

LAPORAN TUGAS BESAR III
IF2211 STRATEGI ALGORITMA
PEMANFAATAN PATTERN MATCHING DALAM MEMBANGUN SISTEM DETEKSI
INDIVIDU BERBASIS BIOMETRIK MELALUI CITRA SIDIK JARI

Disusun untuk memenuhi tugas mata kuliah Strategi Algoritma
pada Semester II Tahun Akademik 2023/2024.



Oleh Kelompok 25 Mampusnya Diundur:
Thea Josephine Halim 13522012
Kayla Namira Mariadi 13522050
Diana Tri Handayani 13522104

PROGRAM STUDI TEKNIK INFORMATIKA
SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG
BANDUNG
2024

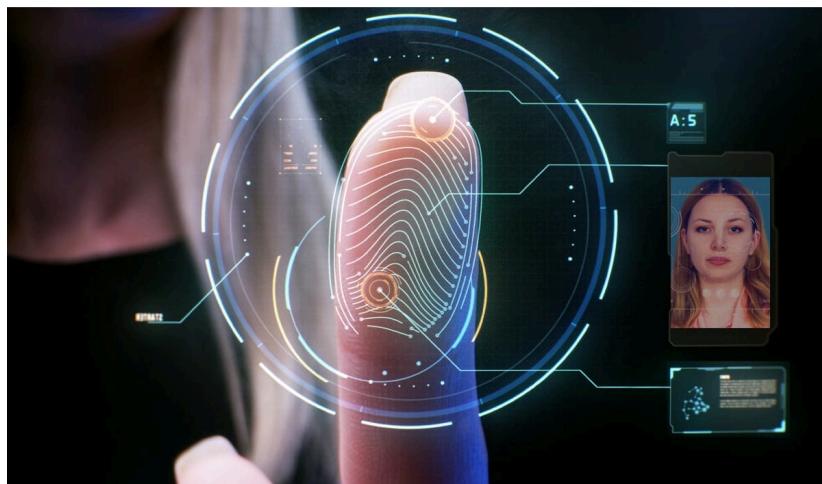
DAFTAR ISI

DAFTAR ISI	1
BAB I	
DESKRIPSI MASALAH	3
BAB II	
LANDASAN TEORI	4
2.1. Dasar Teori	4
2.1.1. Fingerprint Pattern Matching	4
2.1.2. Boyer Moore	5
2.1.3. Knuth Morris Pratt (KMP)	5
2.1.4. Bahasa alay	6
2.1.5. Persentase Kemiripan	6
2.2. Pembangunan Aplikasi Desktop	8
2.2.1. Integrated Development Environment (IDE)	8
2.2.2. Framework dan NuGet Packages	8
BAB III	
ANALISIS PEMECAHAN MASALAH	10
3.1. Langkah Pemecahan Masalah	10
3.1.1. Dummy Database (jika database masih kosong)	10
3.1.2. Preprocessing Images	10
3.1.3. Pencarian Solusi Algoritma	11
3.2. Proses penyelesaian solusi dengan Algoritma	12
3.2.1. Algoritma Knuth-Morris-Pratt (KMP)	12
3.2.2. Algoritma Boyer-Moore (BM)	13
3.3. Fitur Fungsional dan Arsitektur Aplikasi Desktop	13
3.3.1. Fitur Fungsional	14
3.3.2. Arsitektur Aplikasi Desktop	15
3.4 Ilustrasi Kasus	16
3.5 Batas Persentase Kemiripan	16
BAB IV	
IMPLEMENTASI DAN PENGUJIAN	17
4.1. Struktur Data	17
4.2. Implementasi Program	18
4.2.1. Algorithm	18
4.2.1.1. KMP.cs	18
4.2.2.1. Database.cs	27
4.4. Struktur Data dan Implementasi	41

4.5. Tata Cara Penggunaan Program	42
4.6. Tampilan Antarmuka	42
4.7. Hasil Pengujian	43
4.7.1. Pengujian Boyer Moore	44
4.7.2. Pengujian KMP	46
4.7.3. Pengujian Levenshtein	49
4.8. Analisis Hasil Pengujian	50
 BAB V	54
KESIMPULAN DAN SARAN	54
5.1. Kesimpulan	54
5.2. Saran	54
5.3. Refleksi	55
LAMPIRAN	56
Repository	56
Video	56
DAFTAR PUSTAKA	56

BAB I

DESKRIPSI MASALAH



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.

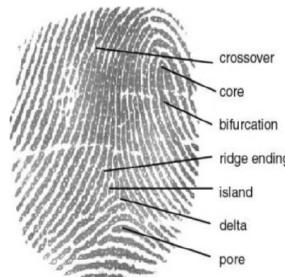
BAB II

LANDASAN TEORI

2.1. Dasar Teori

2.1.1. Fingerprint Pattern Matching

Fingerprint pattern matching adalah teknik biometrik yang digunakan untuk mengidentifikasi dan memverifikasi identitas individu berdasarkan pola unik pada sidik jari mereka. Proses ini melibatkan perbandingan antara sidik jari yang diambil pada saat ini dengan sidik jari yang sudah ada dalam database. Pola sidik jari terdiri dari berbagai titik karakteristik yang dikenal sebagai minutiae, yang meliputi bifurkasi (percabangan garis), ridge endings (ujung-ujung garis), dan lain-lain. Setiap sidik jari memiliki ribuan titik minutiae yang unik, sehingga membuat setiap sidik jari berbeda satu sama lain. Metode pencocokan ini sering digunakan dalam berbagai aplikasi keamanan seperti akses kontrol, identifikasi kriminal, dan perangkat autentikasi pribadi.



Gambar 2.1.1.1 Fingerprint Minutiae

Sumber: <https://www.researchgate.net>

Teknik fingerprint pattern matching dapat dibagi menjadi dua kategori utama: minutiae-based matching dan pattern-based matching. Minutiae-based matching berfokus pada ekstraksi dan perbandingan titik-titik minutiae, yang memungkinkan pencocokan yang sangat akurat meskipun pola sidik jari sebagian terhalang atau rusak. Sementara itu, pattern-based matching menggunakan analisis keseluruhan pola atau tekstur sidik jari, termasuk ridges dan valleys, yang biasanya lebih cepat tetapi mungkin kurang akurat dibandingkan metode berbasis minutiae.

Pada proyek kali ini akan digunakan metode pencocokan fingerprint pattern matching. Sebelum melakukan pencocokan, akan dilakukan dulu preprocessing untuk mendapatkan area pencocokan yang lebih akurat dan fokus pada pola utama sidik jari.

Awalnya gambar berwarna diubah menjadi grayscale untuk menyederhanakan analisis, tetapi untuk proyek ini telah disediakan gambar fingerprint greyscale hasil scan. Selanjutnya, kontras citra ditingkatkan menggunakan histogram equalization, diikuti dengan Gaussian blur untuk mengurangi noise. Proses ini dilanjutkan dengan adaptive thresholding yang menghasilkan citra biner, di mana ridges dan valleys lebih jelas terlihat. Terakhir, kontur dalam citra biner dideteksi, dan kontur terbesar dipilih sebagai representasi utama dari sidik jari untuk analisis lebih lanjut. Setelah menentukan kontur yang menunjukkan lokasi utama tengah fingerprint, akan dilakukan cropping area yang tidak memenuhi.

Setelah preprocessing fingerprint menjadi citra biner, langkah selanjutnya adalah mengubah citra biner tersebut menjadi representasi ASCII 8-bit. Dalam konteks ini, citra biner yang terdiri dari piksel-piksel dengan nilai 0 dan 1 dikonversi menjadi karakter-karakter ASCII yang masing-masing diwakili oleh 8-bit (1 byte). Setiap piksel biner (0 atau 1) dikumpulkan dalam kelompok 8 untuk membentuk satu karakter ASCII, sehingga citra biner dapat disimpan dan ditransfer sebagai teks ASCII. Representasi ini memungkinkan data sidik jari lebih efisien dalam penyimpanan dan juga transfer.

2.1.2. Boyer Moore

Boyer-Moore adalah algoritma pencocokan pola yang berbasis pada ide untuk memanfaatkan informasi dari pola yang dicocokkan terakhir dalam pencarian saat memutuskan pergeseran yang tepat dari jendela pencarian. Algoritma ini bekerja dengan menggeser jendela pencarian sepanjang teks yang ingin dicocokkan, dan mencocokkan pola dari kanan ke kiri. Saat algoritma menemukan karakter yang tidak cocok, akan digunakan dua aturan heuristik - aturan karakter yang buruk dan aturan pergeseran baik - untuk menentukan pergeseran jendela pencarian. Jika karakter yang tidak cocok terdapat dalam pola, maka aturan karakter yang buruk akan menentukan pergeseran yang besar. Namun, jika karakter yang tidak cocok tidak terdapat dalam pola, maka aturan pergeseran baik akan menentukan pergeseran yang besar.

2.1.3. Knuth Morris Pratt (KMP)

Algoritma KMP adalah salah satu algoritma pencarian string, dikembangkan secara terpisah oleh Donald E. Knuth pada tahun 1967 dan James H. Morris bersama Vaughan R. Pratt pada tahun 1966. Algoritma ini dirancang untuk menemukan keberadaan pola dalam teks dengan kompleksitas waktu rata-rata $O(m+n)$, dengan m adalah panjang teks dan n

adalah panjang pattern/pola. KMP menghindari pemeriksaan ulang karakter yang sudah dibandingkan dengan menggunakan tabel *Longest Prefix Suffix* (LPS). Tabel ini mencatat panjang prefix proper terpanjang dari pola yang juga merupakan suffix, sehingga ketika terjadi ketidakcocokan, algoritma dapat langsung melompat ke posisi yang sesuai dalam pola tanpa memeriksa kembali karakter yang sama, membuat proses pencarian lebih efisien dibandingkan metode pencarian naif.

2.1.4. Bahasa alay

Bahasa alay merupakan fenomena linguistik yang melibatkan berbagai variasi tulisan seperti kombinasi huruf besar-kecil, penggunaan angka sebagai pengganti huruf, dan penyingkatan kata. Dalam proyek deteksi individu berbasis biometrik dengan sidik jari, penanganan data yang korup seperti bahasa alay menjadi tantangan yang perlu diatasi untuk memastikan akurasi pencocokan data dan identifikasi individu.

Regex (Regular Expression) adalah alat yang sangat efektif untuk mendeteksi dan mengkonversi pola karakter dalam teks. Untuk menangani bahasa alay, kita dapat menggunakan Regex untuk mengidentifikasi dan mengganti karakter-karakter alay dengan huruf alfabet yang sesuai. Langkah-langkahnya meliputi:

- Deteksi Kombinasi Huruf Besar-Kecil: Dengan Regex, dapat dideteksi kombinasi huruf besar-kecil dan mengubahnya menjadi huruf kecil atau besar.
- Penggunaan Angka: Dalam konteks bahasa alay pada program ini, angka '1' diganti dengan huruf 'i', '4' dengan 'a', '5' dengan 's', dan seterusnya. Hal ini dilakukan dengan mendefinisikan pola Regex yang mencari angka-angka tersebut dan menggantinya dengan huruf yang sesuai. Pola regex yang digunakan akan mereferensi pola yang dianalisis dari website alay generator [berikut](#).
- Penyingkatan Kata: Penyingkatan seringkali melibatkan penghapusan vokal atau huruf tertentu. Untuk penyingkatan sendiri tidak dideteksi dengan regex, namun nantinya hasil nama alay yang sudah dikonversi dari regex akan diukur kemiripannya dengan KMP, BM, atau levenshtein.

2.1.5. Persentase Kemiripan

Untuk mengkuantifikasi persentase kemiripan suatu string dengan string lainnya, kami menggunakan algoritma Levenshtein Distance / Edit Distance. Algoritma Levenshtein

Distance digunakan untuk mengukur perbedaan antara dua string dengan menghitung jumlah operasi minimum (insert, delete, replace) yang diperlukan untuk mengubah satu string menjadi string lainnya. Berikut adalah langkah implementasinya:

1. Inisialisasi dua array. previousRow merepresentasikan jarak edit dari string kosong ("") ke setiap prefix pada string kedua, sehingga awalnya diinisialisasi dengan nilai dari 0 hingga index previousRow. currentRow digunakan untuk menghitung jarak edit untuk karakter saat ini dalam string pertama.
2. Iterasi melalui setiap karakter dari string pertama (outer loop). Untuk setiap karakter dalam string pertama, lakukan iterasi melalui setiap karakter dari string kedua (inner loop). Hitung jarak edit menggunakan tiga operasi (tambah, hapus, ganti) berikut:
 - a. Insert: Nilai dari previousRow[j] + 1. Skor operasi insert maksudnya skor untuk mengedit semua string1(string1[0..lenstring1]) menjadi prefix sebelum huruf terakhir string2(string2[0..lenstring2-1]), lalu insert karakter terakhir string2 ke string1.
 - b. Delete: Nilai dari currentRow[j - 1] + 1. Skor operasi delete maksudnya skor untuk mengedit prefix sebelum huruf terakhir string1(string1[0..lenstring1-1]) menjadi semua string2(string2[0..lenstring2]), lalu hapus karakter terakhir string1.
 - c. Replace: Nilai dari previousRow[j - 1] + cost, di mana cost adalah 0 jika karakter sama dan 1 jika berbeda. Skor operasi replace maksudnya skor untuk mengedit prefix sebelum huruf terakhir string1(string1[0..lenstring1-1]) menjadi prefix sebelum huruf terakhir string2(string2[0..lenstring2-1]), lalu ubah karakter terakhir string1 menjadi karakter terakhir string2.
3. Setelah inner loop selesai, tukar previousRow dan currentRow. Dengan cara ini, previousRow selalu menyimpan hasil dari iterasi sebelumnya, dan currentRow direset untuk iterasi berikutnya.
4. Setelah iterasi selesai, nilai akhir dari previousRow[panjangstring2] (elemen terakhir dari previousRow) adalah jarak Levenshtein antara dua string.

Algoritma Levenshtein yang kami implementasikan merupakan improvisasi dari pendekatan biasa yang menggunakan matriks untuk menyimpan hasil perhitungan jarak edit antar karakter. Pendekatan matriks kami modifikasi menjadi lebih efisien dalam penggunaan

memori dengan hanya menggunakan dua array (baris) yang diperbarui secara bergantian. Jika dibandingkan, pendekatan matriks memiliki kompleksitas memori $O(m*n)$, dengan $m=\text{panjang string1}$ dan $n = \text{panjang string2}$. Sedangkan pendekatan dua array memiliki kompleksitas memori $O(n)$. Untuk kompleksitas waktu juga pendekatan matriks lebih lambat, karena baris akan diinisialisasi sebanyak panjang string1. Perbedaan pendekatan ini cukup signifikan ketika membandingkan string yang panjang, seperti ascii.

2.2. Pembangunan Aplikasi Desktop

Pada Tugas Besar III Strategi Algoritma ini, program diimplementasikan dalam bahasa C# dengan antarmuka pengguna grafis (GUI) menggunakan WinForms. Dalam proyek ini, kode backend ditulis dalam C# untuk memanfaatkan kekuatan pemrosesan dan integrasi .NET, sedangkan WinForms digunakan untuk membangun antarmuka pengguna (GUI) dalam editor Visual Studio. Dalam Visual Studio terdapat 2 file yang merupakan file parsial dari Form1.cs, file utama untuk handling GUI dan integrasi. Form1.cs [Design] akan berisi tampilan preview dari GUI, sedangkan Form1.cs akan berisi komponen-komponen yang telah di drag and drop pada Designer dalam bentuk kode. Di dalam Form1.cs kita juga dapat melihat kode yang berada di balik komponen, contohnya dalam komponen button.

2.2.1. Integrated Development Environment (IDE)

Dalam pembuatan tugas besar ini, IDE yang digunakan adalah Visual Studio. Pemilihan Visual Studio sebagai IDE didasarkan pada kemudahannya membangun aplikasi desktop antarmuka pengguna. Hal ini termasuk dari desain GUI yang mudah, debugging yang mudah, pengelolaan proyek yang efisien, dan integrasi yang mudah dengan platform .NET.

