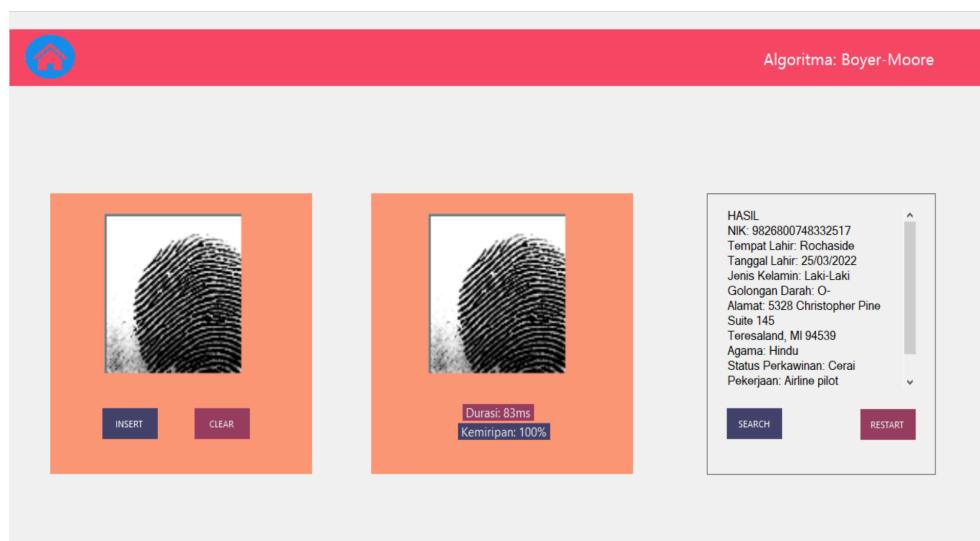
<b>Test Case 1</b> <b>8_M_Left_ring_finger</b>
---

KMP



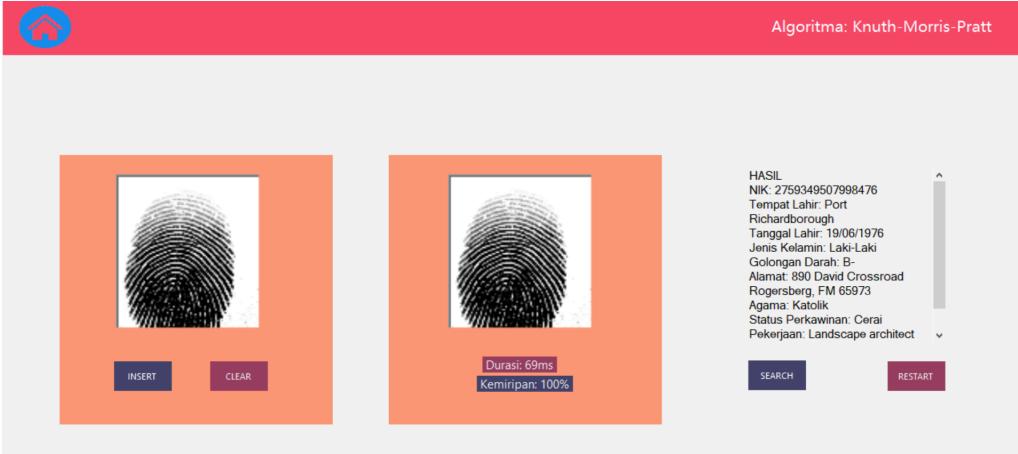
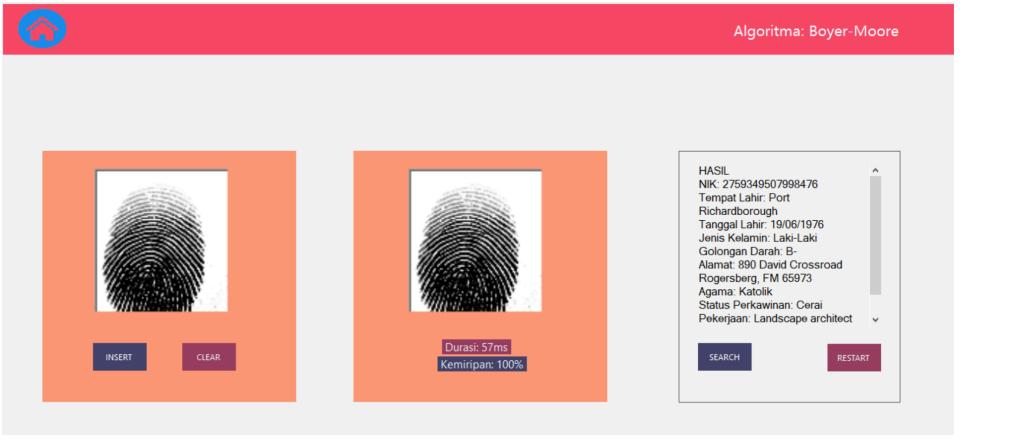
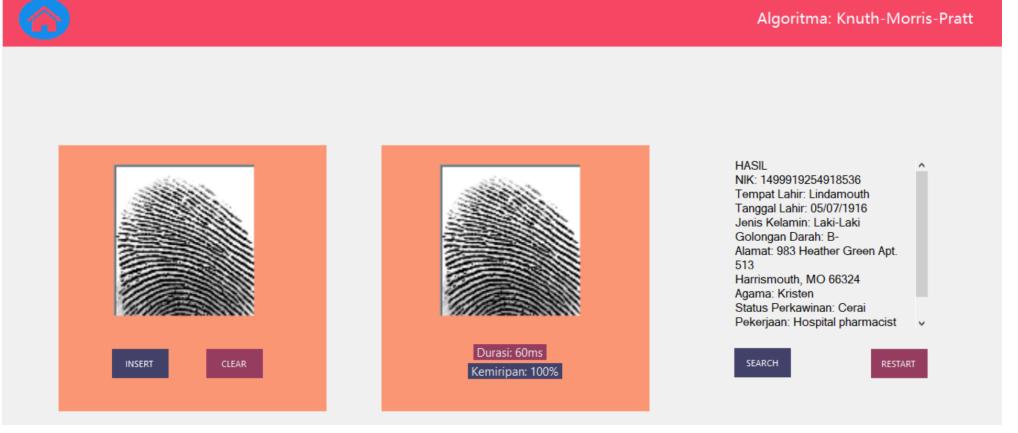
Durasi : 68 ms