2.2.2. Framework dan NuGet Packages

Framework yang kami gunakan pada proyek kali ini adalah WinForm. WinForm memudahkan dalam desain tampilan aplikasi dengan fitur drag and drop nya. Untuk penambahan komponen hanya perlu melakukan drag and drop komponen dari toolbox ke *form designer*. Dilanjutkan dengan kustomisasi (*customize*) properti font, warna teks,

ukuran, *alignment* dan lainnya. Untuk menyambungkan algoritma dengan interface, kita hanya perlu menambahkan *handle events*.

Nuget Packages simpelnya adalah *package manager* untuk .NET. NuGet memungkinkan pengembang untuk berbagi, mengunduh, dan mengelola paket pustaka pihak ketiga atau pustaka internal yang digunakan dalam proyek .NET. NuGet menyediakan cara mudah untuk mengelola dependensi proyek, memperbarui pustaka, dan menghindari masalah konflik versi. Pada proyek ini kami menggunakan packages EmguCV dan Bogus. Bogus sangat berguna untuk membuat data mock secara realistik, seperti nama pengguna, alamat, nomor telepon, dan banyak lagi. Sedangkan EmguCV akan memungkinkan kami menggunakan fungsi OpenCV dalam proyek terutama dalam preprocessing image menjadi ASCII.

BAB III

ANALISIS PEMECAHAN MASALAH

3.1. Langkah Pemecahan Masalah

Pada tugas besar kali ini, kami mengawali penggerjaan dengan analisis permasalahan untuk memudahkan penggerjaan. Analisis permasalahan tersebut menghasilkan beberapa bagian yang dijabarkan sebagai berikut:

3.1.1. Dummy Database (jika database masih kosong)

Dalam membangun sistem deteksi individu berbasis biometrik sidik jari, diperlukan basis data untuk menyimpan data dari setiap pemilik sidik jari. Basis data yang kami pakai berasal dari proses dummy dengan jumlah data yang menyesuaikan banyak data gambar sidik jari pada dataset “[Sokoto Coventry Fingerprint Dataset \(SOCOFing\)](#)” . Hasil dummy basis data ini akan menjadi sumber informasi yang akan ditampilkan dalam aplikasi.

3.1.2. Preprocessing Images

Dalam pencocokan gambar sidik jari antara masukan dan basis data, dilakukan preprocessing gambar terlebih dahulu. Langkah preprocessing yang pertama memangkas gambar agar hanya berfokus pada data sidik jari. Dalam pemangkasannya dilakukan konversi gambar menjadi grayscale, menerapkan threshold biner, menemukan kontur, dan memotong gambar ke persegi pembatas dari kontur terbesar. Setelah gambar berhasil dipangkas, gambar akan dikonversi menjadi kode ASCII untuk setiap pixelnya.



Gambar 3.1.2.1 Representasi ASCII sebuah gambar fingerprint

Konversi dilakukan dengan cara memetakan rentang intensitas grayscale ke rentang karakter ASCII. Dalam preprocessing ini, kami telah menyediakan pengubahan gambar yang telah dipotong menjadi kode ASCII secara penuh dan/atau hanya mengambil gambar dalam ukuran 30 pixel. Pengambilan data gambar dalam ukuran 30 pixel, dilakukan untuk menghasilkan pola yang lebih sederhana tetapi tetap akurat karena rata-rata dari ukuran gambar yang ada sekitar 90 pixel dan berdasarkan pengamatan, $\frac{1}{3}$ gambar terpenting cukup mewakili dari data gambar tersebut. Sehingga ketika menghadapi data yang besar, program dapat tetap berjalan dalam waktu yang singkat. Keluaran dari preprocessing gambar sidik jari adalah kode ASCII yang akan dicocokkan pada basis data.

3.1.3. Pencarian Solusi Algoritma

Ketika data gambar telah diubah menjadi kode ASCII, diperlukan langkah pencocokan string untuk menemukan gambar yang benar-benar sama (*exact match*). Terdapat dua algoritma yang umum digunakan dalam proses pencocokan string, yaitu algoritma Knuth-Morris-Pratt (KMP) dan algoritma Boyer-Moore (BM). Kedua algoritma tersebut memiliki metode yang berbeda, sehingga masing-masing algoritma

memiliki kelebihan dan kekurangan tersendiri. Setelah ditemukan string yang benar-benar cocok akan diambil basis data yang sesuai untuk menjadi keluaran aplikasi.

Namun, jika tidak ditemukan gambar yang benar-benar sama (*exact match*) dalam basis data, akan dilakukan pendekatan dengan mencari gambar dengan tingkat kemiripan yang paling tinggi. Kemiripan gambar dihitung menggunakan metode Levenshtein Distance yang mengukur jumlah operasi penyisipan, penghapusan, atau substitusi yang diperlukan untuk mengubah satu string menjadi string lain. Sehingga, metode ini memberikan cara yang sangat intuitif untuk memahami seberapa mirip suatu string dengan string berbeda lainnya.

3.2. Proses penyelesaian solusi dengan Algoritma

3.2.1. Algoritma Knuth-Morris-Pratt (KMP)

Algoritma ini menerima sebuah pattern (string) dan teks (string) yang akan dicocokkan keduanya. Umumnya string teks lebih panjang daripada string pattern. Sehingga pattern yang berbentuk string tersebut akan dicari prefix dan suffix terpanjang yang berpola sama pada setiap kemungkinan panjang substring. Hal tersebut biasa dikenal sebagai *Longest Prefix Suffix* (LPS). LPS ini dimanfaatkan agar pembandingan setiap huruf string dapat diminimasi ketika memiliki prefix dan suffix yang sama.

Dalam permasalahan ini, gambar sidik jari yang menjadi masukan untuk dicari biodatanya berperan sebagai string pattern. Sedangkan gambar yang tersimpan pada basis data berperan sebagai string teks. Hal tersebut terjadi karena gambar sidik jari basis data akan di preprocessing secara penuh, sedangkan gambar sidik jari dimasukan akan di preprocessing hanya dalam ukuran 30 pixel. Langkah tersebut menyebabkan panjang kode ASCII dari masukan cenderung lebih pendek daripada kode ASCII dari gambar basis data. Oleh karena itu, gambar sidik jari masukan berperan sebagai pattern.

Ketika di dalam fungsi pencarian LPS, data disimpan dalam bentuk array of integer dengan indeks array merujuk pada panjang substringnya. Sehingga LPS ketika substring sepanjang i disimpan pada array berindeks i . Array LPS ini akan menjadi rujukan indeks awal pencocokan pattern ketika ditemukan sebuah perbedaan dalam mencocokkan dengan string teks. Sehingga, pencocokan pattern tidak selalu harus dari indeks ke-0 ketika terdapat prefix dan suffix yang sama. LPS ini memiliki kelebihan

dalam mencocokkan string pattern dengan pola yang berulang dan variasi karakter string yang sederhana. Contohnya ketika string dalam bentuk *binary*.

3.2.2. Algoritma Boyer-Moore (BM)

Algoritma BM juga menerima masukan berupa sebuah pattern (string) dan teks (string) yang akan dicocokkan keduanya. Namun, dalam algoritma ini akan dicari posisi indeks terakhir pada pattern dari setiap kemungkinan huruf yang ada pada teks. Selain itu, ketika dalam algoritma KMP pattern dicek dari indeks ke-0, algoritma ini pattern dicek dari indeks ke-M hingga 0 dengan M adalah panjang string pattern. Posisi atau indeks karakter terakhir dapat menentukan seberapa jauh pergeseran untuk melakukan pengecekan selanjutnya.

Berdasarkan alasan yang sama dengan algoritma KMP, gambar sidik jari yang menjadi masukan untuk dicari biodatanya berperan sebagai string pattern dan gambar yang tersimpan pada basis data berperan sebagai string teks. Data posisi terakhir setiap karakter disimpan dalam suatu array of integer dengan indeks yang menunjukkan konversi suatu string menjadi integer berdasarkan ASCII. Sehingga array berukuran 256 yang menunjukkan banyak kemungkinan karakter ASCII. Ketika pattern tidak mengandung suatu karakter di ASCII, maka array akan diisi -1 karena untuk menandakan *out of index* sekaligus penyesuaian jumlah pergeseran ketika karakter tidak terdapat pada pattern.

Algoritma BM memiliki kelebihan ketika variasi karakter yang sangat beragam yang dapat menyebabkan pergeseran dapat bergerak jauh sehingga jumlah perbandingan karakter dapat minimal. Dalam permasalahan ini, berdasarkan hasil preprocessing image yang menghasilkan karakter ASCII yang sangat bervariasi, seharusnya algoritma ini dapat menyelesaikan pencocokan string ASCII dengan baik dan cepat.

3.3. Fitur Fungsional dan Arsitektur Aplikasi Desktop

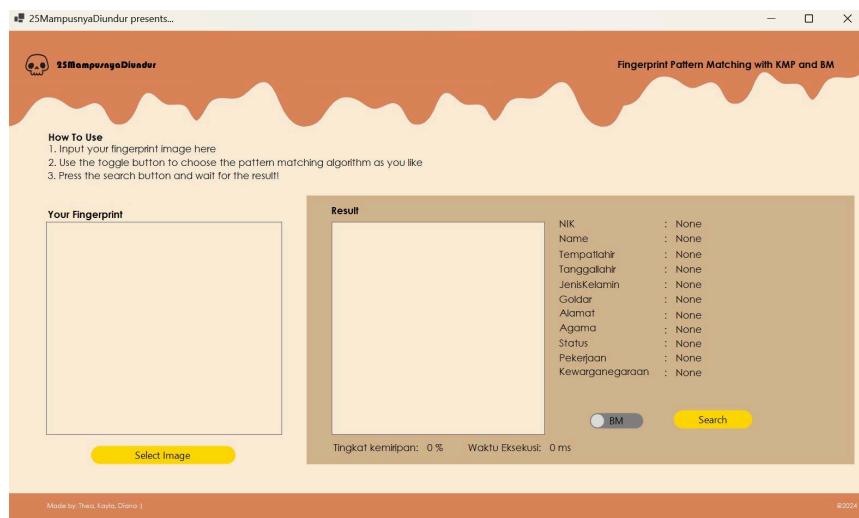
Pada tugas besar ini akan diwujudkan dalam bentuk aplikasi desktop sebagai tempat interaksi dengan pengguna. Pengguna akan memasukkan gambar sidik jari yang ingin dicari biodatanya. Pengguna dapat memilih di antara kedua algoritma dalam pemrosesannya, yaitu dengan tombol switch KMP atau BM. Aplikasi akan melakukan proses di balik layar sebelum menampilkan hasilnya ke layar dalam bentuk list biodata yang tersedia.

3.3.1. Fitur Fungsional

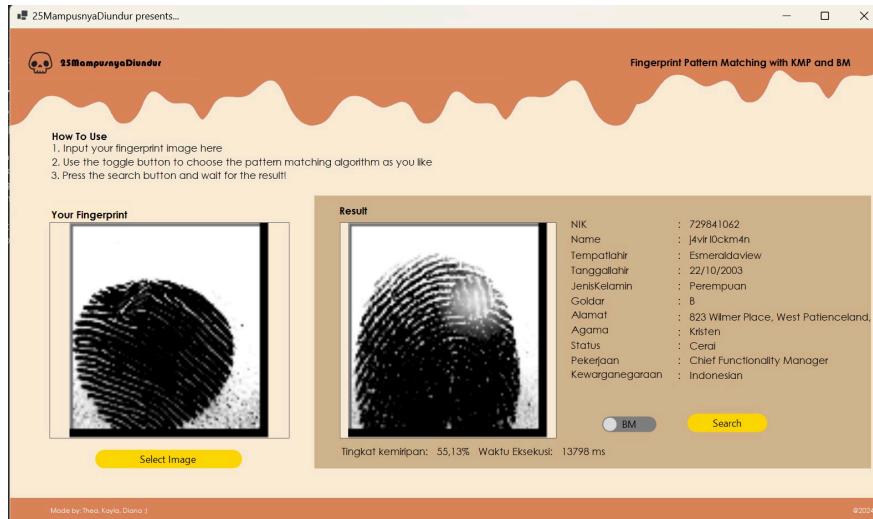
Fitur-fitur fungsional yang terdapat dalam website kami yaitu sebagai berikut

- a. *Button ‘Select Image’*, digunakan untuk memilih gambar yang ingin dimasukkan pengguna
- b. *Switch Button*, pengguna dapat memilih jenis algoritma untuk dijalankan antara KMP atau BM
- c. *Button ‘Search’*, digunakan untuk memulai proses pencarian

Setelah pencarian berhasil dilakukan, akan ditampilkan gambar sidik jari berdasarkan basis data beserta informasi biodata pemilik sidik jari. Selain itu juga disertakan tingkat kemiripan antara gambar masukan pengguna dengan gambar yang ditemukan di basis data beserta lama waktu eksekusi pencarian. Cara penggunaan secara singkat dalam bahasa Inggris juga ditampilkan di awal aplikasi. Berikut adalah tangkapan layar aplikasi sebelum dan sesudah digunakan:



Gambar 3.3.1 Antarmuka sebelum menerima masukan gambar



Gambar 3.3.2 Antarmuka setelah berhasil melakukan pencarian

3.3.2. Arsitektur Aplikasi Desktop

Aplikasi desktop yang dibuat memiliki satu halaman yang dijalankan melalui Forms1.cs. Halaman tersebut terdiri dari header yang mengandung judul atau nama aplikasi, bagian cara penggunaan, bagian memasukkan gambar sidik jari, bagian memilih algoritma, memulai pencarian beserta tempat keluaran muncul, dan footer yang berisi *credit*. Aplikasi dijalankan secara lokal atau dengan basis data di tempat yang sama. Hal ini sedikit menyebabkan aplikasi memiliki ukuran yang lumayan besar karena harus menyimpan basis data berupa gambar sekaligus.

Tampilan aplikasi desktop ini memanfaatkan framework UI WinForm karena cocok digunakan dalam pembangunan aplikasi dalam desktop Windows dan menggunakan bahasa C# sesuai ketentuan tugas besar. Dalam pembuatan algoritma KMP, BM, preprocessing, dan pengolahan basis data ditulis menggunakan bahasa C#. Untuk sistem manajemen basis data menggunakan MySQL karena mudah digunakan dan *open-source*. Dalam pengembangannya aplikasi ini menggunakan IDE (*Integrated Development Environment*) Visual Studio yang dapat mensupport dengan baik kebutuhan bahasa C# beserta framework WinForms.

3.4 Ilustrasi Kasus

Dalam penggunaan aplikasi, secara umum kasus yang terjadi adalah pengguna memasukkan sebuah gambar sidik jari, memilih algoritma, dan memulai pencarian. Dari pencarian ditemukan sidik jari yang benar-benar sesuai (*exact match*) di dalam basis data dan berhasil menampilkan biodata yang sesuai pada aplikasi. Dalam kasus ini artinya pencocokan sidik jari ditemukan ketika berada pada algoritma KMP atau BM.

Kasus lain yang dapat terjadi adalah ketika tidak ditemukan sidik jari yang benar-benar sama namun memiliki kemiripan dengan suatu sidik jari dalam basis data. Keluaran yang dihasilkan akan menunjukkan biodata dari sidik jari yang paling mirip beserta persentase kemiripannya. Dalam kasus ini, persentase kemiripan dihitung menggunakan metode Levenshtein Distance.

Kasus terakhir yang dapat terjadi adalah masukan gambar sidik jari tidak ada yang sama maupun mirip dengan gambar di basis data. Aplikasi akan mengeluarkan tingkat kemiripan 0 persen dengan biodata yang seluruhnya berisi ‘None’. Hal ini juga untuk mengatasi ketika gambar yang dimasukkan bukan gambar sidik jari.

3.5 Batas Persentase Kemiripan

Tingkat kemiripan hasil yang ditemukan dengan algoritma Boyer Moore dan Knuth Morris Pratt akan dianggap 100% (karena *exact match*), sedangkan tingkat kemiripan Levenshtein akan ditampilkan sesuai dengan kemiripan kedua gambar. Untuk *threshold* kemiripan Levenshtein akan dibatasi di atas angka 70. Kemiripan di bawah 70% akan dianggap sebagai gambar sidik jari (individu) yang berbeda. Penggunaan batas bawah 70 persen ini diambil dari hasil analisis kami dalam beberapa testcase. Kami menyimpulkan bahwa batas 70 persen ini paling optimal untuk menentukan apakah gambar sidik jari yang diinput sesuai dengan database atau tidak.

BAB IV

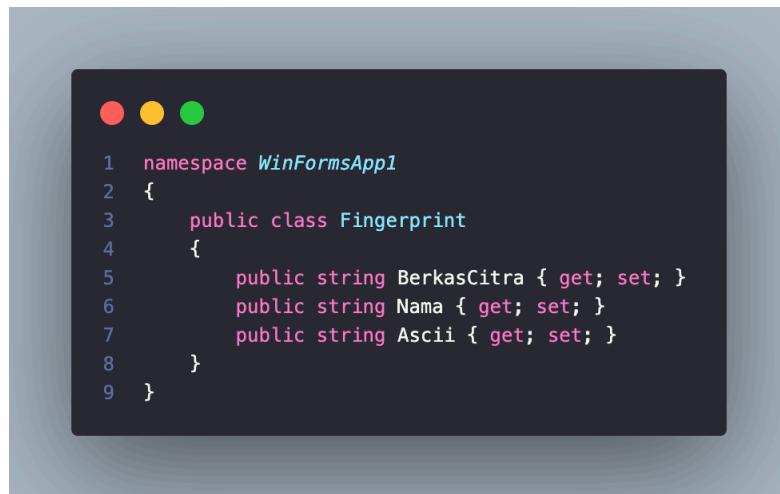
IMPLEMENTASI DAN PENGUJIAN

4.1. Struktur Data

Untuk menyimpan hasil dari query database, kami membuat 2 objek:

1. Fingerprint.cs

Pada Fingerprint, terdapat BerkasCitra yang akan menyimpan path menuju image, Nama berupa nama asli dari pemilik sidik jari, dan Ascii akan menyimpan hasil *full text* ascii yang dimasukkan saat *preprocessing*.



The screenshot shows a code editor window with a dark theme. At the top left, there are three colored circular icons: red, yellow, and green. The code editor displays the following C# code:

```
1  namespace WinFormsApp1
2  {
3      public class Fingerprint
4      {
5          public string BerkasCitra { get; set; }
6          public string Nama { get; set; }
7          public string Ascii { get; set; }
8      }
9 }
```

2. Biodata.cs

Pada Biodata, terdapat data terkait seseorang. Nama pada tabel ini bisa saja berupa nama alay.



```
1 using System;
2 namespace WinFormsApp1
3 {
4     public class Biodata
5     {
6         public string NIK { get; set; }
7         public string Nama { get; set; }
8         public string TempatLahir { get; set; }
9         public DateTime TanggalLahir { get; set; }
10        public string JenisKelamin { get; set; }
11        public string GolonganDarah { get; set; }
12        public string Alamat { get; set; }
13        public string Agama { get; set; }
14        public string StatusPerkawinan { get; set; }
15        public string Pekerjaan { get; set; }
16        public string Kewarganegaraan { get; set; }
17    }
18 }
```

4.2. Implementasi Program

Program dibagi menjadi 2 bagian utama, yaitu Algorithm dan GUI.

4.2.1. Algorithm

4.2.1.1. KMP.cs

Di dalam kelas Algorithm terdapat dua fungsi utama yaitu fungsi BuildLpsArray untuk mencari prefix suffix yang sama terpanjang dan fungsi KMPSearch untuk membandingkan dua string. Pada fungsi BuildLpsArray menerima masukan berupa sebuah string dan array of integer. Setiap kemungkinan panjang substring, dilakukan looping untuk dihitung panjang kesamaan antara prefix dan suffix (LPS), serta disimpan dalam array dari masukan. Jika tidak ada kesamaan, array akan diisi angka 0. Ketika panjang substring masih 0, array juga akan diisi 0.

Pada fungsi KMPSearch menerima masukan dua buah string yaitu sebagai pattern dan text. Fungsi diawali dengan inisiasi variabel untuk panjang pattern dan text, index pattern dan text yang akan diiterasi nantinya, serta array untuk menyimpan hasil pencarian panjang kesamaan prefix dan suffix. Selanjutnya dalam *looping ‘while’*, akan diiterasi untuk mengecek setiap karakter pada pattern dan text. Ketika ditemukan ketidakcocokan akan dicek array (LPS) untuk menentukan indeks pertama pada pattern yang akan dicek selanjutnya.

Ketika semua karakter sudah dicek dan cocok, akan mengembalikan sebuah boolean true. Namun, jika sampai seluruh text telah habis dicek dan tidak ditemukan kecocokan akan dikembalikan nilai false.

```
using System;
namespace WinFormsApp1.Algorithm {
    public static class KMPAlgorithm
    {
        // Fungsi untuk membangun tabel Longest Prefix Suffix (LPS)
        // yang digunakan oleh algoritma KMP
        private static void BuildLpsArray(string pattern, int[] lps)
        {
            int length = 0; // Panjang dari prefix yang cocok
            int i = 1;
            lps[0] = 0; // LPS dari elemen pertama selalu 0

            // Loop untuk menghitung lps[i] untuk i dari 1 ke M-1
            while (i < pattern.Length)
            {
                if (pattern[i] == pattern[length])
                {
                    length++;
                    lps[i] = length;
                    i++;
                }
                else
                {
                    if (length != 0)
                    {
                        length = lps[length - 1];
                    }
                    else
                    {
                        lps[i] = 0;
                        i++;
                    }
                }
            }
        }

        // Fungsi untuk mencari pola dalam teks menggunakan algoritma
        // KMP
    }
}
```

```

public static bool KMPSearch(string pattern, string text)
{
    int M = pattern.Length;
    int N = text.Length;
    // Buat array lps[] yang akan menampung panjang dari prefix
    // suffix terpanjang
    int[] lps = new int[M];
    int j = 0; // Indeks untuk pattern[]

    // Bangun tabel LPS
    BuildLpsArray(pattern, lps);

    int i = 0; // Indeks untuk text[]
    while (i < N)
    {
        if (pattern[j] == text[i])
        {
            j++;
            i++;
        }

        if (j == M)
        {
            System.Diagnostics.Debug.WriteLine("PATTERN"+
pattern);
            System.Diagnostics.Debug.WriteLine("TEXT"+ text);
            System.Diagnostics.Debug.WriteLine("Pattern
ditemukan pada indeks " + (i-j));
            return true;
        }
        else if (i < N && pattern[j] != text[i])
        {
            if (j != 0)
            {
                j = lps[j - 1];
            }
            else
            {
                i++;
            }
        }
    }
}

```

```
        System.Diagnostics.Debug.WriteLine("Pattern tidak ditemukan  
di KMP :(");  
        return false;  
    }  
  
}  
}
```

4.2.1.2. BM.cs

Di dalam kelas BoyerMooreAlgorithm terdapat dua fungsi utama yaitu fungsi BuildBadCharTable untuk mendata indeks terakhir setiap karakter pada pattern dan fungsi BMSearch untuk membandingkan dua string. Pada fungsi BuildBadCharTable menerima masukan berupa sebuah string dan *array of integer*. Diawali dengan iterasi pada array masukan untuk menginisiasi dengan masukan -1. Nilai -1 akan menandakan bahwa karakter dengan kode ASCII sama dengan indeks array tidak terdapat pada pattern. Selanjutnya diiterasi setiap karakter pada pattern untuk melakukan pendataan indeks terakhir dan disimpan pada array.

Pada fungsi BMSearch menerima masukan dua buah string yaitu sebagai pattern dan text. Fungsi diawali dengan inisiasi variabel untuk panjang pattern dan text, variabel untuk menghitung jumlah pergeseran yang harus dilakukan nantinya, serta array untuk menyimpan hasil pencarian data indeks terakhir setiap karakter menggunakan fungsi BuildBadCharTable. Selanjutnya dalam *looping ‘while’*, akan diiterasi untuk mengecek setiap karakter pada pattern dan text dari indeks terakhir pattern. Ketika ditemukan ketidakcocokan akan dicek array data indeks terakhir jumlah pergeseran indeks pengecekan selanjutnya. Ketika semua karakter sudah dicek dan cocok, akan mengembalikan sebuah boolean true. Namun, jika sampai seluruh text telah habis dicek dan tidak ditemukan kecocokan akan dikembalikan nilai false.

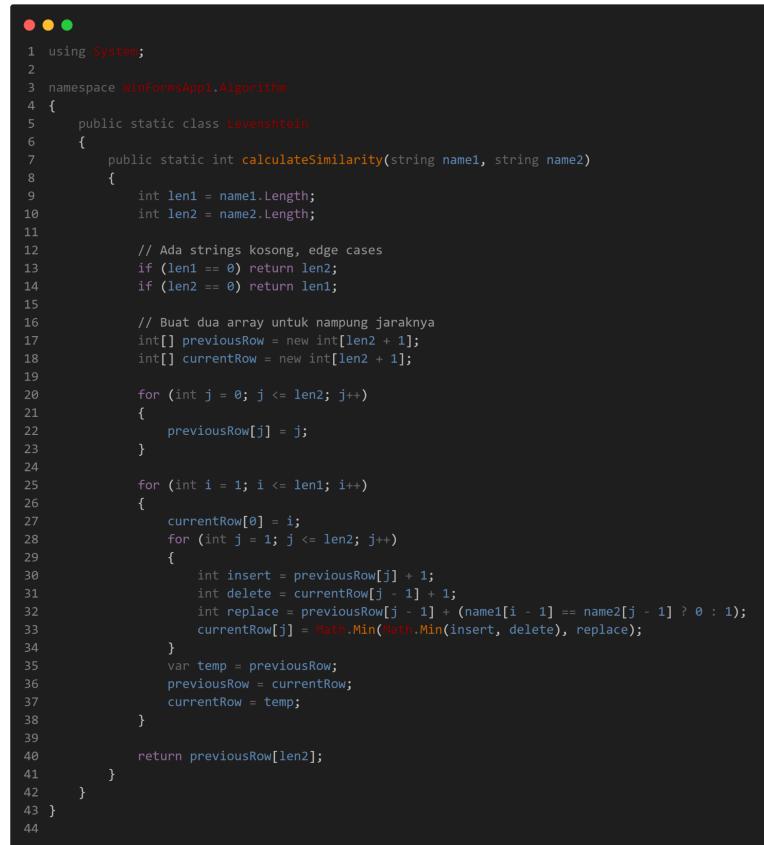


```
1 using System;
2 namespace WinFormsApp1.Algorithm{
3     public static class BoyerMooreAlgorithm
4     {
5         // Fungsi untuk membuat tabel Bad Character Heuristic
6         private static void BuildBadCharTable(string pattern, int[] badChar)
7         {
8             int m = pattern.Length;
9
10            // Inisialisasi semua entri tabel badChar dengan -1
11            for (int i = 0; i < 256; i++)
12                badChar[i] = -1;
13
14            // Isi nilai terakhir dari karakter yang muncul di pattern
15            for (int i = 0; i < m; i++)
16                badChar[(int)pattern[i]] = i;
17        }
18
19        // Fungsi untuk mencari pola dalam teks menggunakan algoritma Boyer-Moore
20        public static bool BMSearch(string pattern, string text)
21        {
22            int m = pattern.Length;
23            int n = text.Length;
24            int[] badChar = new int[256];
25            // Bangun tabel Bad Character Heuristic
26            BuildBadCharTable(pattern, badChar);
27            int s = 0; // s adalah pergeseran dari pattern ke teks
28            while (s <= (n - m))
29            {
30                int j = m - 1;
31
32                // Kurangi indeks j dari belakang pattern ke depan
33                while (j >= 0 && pattern[j] == text[s + j])
34                    j--;
35
36                // Jika pattern cocok dengan teks pada posisi s
37                if (j < 0)
38                {
39                    System.Diagnostics.Debug.WriteLine("PATTERN"+ pattern);
40                    System.Diagnostics.Debug.WriteLine("TEXT"+ text);
41                    System.Diagnostics.Debug.WriteLine("Pattern ditemukan pada indeks " + s);
42                    return true;
43                }
44                else
45                {
46                    // Pergeseran pattern berdasarkan tabel bad character
47                    s += Math.Max(1, j - badChar[text[s + j]]);
48                }
49            }
50            System.Diagnostics.Debug.WriteLine("Pattern tidak ditemukan di BM :(");
51            return false;
52        }
53    }
54 }
```

4.2.1.3. Levenshtein.cs

Fungsi ini bertujuan untuk menghitung kemiripan antara dua string. Pertama, fungsi menginisiasi dua array untuk menyimpan jarak antar karakter pada iterasi sebelumnya dan saat ini. Array previousRow diinisiasi dengan nilai berturut-turut dari 0 hingga panjang string kedua. Selanjutnya, untuk setiap karakter dalam string pertama, iterasi dilakukan untuk setiap karakter dalam string kedua. Pada setiap iterasi, dihitung biaya untuk operasi penyisipan,

penghapusan, dan penggantian. Biaya terendah dari ketiga operasi tersebut dipilih dan disimpan dalam currentRow. Setelah iterasi selesai untuk satu baris, currentRow menjadi previousRow untuk iterasi berikutnya. Nilai akhir dari elemen terakhir previousRow adalah jarak Levenshtein antara dua string.



```
1 using System;
2
3 namespace WinFormsApp1.Algorithm
4 {
5     public static class Levenshtein
6     {
7         public static int calculateSimilarity(string name1, string name2)
8         {
9             int len1 = name1.Length;
10            int len2 = name2.Length;
11
12            // Ada strings kosong, edge cases
13            if (len1 == 0) return len2;
14            if (len2 == 0) return len1;
15
16            // Buat dua array untuk nampung jaraknya
17            int[] previousRow = new int[len2 + 1];
18            int[] currentRow = new int[len2 + 1];
19
20            for (int j = 0; j <= len2; j++)
21            {
22                previousRow[j] = j;
23            }
24
25            for (int i = 1; i <= len1; i++)
26            {
27                currentRow[0] = i;
28                for (int j = 1; j <= len2; j++)
29                {
30                    int insert = previousRow[j] + 1;
31                    int delete = currentRow[j - 1] + 1;
32                    int replace = previousRow[j - 1] + (name1[i - 1] == name2[j - 1] ? 0 : 1);
33                    currentRow[j] = Math.Min(Math.Min(insert, delete), replace);
34                }
35                var temp = previousRow;
36                previousRow = currentRow;
37                currentRow = temp;
38            }
39
40            return previousRow[len2];
41        }
42    }
43 }
44 }
```