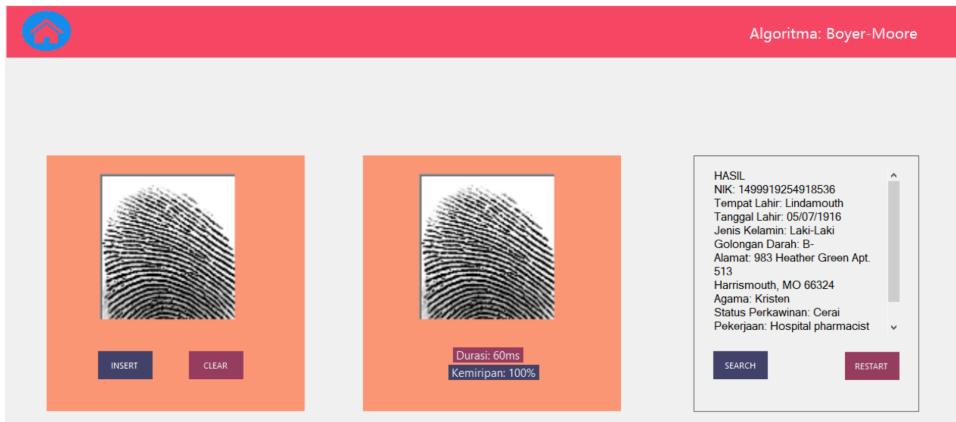
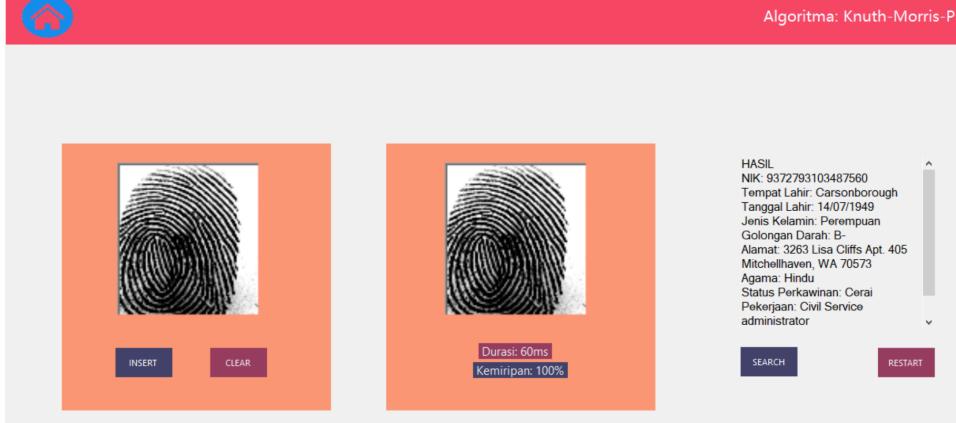
BM