4.2.1.4. Dummy.cs

File ini bertujuan untuk menghasilkan data dummy untuk keperluan pengujian ketika basis data masih kosong. Kami menggunakan library Bogus sebagai faker untuk menghasilkan berbagai macam data palsu. Pertama, akan diambil semua file gambar dengan ekstensi .bmp dari folder test. Untuk setiap file gambar pada folder, dibuat data sidik jari dan biodata palsu. Data sidik jari dimasukkan ke dalam tabel sidik jari. Data biodata dibuat menggunakan Bogus dan dimasukkan ke dalam tabel biodata.



```
1  using Bogus;
2  using WinFormsApp1.Algorithm;
3  using System.IO;
4
5  namespace WinFormsApp1
6  {
7      public static class Dummy
8      {
9          // Kelas dummy buat generate dummy data kalau database masih kosong
10         // Pakai Package Bogus untuk dapatkan macam-macam fake data
11         public static void GenerateDummy(string projectDirectory, Fingerprints fingerprints)
12         {
13             // Kalo kosong pass
14             if (!fingerprints.IsFingerprintTableEmpty())
15             {
16                 Console.WriteLine("Table sidik_jari already has data. Skipping dummy data generation.");
17                 return;
18             }
19
20             string[] imageFiles = Directory.GetFiles(projectDirectory, "*.bmp");
21             Console.WriteLine($"[{projectDirectory}]");
22             var faker = new Faker();
23             foreach (string imagePath in imageFiles)
24             {
25                 string fileName = Path.Combine("test", Path.GetFileName(imagePath));
26                 Person person = new Faker().Person;
27                 string name = person.FullName;
28                 fingerprints.InsertFingerprint(name, fileName);
29                 Console.WriteLine($"{name}, {fileName}");
30
31                 string NIK = faker.Random.Number(100000000, 99999999).ToString();
32                 string Nama = AlayTranslator.ConvertToAlay(name);
33                 string TempatLahir = faker.Address.City();
34                 DateTime TanggalLahir = faker.Date.Past(30, DateTime.Now.AddYears(-20));
35                 string JenisKelamin = faker.PickRandom(new[] { "Laki-Laki", "Perempuan" });
36                 string GolonganDarah = faker.PickRandom(new[] { "A", "B", "AB", "O" });
37                 string Alamat = faker.Address.FullAddress();
38                 string Agama = faker.PickRandom(new[] { "Islam", "Kristen", "Katolik", "Hindu", "Buddha", "Konghucu" });
39                 string StatusPerkawinan = faker.PickRandom(new[] { "Belum Menikah", "Menikah", "Cerai" });
40                 string Pekerjaan = faker.Name.JobTitle();
41                 string Kewarganegaraan = "Indonesian";
42                 fingerprints.InsertBiodata(NIK, Nama, TempatLahir, TanggalLahir, JenisKelamin, GolonganDarah, Alamat, Agama, StatusPerkawinan, Pekerjaan, Kewarganegaraan);
43                 Console.WriteLine($"{NIK}, {Nama}, {TempatLahir}, {TanggalLahir}, {JenisKelamin}, {GolonganDarah}, {Alamat}, {Agama}, {StatusPerkawinan}, {Pekerjaan}, {Kewarganegaraan}");
44             }
45         }
46     }
47 }
48 }
```

4.2.1.5. AlayTranslator.cs

File ini terdiri dari 2 fungsi:

1. translateAlay.cs

Fungsi ini bertujuan untuk memurnikan nama alay pada tabel biodata menggunakan regex. Adapun jenis nama alay yang di-*handle* hanya berupa pengubahan angka menjadi huruf dan *case* dari nama. Untuk pencocokan dengan nama asli, dilakukan dengan algoritma KMP-BM, dan jika tidak cocok akan diukur dengan levenshtein distance nantinya. Langkah-langkahnya meliputi:

1. Penggunaan Angka: Pertama, nama alay yang mengandung angka akan diubah menjadi huruf yang bersesuaian ($1 \rightarrow i$, $4 \rightarrow a$, $6 \rightarrow g$, $3 \rightarrow e$, $0 \rightarrow o$, $5 \rightarrow s$). Hal ini dilakukan dengan mendefinisikan pola Regex yang mencari angka-angka tersebut dan menggantinya dengan huruf yang sesuai.
2. Deteksi Kombinasi Huruf Besar-Kecil: Kedua, regex mendeteksi kombinasi huruf besar-kecil dan mengubah semua huruf menjadi huruf kecil terlebih dahulu. Kemudian, setiap awal kata akan dikapitalisasi (karena merupakan nama, sehingga diasumsikan pada tabel sidiq_jari, nama ditulis dengan setiap awal kata dikapitalisasi).
3. Penyingkatan Kata: Penyingkatan seringkali melibatkan penghapusan vokal atau huruf tertentu. Untuk mencocokkan hasil pengubahan nama alay dengan regex dengan nama asli, diaplikasikan algoritma KMP-BM, dan jika tidak cocok akan diukur dengan levenshtein distance.

```
● ● ●
1 using System;
2 using System.Text.RegularExpressions;
3 namespace WinFormsApp1.Algorithm
4 {
5     public class AlayTranslator
6     {
7         // pakai regex untuk translate nama alay sesuai aturan tertentu
8         // cases ada pada penggunaan angka untuk huruf vokal, konversi lower dan uppercase
9         public static string translateAlay(string input)
10        {
11            string numPattern = "[14630]";
12            string result = Regex.Replace(input, numPattern, m => {
13                switch (m.Value)
14                {
15                    case "1": return "i";
16                    case "4": return "a";
17                    case "6": return "g";
18                    case "3": return "e";
19                    case "5": return "s";
20                    case "0": return "o";
21                    default: return m.Value;
22                }
23            });
24            result = result.ToLower();
25            result = Regex.Replace(result, @"\b[a-z]", m => m.Value.ToUpper());
26
27            return result;
28        }
29    }
30 }
```

2. ConvertToAlay.cs

Fungsi ini bertujuan untuk mengubah suatu nama asli menjadi nama alay dengan mengubah beberapa huruf menjadi angka, dan menyingkat nama. Fungsi ini digunakan untuk keperluan seeding database.

```
31     // ConvertToAlay untuk dummy data name pada tabel biodata
32     public static string ConvertToAlay(string input)
33     {
34         string result = Regex.Replace(input, "[AIEGoag]", m =>
35         {
36             switch (m.Value)
37             {
38                 case "A": return "4";
39                 case "a": return "4";
40                 case "I": return "1";
41                 case "E": return "3";
42                 case "e": return "E";
43                 case "o": return "0";
44                 case "g": return "9";
45                 default: return m.Value;
46             }
47         });
48         result = result.ToLower();
49
50         string temp = "";
51         bool firstVowel = false;
52         foreach (char c in result)
53         {
54             if ("aeiou".IndexOf(c) >= 0)
55             {
56                 if (!firstVowel)
57                 {
58                     temp += c;
59                     firstVowel = true;
60                 }
61             }
62             else
63             {
64                 temp += c;
65             }
66         }
67         return temp;
68     }
69 }
70 }
```

4.2.2. Database

4.2.2.1. Database.cs

File ini berisi fungsi-fungsi yang mengenkapsulasi proses koneksi menuju database.



```
1  using MySql.Data.MySqlClient;
2
3  namespace WinFormsApp1
4  {
5      public class Database
6      {
7          private readonly MySqlConnection connection;
8
9          public Database(string connectionString)
10         {
11             connection = new MySqlConnection(connectionString);
12         }
13
14         public void OpenConnection()
15         {
16             if (connection.State == System.Data.ConnectionState.Closed)
17             {
18                 connection.Open();
19             }
20         }
21
22         public void CloseConnection()
23         {
24             if (connection.State == System.Data.ConnectionState.Open)
25             {
26                 connection.Close();
27             }
28         }
29
30         public MySqlConnection GetConnection()
31         {
32             return connection;
33         }
34     }
35 }
36 }
37 }
38 }
```

4.2.2.2 FingerprintDB.cs

File ini berisi fungsi-fungsi untuk melakukan query ke database, seperti mendapatkan data Fingerprint dan Biodata, melakukan alter table untuk menambahkan atribut ascii di preprocess, dan insert ke database.

1. InsertFingerprint

Memasukkan data Fingerprint ke tabel sidik_jari



```
1 using MySql.Data.MySqlClient;
2
3 namespace WinFormsApp1
4 {
5     public class Fingerprints
6     {
7         private readonly Database db;
8
9         public Fingerprints(Database database)
10        {
11            db = database;
12        }
13
14         public void InsertFingerprint(string nama, string berkas_citra)
15        {
16             db.OpenConnection();
17             MySqlCommand cmd = new MySqlCommand("INSERT INTO sidik_jari (berkas_citra, nama) VALUES (@berkas_citra, @nama)", db.GetConnection());
18             cmd.Parameters.AddWithValue("@berkas_citra", berkas_citra);
19             cmd.Parameters.AddWithValue("@nama", nama);
20             cmd.ExecuteNonQuery();
21             db.CloseConnection();
22        }
23    }
```

2. InsertBiodata

Memasukkan data Biodata ke tabel biodata



```
1     public Fingerprint GetFingerprintByName(string nama)
2     {
3         db.OpenConnection();
4         MySqlCommand cmd = new MySqlCommand("SELECT * FROM sidik_jari WHERE nama = @nama", db.GetConnection());
5         cmd.Parameters.AddWithValue("@nama", nama);
6         MySqlDataReader reader = cmd.ExecuteReader();
7
8         Fingerprint fingerprint = null;
9         if (reader.Read())
10        {
11             fingerprint = new Fingerprint{
12                 BerkasCitra = reader.GetString("berkas_citra"),
13                 Nama = reader.GetString("nama"),
14                 Ascii = reader.IsDBNull(reader.GetOrdinal("ascii")) ? null : reader.GetString("ascii")
15             };
16        }
17
18         db.CloseConnection();
19         return fingerprint;
20     }
```

3. GetFingerprintByName

Metode ini mengambil data sidik jari berdasarkan nama tertentu.

```
● ● ●
1 public Fingerprint GetFingerprintByName(string nama)
2 {
3     db.OpenConnection();
4     MySqlCommand cmd = new MySqlCommand("SELECT * FROM sidik_jari WHERE nama = @nama", db.GetConnection());
5     cmd.Parameters.AddWithValue("@nama", nama);
6     MySqlDataReader reader = cmd.ExecuteReader();
7
8     Fingerprint fingerprint = null;
9     if (reader.Read())
10    {
11        fingerprint = new Fingerprint{
12            BerkasCitra = reader.GetString("berkas_citra"),
13            Nama = reader.GetString("nama"),
14            Ascii = reader.IsDBNull(reader.GetOrdinal("ascii")) ? null : reader.GetString("ascii")
15        };
16    }
17
18    db.CloseConnection();
19    return fingerprint;
20 }
```

4. GetAllFingerprintDataSeparated

Metode ini mengambil semua nama, berkas_citra, dan fulltext ascii dari tabel sidik_jari dan return ketiga value tersebut dalam bentuk sebuah tuple yang terdiri dari list nama, berkas_citra, dan ascii. Pembuatan metode ini bertujuan untuk mengurangi jumlah looping yang dibutuhkan untuk mengekstrak data nama, path image, dan ascii representasi untuk setiap data di database.

```
● ● ●
1 public (List<string>, List<string>, List<string>) GetAllFingerprintDataSeparated()
2 {
3     List<string> fingerprintList_berkas = new();
4     List<string> fingerprintList_nama = new();
5     List<string> fingerprintList_ascii = new();
6
7     db.OpenConnection();
8     MySqlCommand cmd = new MySqlCommand("SELECT * FROM sidik_jari", db.GetConnection());
9     MySqlDataReader reader = cmd.ExecuteReader();
10
11    while (reader.Read())
12    {
13        Fingerprint fingerprint = new Fingerprint{
14            BerkasCitra = reader.GetString("berkas_citra"),
15            Nama = reader.GetString("nama"),
16            Ascii = reader.IsDBNull(reader.GetOrdinal("ascii")) ? null : reader.GetString("ascii")
17        };
18        fingerprintList_berkas.Add(fingerprint.BerkasCitra);
19        fingerprintList_nama.Add(fingerprint.Nama);
20        fingerprintList_ascii.Add(fingerprint.Ascii);
21    }
22
23    db.CloseConnection();
24    return (fingerprintList_berkas, fingerprintList_nama, fingerprintList_ascii);
25 }
```

5. GetAllFingerprintData

Metode ini mengambil semua data sidik jari dari tabel sidik_jari.

```
public List<Fingerprint> GetAllFingerprintData()
```

```

    {
        List<Fingerprint> fingerprintList = new();

        db.OpenConnection();
        MySqlCommand cmd = new MySqlCommand("SELECT * FROM sidik_jari", db.GetConnection());
        MySqlDataReader reader = cmd.ExecuteReader();

        while (reader.Read())
        {
            Fingerprint fingerprint = new Fingerprint{
                BerkasCitra = reader.GetString("berkas_citra"),
                Nama = reader.GetString("nama"),
                Ascii =
reader.IsDBNull(reader.GetOrdinal("ascii")) ? null :
reader.GetString("ascii")
            };
            fingerprintList.Add(fingerprint);
        }

        db.CloseConnection();
        return fingerprintList;
    }
}

```

6. GetAllBiodataData

Metode ini mengambil semua berkas_citra dari tabel sidik_jari.

```

public List<Biodata> GetAllBiodataData()
{
    List<Biodata> biodataList = new();

    db.OpenConnection();
    MySqlCommand cmd = new MySqlCommand("SELECT * FROM biodata", db.GetConnection());
    MySqlDataReader reader = cmd.ExecuteReader();

    while (reader.Read())
    {
        Biodata biodata = new Biodata
        {
            NIK = reader.GetString("NIK"),
            Nama = reader.GetString("nama"),
            ...
        };
        biodataList.Add(biodata);
    }
}

```

```

        TempatLahir = reader.GetString("tempat_lahir"),
        TanggalLahir =
reader.GetDateTime("tanggal_lahir"),
        JenisKelamin =
reader.GetString("jenis_kelamin"),
        GolonganDarah =
reader.GetString("golongan_darah"),
        Alamat = reader.GetString("alamat"),
        Agama = reader.GetString("agama"),
        StatusPerkawinan =
reader.GetString("status_perkawinan"),
        Pekerjaan = reader.GetString("pekerjaan"),
        Kewarganegaraan =
reader.GetString("kewarganegaraan")
    };
    biodataList.Add(biodata);
}

db.CloseConnection();
return biodataList;
}

```

7. alterTable

Metode ini menambahkan kolom ascii ke dalam tabel sidik_jari. Pertama-tama akan dilakukan pengecekan untuk memastikan atribut ascii belum ada pada tabel, sebab bisa menyebabkan error. Apabila atribut ascii sudah ada, kita lewati tahap penambahan atribut ascii. Atribut ascii ini akan diganti tipe datanya menjadi *longtext* agar bisa memuat seluruh teks ascii.

```

1 public void alterTable()
2 {
3     db.OpenConnection();
4
5     MySqlCommand checkColumnCmd = new MySqlCommand(
6         "SELECT COUNT(*) FROM information_schema.COLUMNS WHERE TABLE_NAME = 'sidik_jari' AND COLUMN_NAME = 'ascii'",
7         db.GetConnection()
8     );
9     int columnExists = Convert.ToInt32(checkColumnCmd.ExecuteScalar());
10
11    if (columnExists > 0)
12    {
13        // Check the data type of the 'ascii' column
14        MySqlCommand checkColumnTypeCmd = new MySqlCommand(
15            "SELECT DATA_TYPE FROM information_schema.COLUMNS WHERE TABLE_NAME = 'sidik_jari' AND COLUMN_NAME = 'ascii'",
16            db.GetConnection()
17        );
18        string dataType = checkColumnTypeCmd.ExecuteScalar().ToString();
19
20        // If the data type is not LONGTEXT, alter the column to LONGTEXT
21        if (dataType.ToUpper() != "LONGTEXT")
22        {
23            MySqlCommand alterColumnCmd = new MySqlCommand(
24                "ALTER TABLE sidik_jari MODIFY COLUMN ascii LONGTEXT",
25                db.GetConnection()
26            );
27            alterColumnCmd.ExecuteNonQuery();
28            System.Diagnostics.Debug.WriteLine("Column 'ascii' modified to LONGTEXT.");
29        }
30        else
31        {
32            System.Diagnostics.Debug.WriteLine("Column 'ascii' is already LONGTEXT.");
33        }
34    }
35    else
36    {
37        // Add the 'ascii' column as LONGTEXT if it doesn't exist
38        MySqlCommand addColumnCmd = new MySqlCommand(
39            "ALTER TABLE sidik_jari ADD COLUMN ascii LONGTEXT",
40            db.GetConnection()
41        );
42        addColumnCmd.ExecuteNonQuery();
43        System.Diagnostics.Debug.WriteLine("Column 'ascii' added as LONGTEXT.");
44    }
45
46    db.CloseConnection();
47 }