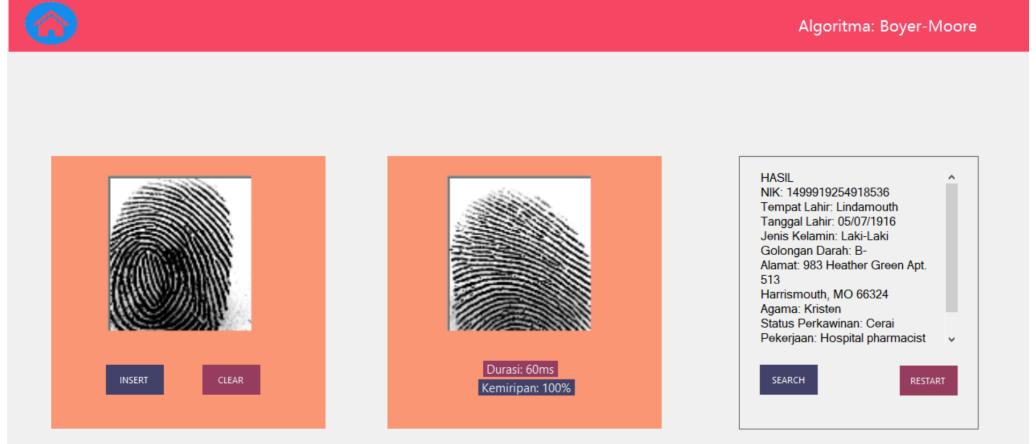
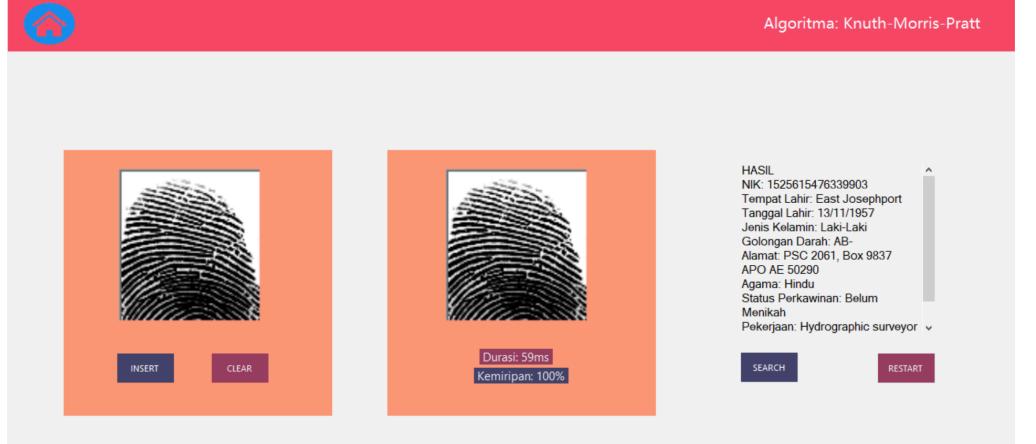
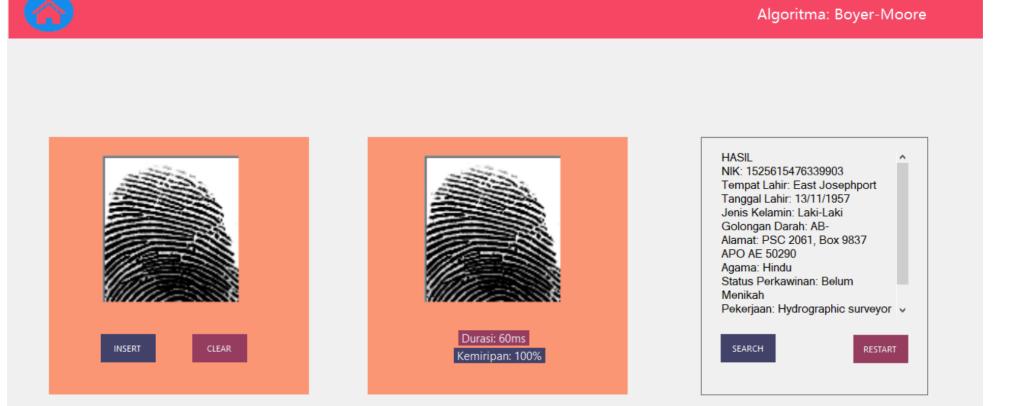