```

8. insertAscii

Metode ini memperbarui kolom ascii pada tabel sidik_jari berdasarkan nama.

```

public void insertAscii(string nama, string ascii){
    db.OpenConnection();

    MySqlCommand cmd = new MySqlCommand("UPDATE sidik_jari
SET ascii = @ascii WHERE nama = @nama", db.GetConnection());
    cmd.Parameters.AddWithValue("@ascii", ascii);
    cmd.Parameters.AddWithValue("@nama", nama);
    cmd.ExecuteNonQuery();
    db.CloseConnection();
}

```

4.2.2.3. PreprocessImg.cs

File ini berisi prosedur *preprocessing* sebuah image menjadi representasi ASCII-nya. Berdasarkan [referensi](#) yang didapatkan pada spesifikasi, sebuah pixel akan diubah menjadi representasi binernya, dalam bentuk 0 dan 1. Dan

dari 8 bit ini akan dikonversi lagi menjadi ASCII nya. Sebuah pixel diubah menjadi 8 bit, artinya sebuah pixel akan mewakili sebuah karakter ASCII. Untuk mempercepat proses pencocokan, kami mengambil pattern sebanyak 30 karakter ASCII dari ASCII yang didapatkan dari gambar input. Awalnya kami melakukan pengambilan 30 karakter ini pada tengah image yang sudah di preprocessed (sudah dipotong bagian putihnya). Akan tetapi, setelah dilakukan beberapa kali testing, kami temukan bahwa metode ini kurang optimal karena sering kali kami mengambil pattern di titik yang salah. Pattern yang kami ambil rata-rata berasal dari lokasi yang kerap kali terjadi perubahan minor. Hal ini menyebabkan algoritma BM dan KMP menjadi sia-sia, sebab mereka adalah metode pattern matching yang exact, sehingga perubahan minor ini menjadikan pattern tidak ditemukan. Beberapa kali kami melakukan percobaan dan algoritma KMP dan BM hanya berhasil digunakan pada image sidik jari yang memang sudah terdapat pada database (sama persis). Maka dari itu, kami memutuskan untuk mengambil 2×30 karakter ASCII. Tiga puluh karakter yang pertama akan kami ambil dari baris $\frac{1}{3}$ image input, dan 30 karakter sisanya pada $\frac{3}{4}$ image input. Kami akan melakukan pencocokan dengan 30 karakter pertama, dan jika gagal akan dicoba 30 karakter selanjutnya. Hal ini bertujuan untuk 2 kali filtering. Setelah melakukan testing dan analisis lebih lanjut, metode baru yang kami terapkan ini terbukti lebih efektif. Penggunaan algoritma KMP dan BM juga jauh lebih sering digunakan daripada terus-menerus mengandalkan Levenshtein saja dan mempercepat waktu pemrosesan.

```

1  using System;
2  using System.Drawing;
3  using System.Text;
4  using Emgu.CV;
5  using Emgu.CV.CvEnum;
6  using Emgu.CV.Features2D;
7  using Emgu.CV.Structure;
8  using Emgu.CV.Util;
9
10 namespace WinFormsApp1
11 {
12     public static class Preprocessing
13     {
14         public static char ConvertIntensityToAsciiChar(byte intensity)
15         {
16             const byte asciiStart = 32;
17             const byte asciiEnd = 126;
18             const byte asciiRange = asciiEnd - asciiStart + 1;
19             return (char)(asciiStart + (intensity * asciiRange / 256));
20         }

```

Metode ConvertIntensityToAsciiChar akan melakukan konversi nilai intensity sebuah pixel kepada karakter ASCII-nya.

```

1 // this is a image cropper so we can only obtain the main focus of the image
2 public static Image<byte> CropTheImage(Image<byte>, byte> image)
3 {
4     // Konversi ke greyscale, siapatahu blm bnw
5     Image<Gray, byte> grayImage = image.Convert<Gray, byte>();
6
7     // Pasang threshold
8     Image<Binary, byte> binaryImage = new Image<Binary, byte>(grayImage.Size);
9     CvInvoke.Threshold(grayImage, binaryImage, 127, 255, ThresholdType.Binary);
10
11    // Find contour buat cropping white area yang tidak relevan
12    VectorOfVectorOfPoint contours = new VectorOfVectorOfPoint();
13    Mat hierarchy = new Mat();
14    CvInvoke.FindContours(binaryImage, contours, hierarchy, RetrType.External, ChainApproxMethod.ChainApproxSimple);
15
16    // Cari contour yang terbesar alias area fingerprint
17    double maxArea = -89;
18    int maxAreaContourIndex = -1;
19    for (int i = 0; i < contours.Size; i++)
20    {
21        double area = CvInvoke.ContourArea(contours[i]);
22        if (area > maxArea)
23        {
24            maxArea = area;
25            maxAreaContourIndex = i;
26        }
27    }
28
29    // Cari petak bounds yang ngewakilin area contour
30
31    // No contours found
32    if (maxAreaContourIndex == -1) return image;
33    Rectangle boundingRect = CvInvoke.BoundingRectangle(contours[maxAreaContourIndex]);
34
35    // Crop the image
36    Image<byte>, byte> croppedImage = image.Copy(boundingRect);
37    return croppedImage;
38}
39

```

Metode CropTheImage akan mempersiapkan image sebelum diproses serta melakukan cropping area supaya proses perbandingan hanya difokuskan kepada inti gambar sidik jari

```

1 // minimal disini menunjukkan apakah kita ingin mengambil size 1x30px dari sebuah image (jadi pattern)
2 public static (string, string) ConvertImageToAscii(bool minimal, Image<byte> image)
3 {
4     Image<gray>, byte> grayImage = CropTheImage(image);
5 
6     // Convert each pixel to ASCII
7     if (!minimal)
8     {
9         System.Diagnostics.Debug.WriteLine("You have chosen to NOT crop the ascii to 30px as a PATTERN");
10        StringBuilder asciiString = new StringBuilder();
11        for (int y = 0; y < grayImage.Rows; y++)
12        {
13            for (int x = 0; x < grayImage.Cols; x++)
14            {
15                byte intensity = grayImage.Data[y, x, 0]; // Get the intensity value
16                char asciiChar = ConvertIntensityToAsciiChar(intensity); // Convert Intensity to ASCII character
17                asciiString.Append(asciiChar);
18            }
19            asciiString.AppendLine();
20        }
21        // Output the ASCII string
22        return (asciiString.ToString(), string.Empty);
23    }
24    else
25    {
26        System.Diagnostics.Debug.WriteLine("You have chosen to crop the ascii to 30px as a PATTERN");
27        int row1 = grayImage.Rows / 4;
28        int row3 = 3 * grayImage.Rows / 4;
29 
30        // Center the 30 characters if the image is wider than 30 columns
31        int startCol = Math.Max(0, (grayImage.Cols - 30) / 2);
32 
33        StringBuilder asciiTop = new StringBuilder();
34        StringBuilder asciiBottom = new StringBuilder();
35 
36        for (int x = startCol; x < startCol + 30 && x < grayImage.Cols; x++)
37        {
38            byte intensityTop = grayImage.Data[row1, x, 0]; // Get the intensity value for the top
39            char asciiCharTop = ConvertIntensityToAsciiChar(intensityTop); // Convert Intensity to ASCII character
40            asciiTop.Append(asciiCharTop);
41 
42            byte intensityBottom = grayImage.Data[row3, x, 0]; // Get the intensity value for the bottom
43            char asciiCharBottom = ConvertIntensityToAsciiChar(intensityBottom); // Convert Intensity to ASCII character
44            asciiBottom.Append(asciiCharBottom);
45        }
46 
47        // Return the ASCII strings for both sections
48        return (asciiTop.ToString(), asciiBottom.ToString());
49    }
50 }

```

Metode ConvertImageToAscii akan menjadi fungsi utama dalam seluruh preprocessing. Metode ini akan memproses setiap pixel yang ada menjadi representasi 8 bit nya lalu mengubah 8 bit tersebut menjadi sebuah karakter ASCII. Pada metode ini terdapat opsi untuk cropping 30 pixel dengan adanya boolean minimum.

4.3. GUI

4.3.1.1. Form1.cs

Form1.cs akan berfungsi sebagai main program yang akan menghubungkan komponen GUI dengan fungsi-fungsi logic yang ada. Sebagai tambahan, pada perbandingan antara ASCII inputan dan database, kami menambahkan multithreading untuk mempercepat algoritma. Awalnya program akan melakukan perbandingan dengan menggunakan KMP dan BM beserta parallel multithread untuk setiap iterasi data dalam database. Akan dipasang sebuah lock pada variable found, resultNama, dan resultPath karena selama algoritma, variabel tersebut yang akan diubah, dan bisa saja terjadi *race condition*. Dengan menggunakan multithreading kami berhasil meningkatkan kecepatan program hingga 2-3 kali lipat. Konsep ini juga kami terapkan pada penerapan algoritma Levenshtein.

```

1 using System;
2 using System.Drawing;
3 using System.Text;
4 using System.Diagnostics;
5 using System.Windows.Forms;
6 using Emgu.CV;
7 using Emgu.CV.CvEnum;
8 using Emgu.CV.Structure;
9 using Emgu.CV.Util;
10 using System.Threading.Tasks;
11
12
13 using WinFormsApp1.Algorithms;
14 using MySql.Data.MySqlClient;
15
16 namespace WinFormsApp1
17 {
18     public partial class Form1 : Form
19     {
20         private bool toggledOn; //status toggle button
21         private String asciiBottom; //ascii text image input yang sudah cropped 30px
22         private String asciiTop; //ascii text image input yang sudah cropped 30px
23         private String resultName; //hasil nama yang didapatkan (ascii)
24         private String resultPath; //path fingerprint di db yang paling mirip
25         private String fullAscii; //ascii image input fulltext dan tidak dicrop
26         private readonly Fingerprints fingerprints; //data fingerprints yang paling mirip
27
28         public Form1()
29         {
30             InitializeComponent();
31             buttonoval2.Click += Buttonoval2_Click;
32             search1.Click += ButtonSearch1_Click;
33             // JANGAN LUPA SESUAIKAN CONNECTION STRING DENGAN DB KAMU
34             string connectionString = "server=localhost;user id=root;password=password;database=fingerprint";
35             Database db = new Database(connectionString);
36
37             fingerprints = new Fingerprints(db);
38
39             /// alter table
40             fingerprints.alterTable();
41             Dummy.GenerateDummy(@"Data.Combine(GetProjectDirectory(), "test"), fingerprints);
42
43             // Load the ascii representation, preprocessing ascii dan memasukkan ke tabel
44             UpdateDatabaseWithAsciiRepresentation();
45         }

```

Inisialisasi komponen, dilanjutkan dengan melakukan dummy database dan pemrosesan insert data representasi ASCII ke database

```

1 // getting the project dir for diff user
2     private string GetProjectDirectory()
3     {
4         string currentDirectory = AppDomain.CurrentDomain.BaseDirectory;
5         string projectDirectory = Path.Combine(
6             (directory.GetParent(currentDirectory).Parent.Parent.Parent.FullName, "...", "...", "..."));
7         return projectDirectory;
8     }

```

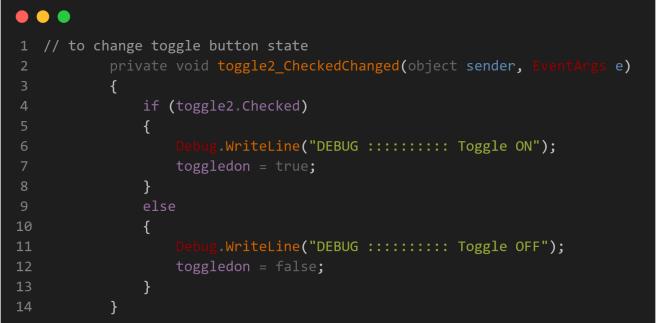
Fungsi ini mengembalikan direktori proyek saat ini, berguna untuk menemukan jalur file relatif terhadap direktori.

```

1 private void UpdateDatabaseWithAsciiRepresentation()
2     {
3         string projectDirectory = GetProjectDirectory();
4
5         (List<string> berkasDatabase, List<string> namaDatabase, List<string> asciiDatabase) = fingerprints.GetAllFingerprintDataSeparated();
6         int id = 0;
7         foreach (var fingerprint in berkasDatabase)
8         {
9             string imagePath = Data.Combine(projectDirectory, fingerprint);
10            if (!File.Exists(imagePath))
11            {
12                Image<byte> image = new Image<byte>[imagePath];
13                string asciiRepresentation = string.Empty;
14                Preprocessing.ConvertImageToAscii(false, image);
15                fingerprints.InsertAscii(namaDatabase[id], asciiRepresentation);
16            }
17            else
18            {
19                Console.WriteLine($"Image file {imagePath} not found.");
20            }
21        }
22    }

```

- Mengambil semua data fingerprint dari database dan memisahkannya menjadi tiga list: berkas, nama, dan ASCII.
- Mengiterasi setiap berkas_citra dan mengonversi gambar pada path tersebut ke representasi ASCII menggunakan metode Preprocessing.ConvertImageToAscii.
- Memasukkan representasi ASCII ke

	dalam database.
 <pre>1 // to change toggle button state 2 private void toggle2_CheckedChanged(object sender, EventArgs e) 3 { 4 if (toggle2.Checked) 5 { 6 Debug.WriteLine("DEBUG :::::::::::: Toggle ON"); 7 toggledon = true; 8 } 9 else 10 { 11 Debug.WriteLine("DEBUG :::::::::::: Toggle OFF"); 12 toggledon = false; 13 } 14 }</pre>	Mengubah status toggle button toggle2. Jika toggle2 dicentang, status toggledon diatur ke true, jika tidak diatur ke false.

```

1  private void ButtonSearch1_Click(object sender, EventArgs e)
2  {
3      // Start the stopwatch for exec time
4      Stopwatch stopwatch = Stopwatch.StartNew();
5      float similarity = 0;
6      // Regex find bio that match
7      (resultnama, resultpath, similarity) = PerformPatternMatching();
8      bios = fingerprints.GetAllBiodataData();
9      Debug.WriteLine("displaying results.....");
10     // Debug.WriteLine("PATH GAMBAR WOY: "+resultpath);
11     bool found = false;
12     if(resultpath != null)
13     {
14         Biodata resultbio = null;
15         int mindist = int.MaxValue;
16         foreach (var item in bios){
17             // KMP
18             if (toggledon){
19                 found = KMPAlgorithme.KMPSearch(resultnama, item>Nama);
20             }
21             // BM
22             else{
23                 found = BoyerMooreAlgorithme.BMSearch(resultnama, item>Nama);
24             }
25             if (found){
26                 Debug.WriteLine("Berhasil nemu nama alay pakai KMP BM");
27                 resultbio = item;
28             }
29         }
30         if(!found){
31             foreach (var item in bios)
32             {
33                 Debug.WriteLine("Berhasil nemu nama alay pakai KMP BM");
34                 string purified = AlayTranslator.translateAlay(item>Nama);
35                 int caldist = Levenshtein.calculateSimilarity(resultnama, purified);
36                 if (caldist < mindist)
37                 {
38                     mindist = caldist;
39                     resultbio = item;
40                 }
41             }
42         }
43         pictureBox3.Image = Image.FromFile(resultpath);
44         pictureBox3.SizeMode = PictureBoxSizeMode.Zoom;
45         labelNama.Text = resultnama;
46         labelNIK.Text = resultbio.NIK;
47         labelTempatLahir.Text = resultbio.TempatLahir;
48         labelTanggalLahir.Text = resultbio.TanggalLahir.ToString("dd/MM/yyyy");
49         labelJenisKelamin.Text = resultbio.JenisKelamin;
50         labelGoldar.Text = resultbio.GolonganDarah;
51         labelAlamat.Text = resultbio.Alamat;
52         labelAgama.Text = resultbio.Agama;
53         labelStatus.Text = resultbio.StatusPerkawinan;
54         labelPekerjaan.Text = resultbio.Pekerjaan;
55         labelKWN.Text = resultbio.Kewarganegaraan;
56         labelKemiripan.Text = Math.Round((decimal)similarity,2).ToString() + "%";
57     }
58     // sidik jari not found, reset display
59     else
60     {
61         Bitmap bitmap = new Bitmap(pictureBox3.Width, pictureBox3.Height);
62         pictureBox3.Image = bitmap;
63         pictureBox3.Invalidate();
64         labelNama.Text = "None :";
65         labelNIK.Text = "None :(";
66         labelTempatLahir.Text = "None :(";
67         labelTanggalLahir.Text = "None :(";
68         labelJenisKelamin.Text = "None :(";
69         labelGoldar.Text = "None :(";
70         labelAlamat.Text = "None :(";
71         labelAgama.Text = "None :(";
72         labelStatus.Text = "None :(";
73         labelPekerjaan.Text = "None :(";
74         labelKWN.Text = "None :(";
75         labelKemiripan.Text = "0 %";
76     }
77     stopwatch.Stop();
78     // Format the elapsed time as a string
79     labelEksekusi.Text = stopwatch.ElapsedMilliseconds.ToString() + " ms";
80
81     Debug.WriteLine("DEBUG Execution Time: " + stopwatch.ElapsedMilliseconds.ToString());
82
83 }