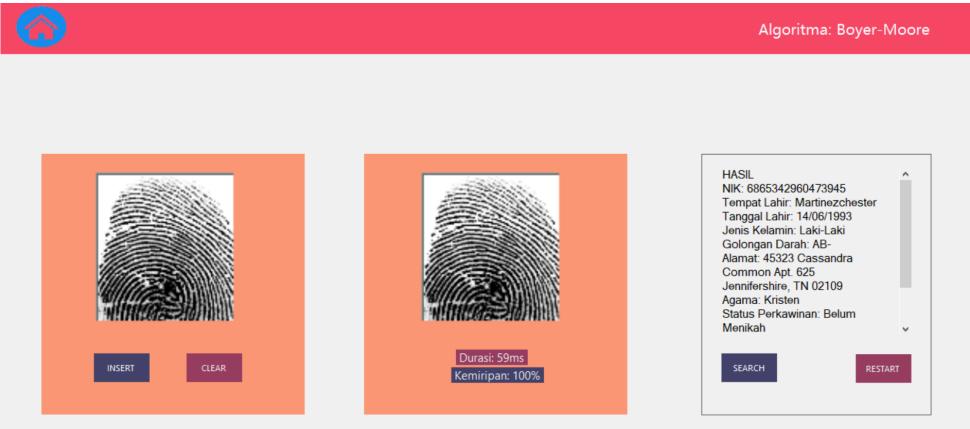
Durasi :83ms

**Test Case 2**  
**1\_\_M\_Left\_little\_finger**

KMP	 <p>Durasi: 69 ms</p>
BM	 <p>Durasi: 57ms</p>
<b>Test Case 3</b> <b>12_M_Right_thumb_finger</b>	
KMP	

	Durasi : 60 ms
BM	 <p>Algoritma: Boyer-Moore</p>
	Durasi : 60 ms
<b>Test Case 4</b> <b>599_M_Right_index_finger</b>	
KMP	 <p>Algoritma: Knuth-Morris-Pratt</p>
	Durasi : 60 ms

BM	 <p>Algoritma: Boyer-Moore</p> <p>Durasi : 60 ms</p>
<b>Test Case 5</b> <b>130_F_Right_thumb_finger</b>	
KMP	 <p>Algoritma: Knuth-Morris-Pratt</p> <p>Durasi : 59 ms</p>
Durasi : 59 ms	
BM	 <p>Algoritma: Boyer-Moore</p> <p>Durasi : 60 ms</p>

	Durasi : 60 ms
<b>Test Case 6</b> <b>453_F_Right_thumb_finger</b>	
KMP	 <p>Algoritma: Knuth-Morris-Pratt</p> <p>HASIL  NIK: 6865342960473945  Tempat Lahir: Martinezchester  Tanggal Lahir: 14/06/1993  Jenis Kelamin: Laki-Laki  Golongan Darah: AB-  Alamat: 45323 Cassandra Common Apt. 625  Jennifershire, TN 02109  Agama: Kristen  Status Perkawinan: Belum Menikah</p> <p>DURASI: 67ms KEMIRIPAN: 100%</p> <p>SEARCH RESTART</p>
Durasi : 67 ms	
BM	 <p>Algoritma: Boyer-Moore</p> <p>HASIL  NIK: 6865342960473945  Tempat Lahir: Martinezchester  Tanggal Lahir: 14/06/1993  Jenis Kelamin: Laki-Laki  Golongan Darah: AB-  Alamat: 45323 Cassandra Common Apt. 625  Jennifershire, TN 02109  Agama: Kristen  Status Perkawinan: Belum Menikah</p> <p>DURASI: 59ms KEMIRIPAN: 100%</p> <p>SEARCH RESTART</p>
Durasi : 59 ms	

#### 4.4 Analisis Hasil Pengujian

TC	BM	KMP
1		✓
2	✓	

3	=	=
4	=	=
5		✓
6	✓	

Berdasarkan hasil pengujian di atas dapat dilihat bahwa pada kasus rata-rata algoritma KMP dan BM cenderung memiliki waktu eksekusi yang sama. Namun pada gambar yang kontrasnya lebih tinggi, algoritma Boyer-Moore akan lebih cepat. Gambar yang memiliki kontras yang tinggi saat dikonversi menjadi ascii akan menghasilkan alfabet string ascii yang lebih luas juga. Algoritma BM lebih cepat ketika jumlah karakter yang berbeda dalam alfabet yang digunakan lebih banyak, inilah yang menyebabkan algoritma BM lebih unggul daripada algoritma KMP pada gambar yang memiliki kontras yang tinggi.

Sementara itu, pada gambar yang memiliki kontras yang rendah ditemukan bahwa algoritma KMP lebih unggul daripada algoritma BM. Gambar yang memiliki kontras yang rendah akan menghasilkan string ascii yang cenderung monoton dan/atau berulang. Algoritma KMP lebih cepat dan efisien dalam menangani teks yang memiliki pola berulang karena kemampuannya untuk menghindari perbandingan ulang dan tidak melakukan backtracking pada teks, inilah yang menyebabkan algoritma KMP lebih unggul daripada algoritma BM pada gambar yang memiliki kontras yang rendah.

## **BAB V**

### **Kesimpulan dan Saran**

#### **Kesimpulan**

Dari hasil test case dapat disimpulkan bahwa algoritma KMP dan BM memiliki kelebihan dan kekurangan masing-masing. Algoritma KMP lebih efisien dalam mencocokkan gambar sidik jari dengan kontras yang rendah. Sementara itu, Algoritma BM lebih efisien dalam mencocokkan gambar sidik jari dengan kontras yang tinggi. Pada akhirnya pemilihan algoritma ini bisa disesuaikan dengan kebutuhan pada setiap kasus.

#### **Saran**

Pemilihan pola (pattern) dalam pengambilan sampel dari gambar dapat ditingkatkan untuk menghasilkan hasil yang lebih optimal. Dengan memilih pola yang lebih efisien, waktu pencarian dapat dikurang.

Selain itu, penggunaan Docker untuk database dapat meningkatkan portabilitas dan konsistensi data. Docker memungkinkan pengembang untuk menjalankan database dalam wadah (container) yang terisolasi, memastikan bahwa lingkungan database tetap konsisten di berbagai tahap pengembangan, pengujian, dan produksi.

## **LAMPIRAN**

Link Repository Github: [https://github.com/AlbertGhazaly/Tubes3\\_Barbarians](https://github.com/AlbertGhazaly/Tubes3_Barbarians)

## **DAFTAR PUSTAKA**

- Munir, R. (2021). Pencocokan String. Diakses pada 9 Juni 2024, dari  
<https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Pencocokan-string-2021.pdf>
- Munir, R. (2019). String Matching dengan Regex. Diakses pada 9 Juni 2024, dari  
<https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2022-2023/String-Matching-dengan-Regex-2019.pdf>
- Munir, R. (2019). Modul Praktikum NLP: Regex. Diakses pada 9 Juni 2024, dari  
<https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2019-2020/Modul-Praktikum-NLP-Regex.pdf>