```

- Memulai pencarian sidik jari menggunakan algoritma pencocokan pola sambil mengukur runtime
- Menggunakan metode PerformPattern Matching untuk mencari sidik jari yang paling cocok.
- Mencari biodata untuk nama yang bersesuaian
- Menampilkan hasil pencarian jika ditemukan atau mengatur ulang tampilan jika tidak ditemukan.

```

1 private void ButtonOval2_Click(object sender, EventArgs e)
2 {
3     openFileDialog openFileDialog = new OpenFileDialog();
4     openFileDialog.Filter = "Fingerprint Imgs (*.BMP)|*.BMP";
5     openFileDialog.FilterIndex = 1;
6     openFileDialog.RestoreDirectory = true;
7
8     if (openFileDialog.ShowDialog() == DialogResult.OK)
9     {
10         Debug.WriteLine("DEBUG----- GAMBAR DITERIMA DAN DI LOAD");
11         // Get the selected file name
12         string selectedFileName = openFileDialog.FileName;
13
14         // Load the selected image into pictureBox2, display image input
15         pictureBox2.Image = Image.FromFile(selectedFileName);
16         pictureBox2.SizeMode = PictureBoxSizeMode.Zoom;
17
18         // Load the image
19         Image<Byte, byte> image = new Image<Byte, byte>(<selectedFileName>);
20         (asciitop, ascibottom) = Preprocessing.ConvertImageToAscii(true, image);
21         (fullascii, string empty) = Preprocessing.ConvertImageToAscii(false, image);
22
23         Debug.WriteLine("DEBUG----- PROGRAM SELESAI EKSEKUSI");
24     }
25 }

```

```

1 // Will Perform patternmatching algo based on the toggle button
2 // will return the name and image path attribute if found
3 private (string name, string imagePath, float similarity) PerformPatternMatching()
4 {
5     Debug.WriteLine("DEBUG----- MULAI PATTERN MATCHING");
6     (List<string> berkasDatabase, List<string> nimDatabase, List<string> asciiDatabase) = fingerprints.getAllFingerprintDataSeparates();
7     string projectDirectory = getProjectDirectory();
8
9     object lockObject = new object();
10    bool localFound = false;
11    string resultName = null;
12    string resultPath = null;
13
14    // Parallel pattern matching with KMP and BM
15    Parallel.For(0, asciiDatabase.Count, (i, state) =>
16    {
17        string name = nimDatabase[i];
18        string imagePath = berkasDatabase[i];
19        bool localFound = false;
20
21        if (toggedon)
22        {
23            Debug.WriteLine("Ngacak pakai KMP TOP bro");
24            localFound = KMPAlgorithm.KMPSearch(asciitop, asciiDatabase[i]);
25            if (!localFound)
26            {
27                Debug.WriteLine("Ngacak pakai KMP BOTTOM bro");
28                localFound = KMPAlgorithm.KMPSearch(ascibottom, asciiDatabase[i]);
29            }
30        }
31        else
32        {
33            Debug.WriteLine("Ngacak pakai BM TOP bro");
34            localFound = BoyerMooreAlgorithm.BMSearch(asciitop, asciiDatabase[i]);
35            if (!localFound)
36            {
37                Debug.WriteLine("Ngacak pakai BM BOTTOM bro");
38                localFound = BoyerMooreAlgorithm.BMSearch(ascibottom, asciiDatabase[i]);
39            }
40        }
41
42        if (localFound)
43        {
44            lock (lockObject)
45            {
46                if (!found)
47                {
48                    found = true;
49                    resultName = name;
50                    resultPath = imagePath.Combine(projectDirectory, imagePath);
51                    state.Stop(); // Stop other parallel iterations since we found a match
52                }
53            }
54        }
55    });
56
57    if (found)
58    {
59        return (resultName, resultPath, 100);
60    }
61
62    // If no match is found, use Levenshtein distance
63    int leven = 1000000000;
64    int mindist = -1;
65    Debug.WriteLine("-----");
66    Debug.WriteLine("Jumlah element: " + asciiDatabase.Count);
67
68    Parallel.For(0, asciiDatabase.Count, i =>
69    {
70        int leven = levenshtein.calculateSimilarity(fullascii, asciiDatabase[i]);
71        Debug.WriteLine("NMU NGACIK LEVEN sil " + berkasDatabase[i] + ":" + leven);
72
73        lock (lockObject)
74        {
75            if (leven < mindist)
76            {
77                mindist = leven;
78                idbest = i;
79            }
80        }
81    });
82
83    Debug.WriteLine("-----");
84    Debug.WriteLine("Ini pattern yang kamu pakai sebagai input");
85
86    float converteddist = ((asciiDatabase[idbest].Length - mindist) / (float)asciiDatabase[idbest].Length) * 100;
87    // Set threshold ke 70, kalau lebih kecil bakal ga ketemu hasilnya
88    if (converteddist < 70)
89    {
90        return (null, null, 0);
91    }
92
93    string levensimagedirectory = imagePath.Combine(projectDirectory, berkasDatabase[idbest]);
94    return (nameDatabase[idbest], levensimagedirectory, converteddist);
95 }
96
97

```

Memuat gambar yang dipilih ke dalam pictureBox2 dan mengonversi gambar ke representasi ASCII atas dan bawah menggunakan metode Preprocessing.ConvertImageToAscii.

- Melakukan pencocokan pola antara ASCII sidik jari masukan dengan yang ada di database.
- Menggunakan algoritma KMP atau BM berdasarkan masukan.
- Jika tidak ada kecocokan yang ditemukan, menggunakan Levenshtein untuk mencari hasil termirip.
- Mengembalikan nama dan jalur gambar yang paling cocok beserta persentase kemiripannya.

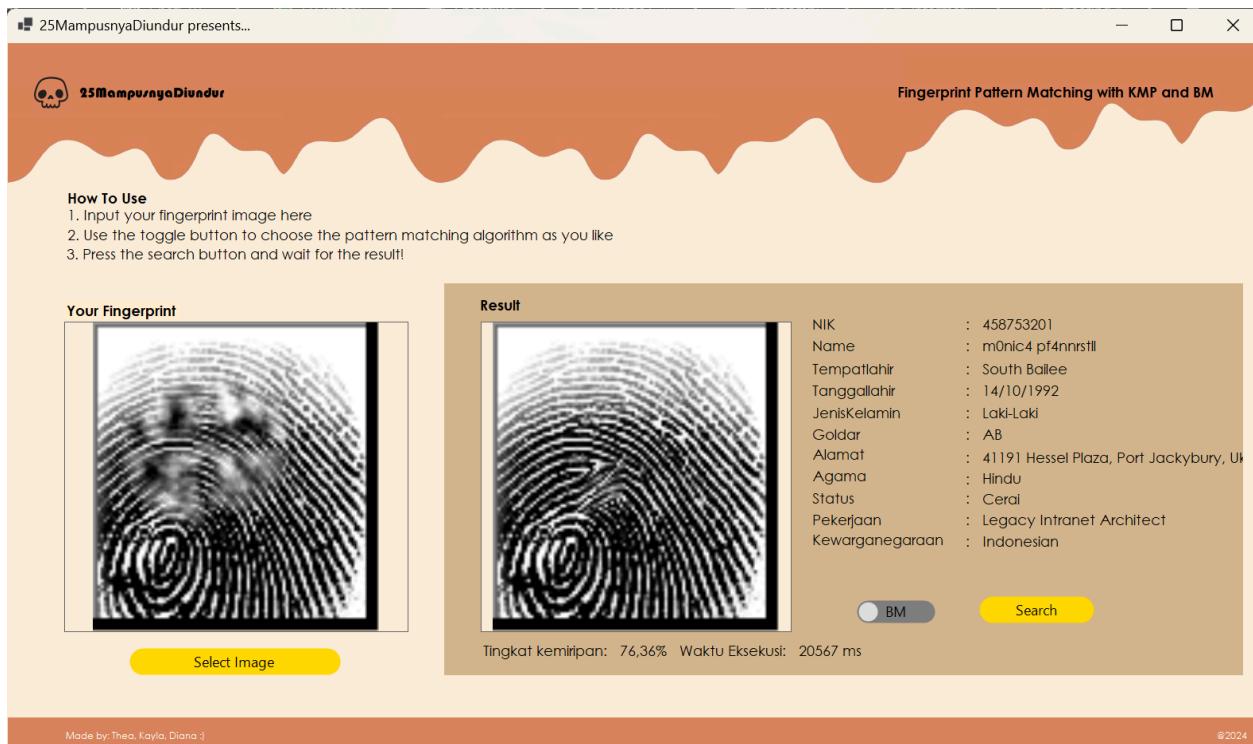
4.4. Struktur Data dan Implementasi

```
.  
└── .vs  
└── src  
    ├── WinFormsApp1  
    │   ├── .vs  
    │   └── Algorithms  
    │       ├── AlayTranslator.cs  
    │       ├── BM.cs  
    │       ├── Dummy.cs  
    │       ├── KMP.cs  
    │       ├── Levenshtein.cs  
    │       ├── Preprocessing.cs  
    │       ├── bin  
    │       ├── obj  
    │       ├── Properties  
    │       └── Resources  
    │           ├── Doodle_Skull.png  
    │           ├── Water_Scarcity-removebg-preview.png  
    └── Toggle  
        ├── ButtonOval.cs  
        ├── RoundedPanel.cs  
        ├── Search.cs  
        ├── Toggle.cs  
        ├── Biodata.cs  
        ├── Database.cs  
        ├── Fingerprint.cs  
        ├── FingerprintDB.cs  
        ├── Form1.cs  
        ├── Form1.Designer.cs  
        ├── Form1.resx  
        └── Program.cs  
            ├── WinFormsApp1.csproj  
            └── WinFormsApp1.csproj.user
```

4.5. Tata Cara Penggunaan Program

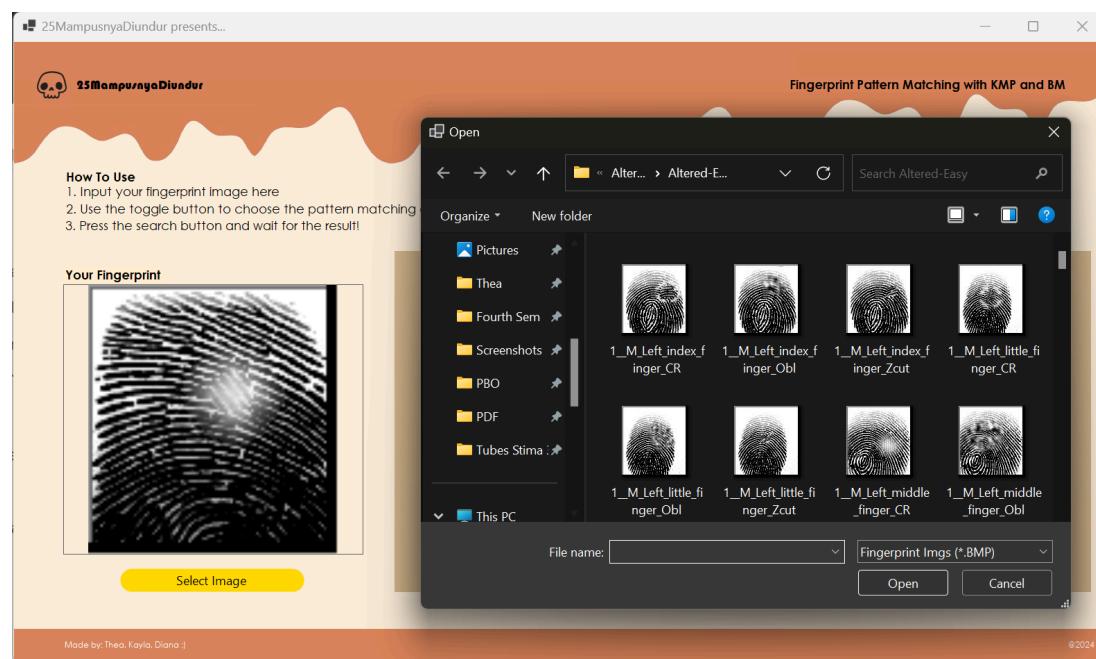
1. Buka IDE Visual Studio dan pilih solution *WinformsApp1.sln*.
2. Tekan tombol execute pada bagian atas menu untuk build solution.
3. Masukkan gambar fingerprint yang akan dicari dengan menekan tombol Select Image.
4. Pilih algoritma yang akan digunakan lalu tekan tombol Search untuk memulai pencarian.
5. Tampilan hasil program dapat berupa:
 - Gambar sidik jari dari database yang paling mirip, biodata, beserta waktu eksekusi dan persentase kemiripannya.
 - Kotak image kosong karena tidak ditemukan gambar yang mirip (kemiripan < 70%).

4.6. Tampilan Antarmuka



Gambar 4.5.1 Tampilan Utama Aplikasi

Antarmuka aplikasi akan terdiri dari bagian petunjuk penggunaan aplikasi (How To Use), input gambar dan *display*-nya, serta *section* hasil yang terdiri dari image sidik jari yang mirip dan hasil biodata yang diambil dari database. Pengguna juga dapat memilih algoritma yang digunakan antara Knuth Morris Pratt dan Boyer Moore dengan menggunakan tombol toggle. Aplikasi juga akan menampilkan lama eksekusi dalam milisekon serta tingkat kemiripan gambar sidik jari yang diinput dengan gambar-gambar sidik jari pada database dalam persentase. Aplikasi ini akan membatasi jenis file gambar yang bisa dimasukkan pengguna berupa bitmap (bmp).



Gambar 4.5.2 Tampilan Input Gambar

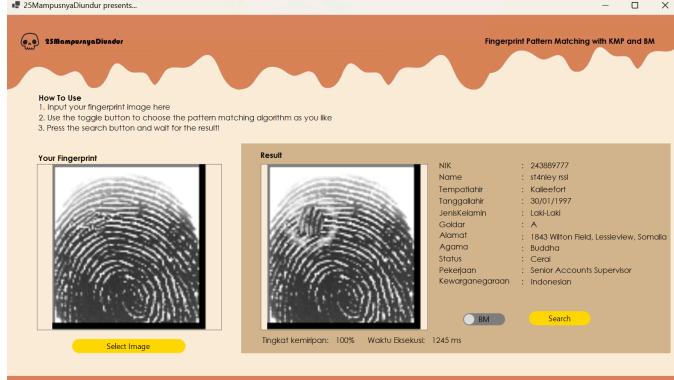
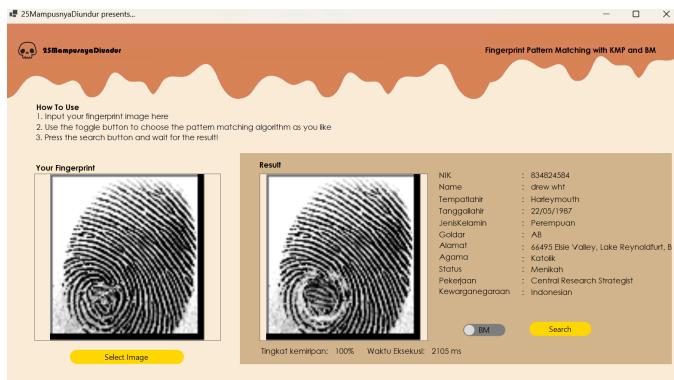
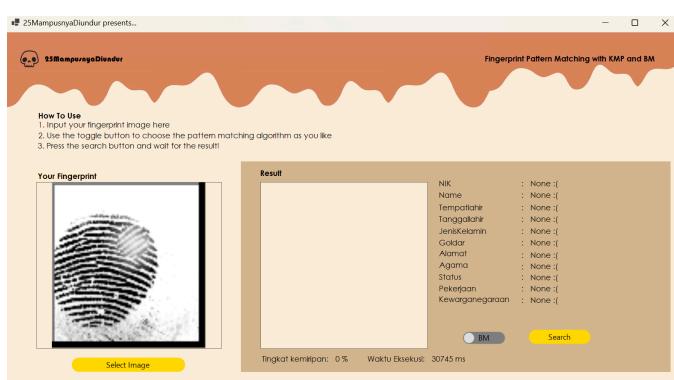
4.7. Hasil Pengujian

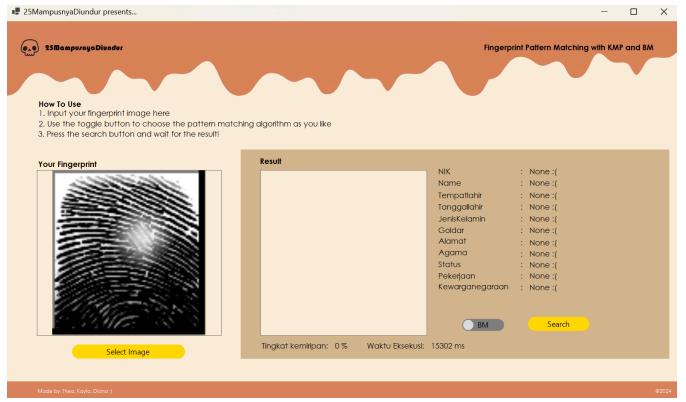
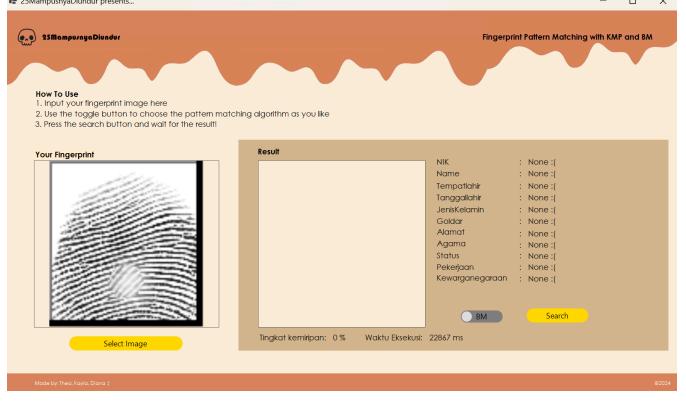
Hasil Pengujian kedua algoritma dilakukan dengan test case yang sama, untuk membandingkan hasil persentase kemiripan serta lama waktu eksekusinya. Perlu dicantumkan bahwa tingkat kemiripan algoritma Boyer Moore dan Knuth Morris Pratt akan dianggap 100%, tingkat kemiripan Levenshtein akan ditampilkan sesuai dengan kemiripan kedua gambar, serta threshold kemiripan Levenshtein akan dibatasi di atas angka 70. Kemiripan di bawah 70% akan dianggap sebagai gambar sidik jari (individu) yang berbeda.

4.7.1. Pengujian Boyer Moore

Tabel 4.6.1.1 Hasil Pengujian BM

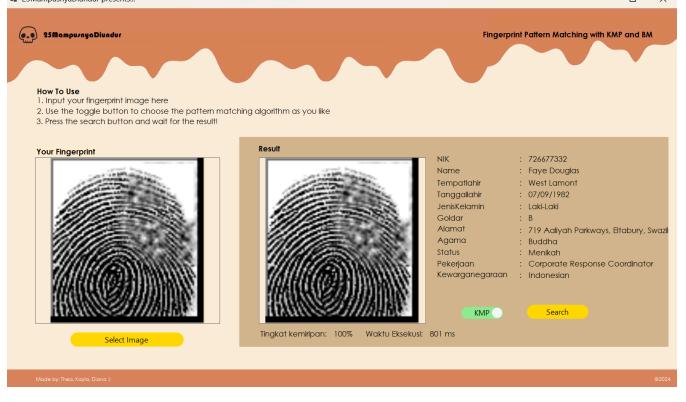
Keterangan Singkat	Input	Keluaran																						
Kasus gambar masukan sudah terdapat pada database.		<p>The screenshot shows the application's main window titled "Fingerprint Pattern Matching with KMP and BM". It displays two fingerprint images side-by-side: "Your Fingerprint" and "Result". Both images show a clear circular pattern. Below the images, there is a yellow button labeled "Select Image". To the right of the images, there is a search result table:</p> <table border="1"> <tr> <td>NIK</td><td>: 72667732</td> </tr> <tr> <td>Name</td><td>: Faye Douglas</td> </tr> <tr> <td>Tempatlahir</td><td>: West Lamont</td> </tr> <tr> <td>Tanggalahir</td><td>: 07/09/1982</td> </tr> <tr> <td>Jeniskelamin</td><td>: Laki-Laki</td> </tr> <tr> <td>Golongan</td><td>: O+</td> </tr> <tr> <td>Alamat</td><td>: 719 Adlyah Parkway, Bhatiaw, Swad</td> </tr> <tr> <td>Agama</td><td>: Budha</td> </tr> <tr> <td>Status</td><td>: Menikah</td> </tr> <tr> <td>Pekerjaan</td><td>: Corporate Response Coordinator</td> </tr> <tr> <td>Kewarganegaraan</td><td>: Indonesian</td> </tr> </table> <p>At the bottom of the interface, it says "Tingkat Kemiripan: 100% Waktu Eksekusi: 404 ms".</p>	NIK	: 72667732	Name	: Faye Douglas	Tempatlahir	: West Lamont	Tanggalahir	: 07/09/1982	Jeniskelamin	: Laki-Laki	Golongan	: O+	Alamat	: 719 Adlyah Parkway, Bhatiaw, Swad	Agama	: Budha	Status	: Menikah	Pekerjaan	: Corporate Response Coordinator	Kewarganegaraan	: Indonesian
NIK	: 72667732																							
Name	: Faye Douglas																							
Tempatlahir	: West Lamont																							
Tanggalahir	: 07/09/1982																							
Jeniskelamin	: Laki-Laki																							
Golongan	: O+																							
Alamat	: 719 Adlyah Parkway, Bhatiaw, Swad																							
Agama	: Budha																							
Status	: Menikah																							
Pekerjaan	: Corporate Response Coordinator																							
Kewarganegaraan	: Indonesian																							
Kasus gambar terdapat perbedaan kecil pada bagian tengah		<p>The screenshot shows the application's main window titled "Fingerprint Pattern Matching with KMP and BM". It displays two fingerprint images side-by-side: "Your Fingerprint" and "Result". Both images show a clear circular pattern with minor central variations. Below the images, there is a yellow button labeled "Select Image". To the right of the images, there is a search result table:</p> <table border="1"> <tr> <td>NIK</td><td>: 730902023</td> </tr> <tr> <td>Name</td><td>: Clark Daniel</td> </tr> <tr> <td>Tempatlahir</td><td>: Koepinmouth</td> </tr> <tr> <td>Tanggalahir</td><td>: 30/01/2002</td> </tr> <tr> <td>Jeniskelamin</td><td>: Tuan</td> </tr> <tr> <td>Golongan</td><td>: O-</td> </tr> <tr> <td>Alamat</td><td>: 94417 Edman Park, West Chisboroug</td> </tr> <tr> <td>Agama</td><td>: Katolik</td> </tr> <tr> <td>Status</td><td>: Belum Menikah</td> </tr> <tr> <td>Pekerjaan</td><td>: Corporate Optimization Planner</td> </tr> <tr> <td>Kewarganegaraan</td><td>: Indonesian</td> </tr> </table> <p>At the bottom of the interface, it says "Tingkat Kemiripan: 100% Waktu Eksekusi: 759 ms".</p>	NIK	: 730902023	Name	: Clark Daniel	Tempatlahir	: Koepinmouth	Tanggalahir	: 30/01/2002	Jeniskelamin	: Tuan	Golongan	: O-	Alamat	: 94417 Edman Park, West Chisboroug	Agama	: Katolik	Status	: Belum Menikah	Pekerjaan	: Corporate Optimization Planner	Kewarganegaraan	: Indonesian
NIK	: 730902023																							
Name	: Clark Daniel																							
Tempatlahir	: Koepinmouth																							
Tanggalahir	: 30/01/2002																							
Jeniskelamin	: Tuan																							
Golongan	: O-																							
Alamat	: 94417 Edman Park, West Chisboroug																							
Agama	: Katolik																							
Status	: Belum Menikah																							
Pekerjaan	: Corporate Optimization Planner																							
Kewarganegaraan	: Indonesian																							
Kasus gambar terdapat perbedaan kecil pada bagian tengah		<p>The screenshot shows the application's main window titled "Fingerprint Pattern Matching with KMP and BM". It displays two fingerprint images side-by-side: "Your Fingerprint" and "Result". Both images show a clear circular pattern with minor central variations. Below the images, there is a yellow button labeled "Select Image". To the right of the images, there is a search result table:</p> <table border="1"> <tr> <td>NIK</td><td>: 452644156</td> </tr> <tr> <td>Name</td><td>: Stewart Armstrong</td> </tr> <tr> <td>Tempatlahir</td><td>: Lake Max</td> </tr> <tr> <td>Tanggalahir</td><td>: 15/07/1964</td> </tr> <tr> <td>Jeniskelamin</td><td>: Pria</td> </tr> <tr> <td>Golongan</td><td>: O-</td> </tr> <tr> <td>Alamat</td><td>: 335 Feny Pike, Beentad, Mayotte</td> </tr> <tr> <td>Agama</td><td>: Konghucu</td> </tr> <tr> <td>Status</td><td>: Cerai</td> </tr> <tr> <td>Pekerjaan</td><td>: Investor Branding Supervisor</td> </tr> <tr> <td>Kewarganegaraan</td><td>: Indonesian</td> </tr> </table> <p>At the bottom of the interface, it says "Tingkat Kemiripan: 100% Waktu Eksekusi: 466 ms".</p>	NIK	: 452644156	Name	: Stewart Armstrong	Tempatlahir	: Lake Max	Tanggalahir	: 15/07/1964	Jeniskelamin	: Pria	Golongan	: O-	Alamat	: 335 Feny Pike, Beentad, Mayotte	Agama	: Konghucu	Status	: Cerai	Pekerjaan	: Investor Branding Supervisor	Kewarganegaraan	: Indonesian
NIK	: 452644156																							
Name	: Stewart Armstrong																							
Tempatlahir	: Lake Max																							
Tanggalahir	: 15/07/1964																							
Jeniskelamin	: Pria																							
Golongan	: O-																							
Alamat	: 335 Feny Pike, Beentad, Mayotte																							
Agama	: Konghucu																							
Status	: Cerai																							
Pekerjaan	: Investor Branding Supervisor																							
Kewarganegaraan	: Indonesian																							

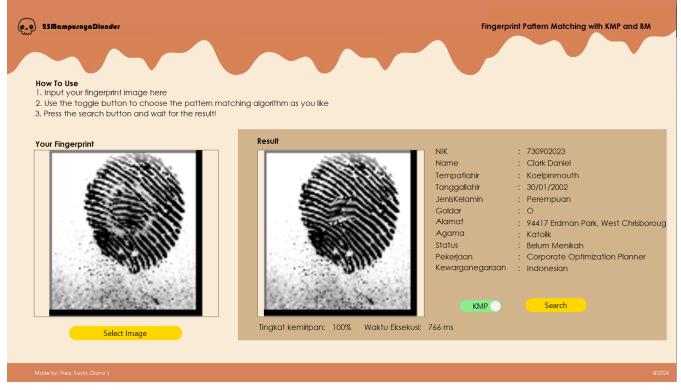
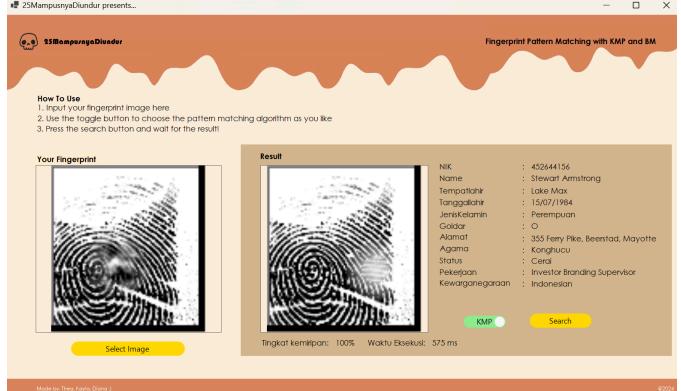
<p>Kasus gambar terdapat perbedaan kecil pada bagian atas</p>		
<p>Kasus gambar terdapat perbedaan kecil pada bagian bawah</p>		
<p>Kasus gambar yang benar-benar berbeda dari gambar di database</p>		

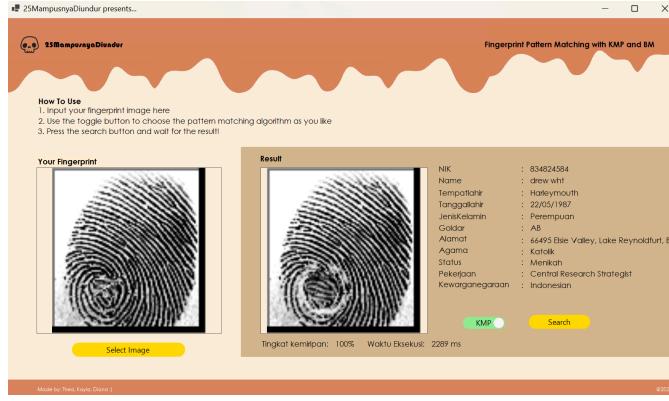
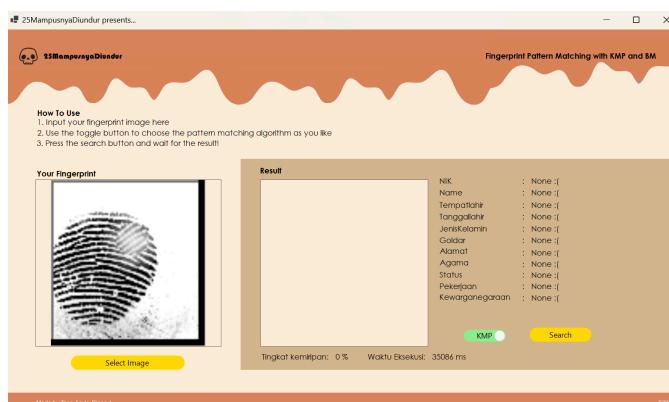
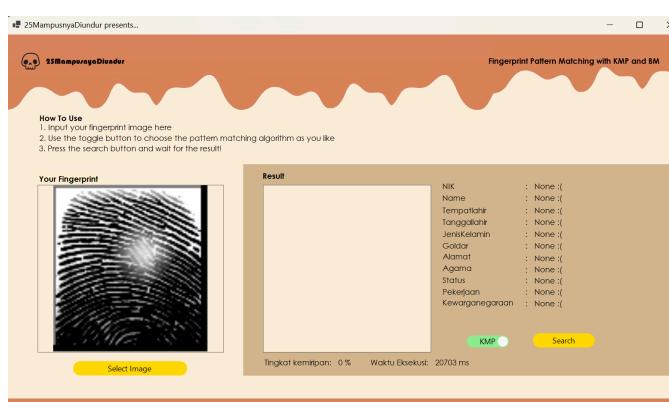
Kasus gambar yang benar-benar berbeda dari gambar di database		
Kasus gambar yang benar-benar berbeda dari gambar di database		

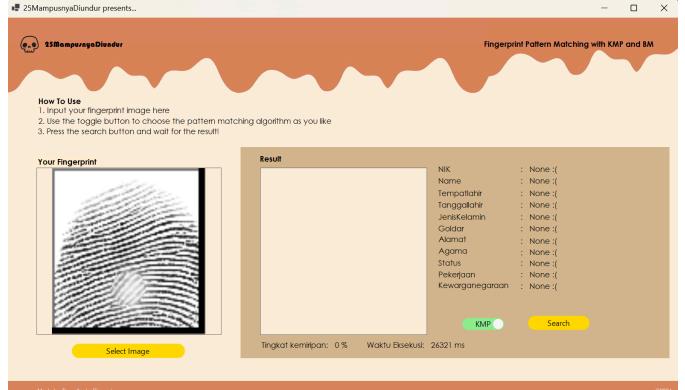
4.7.2. Pengujian KMP

Tabel 4.6.2.1 Hasil Pengujian KMP

Keterangan Singkat	Input	Keluaran
Kasus gambar masukan sudah terdapat pada database.		

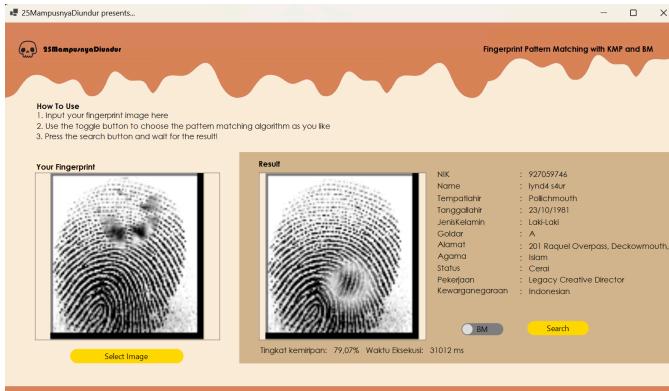
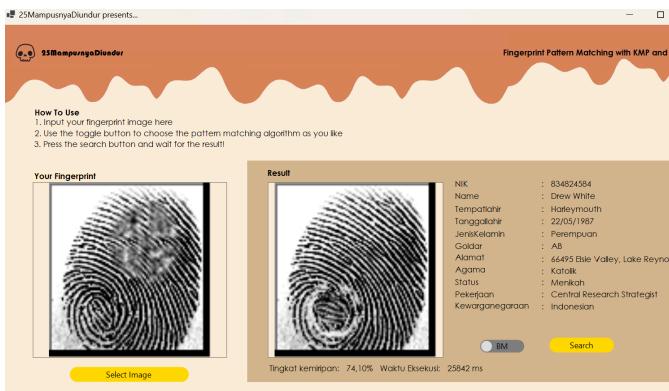
<p>Kasus gambar terdapat perbedaan kecil pada bagian tengah</p>		
<p>Kasus gambar terdapat perbedaan kecil pada bagian tengah</p>		
<p>Kasus gambar terdapat perbedaan kecil pada bagian atas</p>		

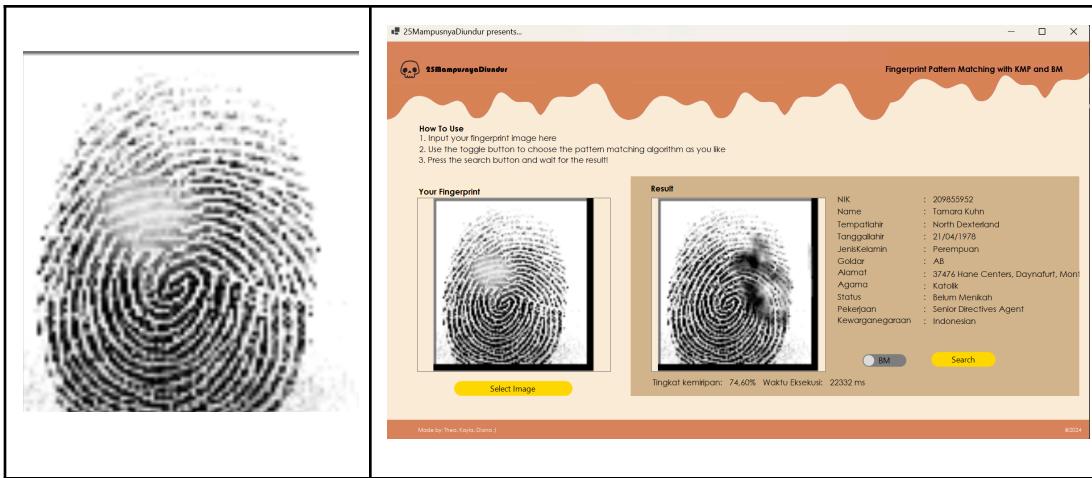
<p>Kasus gambar terdapat perbedaan kecil pada bagian bawah</p>		
<p>Kasus gambar yang benar-benar berbeda dari gambar di database</p>		
<p>Kasus gambar yang benar-benar berbeda dari gambar di database</p>		

Kasus gambar yang benar-benar berbeda dari gambar di database		
---	---	--

4.7.3. Pengujian Levenshtein

Tabel 4.6.3.1 Hasil Pengujian Levenshtein

Input	Keluaran
	
	



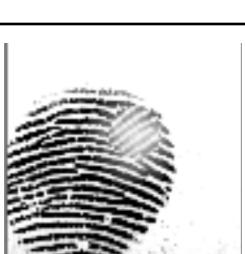
***disclaimer:** pada beberapa screenshot di bagian ini, terdapat biodata yang menunjukkan nama alay karena belum sempat diganti saat proses penulisan laporan.

4.8. Analisis Hasil Pengujian

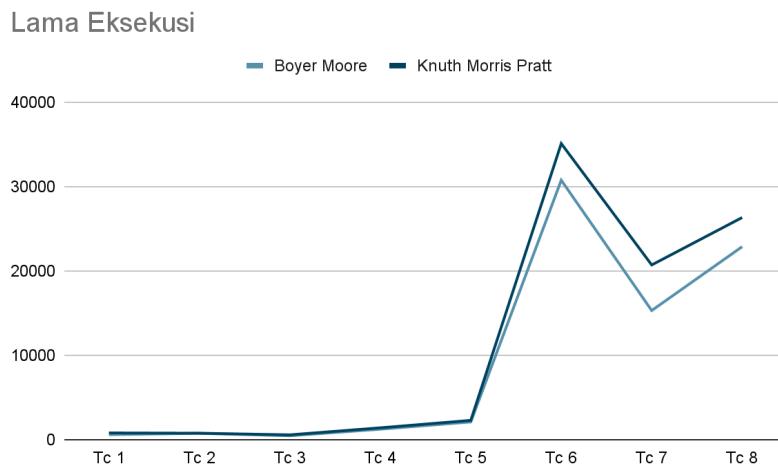
Berdasarkan pengujian yang telah dilakukan pada beberapa test case, kita dapat memetakan perbedaan kedua algoritma dalam tabel perbandingan sebagai berikut:

Tabel 4.6.2.1 Perbandingan Kompleksitas Waktu Algoritma (ms)

No.	Gambar Masukan	Boyer Moore	Knuth Morris Pratt
1.		604	801
2.		758	766

3.		466	575
4.		1245	1408
5.		2105	2289
6.		30745	35086
7.		15302	20703

8.		22867	26321
		9261,5	10993,63



Gambar 4.7.1 Grafik Perbandingan Kompleksitas Waktu

Berdasarkan analisis, rata-rata waktu eksekusi algoritma KMP adalah 10 detik, sementara algoritma BM memiliki rata-rata waktu eksekusi 9 detik. Perbedaan utama antara kedua algoritma ini terletak pada cara mereka menghindari perbandingan yang tidak perlu. Algoritma BM menggunakan heuristik *last occurrence* yang memungkinkan lompatan panjang saat terjadi *mismatch*, sehingga seringkali lebih cepat dalam praktik, terutama pada teks yang panjang dan pola yang jarang muncul. Di sisi lain, KMP menghindari perbandingan berulang dengan menggunakan tabel lompatan (*failure function*), yang lebih efektif pada pola yang memiliki banyak pengulangan.

Berdasarkan pengujian yang kami lakukan, didapatkan bahwa selisih lama waktu eksekusi tidak terlalu berbeda jauh antara KMP dan BM. Hal ini kemungkinan disebabkan oleh pola yang dicari memiliki distribusi yang tidak terlalu jarang atau terlalu

sering sehingga kedua algoritma dapat menunjukkan kinerja yang serupa. Ukuran teks dan pola yang diuji juga dapat mempengaruhi kinerja. Algoritma BM dapat sangat efektif pada dengan alfabet beragam. Dalam konteks pemrosesan sidik jari yang memiliki representasi ASCII yang unik, BM bisa saja lebih unggul. Begitu pula, pada teks yang memiliki banyak pengulangan, keunggulan KMP dalam menghindari perbandingan berulang juga bisa tidak terlalu signifikan. Perbedaan yang tidak terlalu signifikan ini kemungkinan juga disebabkan oleh penggunaan multithreading pada pencocokan string. Ketika menggunakan multithreading untuk pencocokan pola, tidak ada jaminan bahwa bagian tertentu dari data akan selalu diperiksa terlebih dahulu dibandingkan bagian lainnya. Urutan eksekusi akan bergantung pada bagaimana thread dijadwalkan oleh sistem operasi. Thread yang bekerja pada data di tengah-tengah mungkin menyelesaikan tugasnya lebih cepat daripada thread yang bekerja pada bagian awal, tergantung pada ukuran dan kompleksitas bagian data tersebut. Akhirnya, kecepatan menemukan data yang cocok bisa berbeda-beda.

Akan tetapi, terlepas dari kecepatan eksekusi algoritma, hasil pencocokan yang kami dapatkan sudah sesuai dengan yang diharapkan. Pengambilan gambar yang sesuai pada database juga sudah tepat. Pada contoh tabel pengujian Levenshtein, kami mengambil beberapa contoh kasus dengan perbedaan area image pada bagian atas dan bawah. Hal ini menyebabkan metode pencocokan 30 karakter ASCII bagian atas dan bawah image menjadi sia-sia, sebab keduanya akan berbeda. Maka dari itu, akan digunakan algoritma Levenshtein yang membandingkan keseluruhan teks ASCII dari image input dan database. Kami tidak menggunakan 30 karakter ASCII untuk dibandingkan karena dengan membandingkan keseluruhan teks ASCII akan menghasilkan perbedaan Levenshtein yang lebih akurat.

BAB V

KESIMPULAN DAN SARAN

5.1. Kesimpulan

Tugas besar 3 IF2211 Strategi Algoritma mengungkit masalah mengenai pattern matching dalam deteksi fingerprint, di mana kita mendalami algoritma Knuth Morris Pratt (KMP) dan Boyer-Moore (BM) untuk mencocokkan pola sidik jari seseorang dan sidik jari dari database. Berdasarkan analisis, kami dapat menyimpulkan bahwa algoritma BM secara rata-rata dapat menyelesaikan pencarian sedikit lebih cepat dibandingkan algoritma KMP karena sifatnya yang melakukan lompatan panjang saat terjadi mismatch. Namun, perbedaan waktu eksekusi antara kedua algoritma ini tidak signifikan dalam pengujian kami, hanya berbeda sekitar 1 detik. Ini disebabkan oleh distribusi pola yang tidak terlalu jarang atau sering, serta penggunaan multithreading yang menyebabkan urutan pemeriksaan data menjadi tidak teratur dan bergantung pada penjadwalan thread oleh sistem operasi.

Selain itu, kecepatan eksekusi kedua algoritma juga dipengaruhi oleh ukuran teks dan pola yang diuji. Dalam konteks pemrosesan sidik jari yang memiliki representasi ASCII yang unik, algoritma BM menunjukkan keunggulan karena dapat menangani alfabet beragam dengan lebih efektif. Namun, pada teks yang memiliki banyak pengulangan, keunggulan KMP dalam menghindari perbandingan berulang menjadi tidak terlalu signifikan. Penggunaan algoritma Levenshtein juga diperlukan untuk kasus di mana bagian atas dan bawah dari image fingerprint berbeda, sehingga perbandingan keseluruhan teks ASCII menghasilkan perbedaan yang lebih akurat. Dengan demikian, hasil pencocokan yang kami dapatkan sudah sesuai dengan yang diharapkan, dan pengambilan gambar yang sesuai pada database juga sudah tepat, menunjukkan bahwa kombinasi dari berbagai algoritma ini efektif dalam mendeteksi fingerprint dengan akurat.

5.2. Saran

Selama penggerjaan tugas besar ini kami memiliki beberapa saran untuk penggerjaan tugas besar berikutnya, di antaranya:

- Mencari referensi lebih banyak lagi sebagai bahan literasi penambah wawasan.

- Banyak eksplor terkait hal-hal yang baru kita dapatkan untuk mempermudah penggerjaan, semisal dalam hal pembuatan GUI desktop dan penggunaan sintaks dalam C#.

5.3. Refleksi

Tugas Besar 3 IF2211 Strategi Algoritma Semester II Tahun 2023/2024 mengajarkan kami banyak hal, terutama dalam pembuatan GUI desktop application menggunakan WinForm. Kami juga mempelajari lebih mendalam mengenai aplikasi algoritma Knuth Morris Pratt, Boyer Moore, dan Regex dalam dunia nyata. Kami juga menambah wawasan mengenai pemrosesan fingerprint dan bagaimana ketiga konsep algoritma ini bisa digunakan untuk membandingkan data sidik jari.

LAMPIRAN

Repository

https://github.com/pandaandsushi/Tubes3_25MampusnyaDiundur

Video

<https://youtu.be/r5ptTo-pJP0>

DAFTAR PUSTAKA

Asisten, S. A. (2024). Tugas Besar 3 IF2211 Strategi Algoritma 2024. docx. Diakses dari Google Dokumen: [Spesifikasi Tugas Besar 3 Stima 2023/2024.docx](#)

IAmTimCorey. (2022, 8 Agustus). Intro to Windows Forms (WinForms) in .NET 6 [Video]. YouTube. [Intro to Windows Forms \(WinForms\) in .NET 6](#)

Munir, R. (2024). Pencocokan String (String/Pattern Matching). Diakses dari Homepage Rinaldi Munir: [Pencocokan String \(String/Pattern Matching\)](#)

Munir, R. (2024). String Matching dengan Regular Expression. Diakses dari Homepage Rinaldi Munir: [String Matching dengan Regular Expression](#)

N. Kanchana and S. Sarala, "Notice of Retraction: Compressed pattern matching in DNA sequences," 2010 3rd International Conference on Computer Science and Information Technology, Chengdu, China, 2010, pp. 157-160, doi: 10.1109/ICCSIT.2010.5564927.

Shrivastava, Vishal & Sharma, Sumit. (2012). Data Compression of Fingerprint Minutiae. International Journal of Engineering Science and Technology. 4.

S Supatmi and I D Sumitra 2019 IOP Conf. Ser.: Mater. Sci. Eng. 662 022040
<https://www.c-sharpcorner.com/article/c-sharp-regex-examples/>