

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

**Semester II Tahun 2023/2024**

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



#### **Kelompok Franku :**

- |  |                 |
|--|-----------------|
| <b>1. Kristo Anugrah</b>               | <b>13522024</b> |
| <b>2. Francesco Michael Kusuma</b>     | <b>13522038</b> |
| <b>3. M. Hanief Fatkhan Nashrullah</b> | <b>13522100</b> |

**PROGRAM STUDI TEKNIK INFORMATIKA**

**SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA**

**INSTITUT TEKNOLOGI BANDUNG 2024**

## DAFTAR ISI

DAFTAR ISI	2
BAB 1	3
Deskripsi Tugas	3
BAB 2	7
Landasan Teori	7
2.1 Algoritma Knuth-Morris-Pratt	7
2.2 Algoritma BM	10
2.3 Regex	10
2.4 Penjelasan teknik pengukuran persentase kemiripan.	10
2.4.1 Algoritma Fast Fourier Transform	10
2.4.2 Density-based Clustering Algorithm	11
2.4.3 Convex Hull	11
2.4.4 Tahapan Pengukuran Persentase Kemiripan	12
2.5 Penjelasan Enkripsi Basis Data	12
2.6 Aplikasi desktop	13
BAB 3	14
Analisis Pemecahan Masalah	14
3.1 Pemecahan Masalah	14
3.2 Proses Penyelesaian Solusi dengan Algoritma KMP dan BM	14
3.2.1 Proses Penyelesaian Solusi dengan Algoritma KMP	14
3.2.2 Proses Penyelesaian Solusi dengan Algoritma BM	15
3.3 Fitur	15
3.3.1 Fitur Pencarian Biodata dengan Algoritma KMP	15
3.3.2 Fitur Pencarian Biodata dengan Algoritma BM	16
3.3.3 Fitur Pencarian Non-Exact	16
3.4 Ilustrasi kasus	16
BAB 4	18
Implementasi dan Pengujian	18
4.1 Spesifikasi Teknis	18
4.1.1 Struktur Database	18
4.1.2 Fungsi dan Prosedur	18
4.2 Penggunaan Aplikasi	19
4.3 Hasil Pengujian	23
4.4 Analisis	28
BAB 5	30
Penutup	30
5.1 Kesimpulan	30
5.2 Saran	30
5.3 Tanggapan	30

5.4 Refleksi	31
Daftar Pustaka	32
Lampiran	33

## BAB 1

### Deskripsi Tugas



**Gambar 1.** Ilustrasi *fingerprint recognition* pada deteksi berbasis biometrik.

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

Di era digital ini, keamanan data dan akses menjadi semakin penting. Perkembangan teknologi membuka peluang untuk berbagai metode identifikasi yang canggih dan praktis. Beberapa metode umum yang sering digunakan seperti kata sandi atau pin, namun memiliki kelemahan seperti mudah terlupakan atau dicuri. Oleh karena itu, biometrik menjadi alternatif metode akses keamanan yang semakin populer. Salah satu teknologi biometrik yang banyak digunakan adalah identifikasi sidik jari. Sidik jari setiap orang memiliki pola yang unik dan tidak dapat ditiru, sehingga cocok untuk digunakan sebagai identitas individu.

Pattern matching merupakan teknik penting dalam sistem identifikasi sidik jari. Teknik ini digunakan untuk mencocokkan pola sidik jari yang ditangkap dengan pola sidik jari yang terdaftar di database. Algoritma pattern matching yang umum digunakan adalah Bozorth dan Boyer-Moore. Algoritma ini memungkinkan sistem untuk mengenali sidik jari dengan cepat dan akurat, bahkan jika sidik jari yang ditangkap tidak sempurna.

Dengan menggabungkan teknologi identifikasi sidik jari dan pattern matching, dimungkinkan untuk membangun sistem identifikasi biometrik yang aman, handal, dan mudah digunakan. Sistem ini dapat diaplikasikan di berbagai bidang, seperti kontrol akses, absensi karyawan, dan verifikasi identitas dalam transaksi keuangan.

Pada tugas ini, Anda diminta untuk mengimplementasikan sebuah program yang dapat melakukan identifikasi biometrik berbasis sidik jari. Proses implementasi dilakukan dengan menggunakan algoritma Boyer-Moore dan Knuth-Morris-Pratt, sesuai dengan yang diajarkan pada materi dan salindia kuliah.

Secara sekilas, penggunaan algoritma pattern matching dalam mencocokkan sidik jari terdiri atas tiga tahapan utama dengan skema sebagai berikut.



**Gambar 2.** Skema implementasi konversi citra sidik jari.

Sumber: Spesifikasi Tugas Besar 3

Gambar yang digunakan pada proses pattern matching kedua algoritma tersebut adalah gambar sidik jari penuh berukuran  $m \times n$  pixel yang diambil sebesar 30 pixel setiap kali proses pencocokan data. Untuk tugas ini, Anda dibebaskan untuk mengambil jumlah pixel asalkan didasarkan pada alasan yang masuk akal (dijelaskan pada laporan) dan penanganan kasus ujung yang baik (misal jika ternyata ukuran citra sidik jari tidak habis dibagi dengan ukuran pixel yang dipilih). Selanjutnya, data pixel tersebut akan dikonversi menjadi binary. Seperti yang mungkin Anda ketahui sesuai materi kuliah (ya kalau masuk kelas), karena binary hanya memiliki variasi karakter satu atau nol, maka proses pattern matching akan membuat proses pencocokan karakter menjadi lambat karena harus sering mengulangi proses pencocokan pattern. Cara yang dapat dilakukan untuk mengatasi hal tersebut adalah dengan mengelompokkan setiap baris kode biner per 8 bit sehingga membentuk karakter ASCII. Karakter ASCII 8-bit ini yang akan mewakili proses pencocokan dengan string data.

Untuk memberikan gambaran yang lebih jelas, berikut adalah contoh kasus citra sidik jari dan proses serta sampel hasil yang didapatkan. Dataset yang digunakan adalah dataset citra sidik jari yang terdapat pada bagian referensi.

Citra Sidik Jari	Potongan nilai binary	Potongan ASCII 8-bits
	10100000101000001010000 01010000010100000101000 00101000001010000010100 00010100000101000001010 00001010000010100000101 00000101000001010000010 10000010100000101000001 01000001010000010100000 10100000101000001010000 0101000001010	iÿÿÿÿÿÿÿÿÿÿù:Ãÿÿ<ÿÿuw ûÿkÿý,Pþÿ¿<ÿÿ¤ »ÿÿÐ%?ÿ&%íæÚ¤È/Zíÿÿq %ÿúTý~RûÛt íÿÿÿÿÿ iÿÿÿÿÿÿÿÿöÿÿ"bÿÿÁüþÿÊ¤ þ¤/ÿ¤2ÿúYÿÿ\ ^ÿÿþ fÿà !ÿÿÿþÿÿÿÿÿÿÿ <ðÿÖ ²ÿÿQûÿ Nÿá¤ÿÿÿÿÿ iÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ Þ>ÿÙþþ6Gÿÿ;Uÿÿÿfâÿ

**Gambar 3.** Hasil konversi citra sidik jari menjadi binary dan konversi ke ASCII 8-bits.

Sumber: Spesifikasi Tugas Besar 3

Pada tahap implementasi di titik ini, telah dihasilkan serangkaian karakter ASCII 8-bit yang merepresentasikan sebuah sidik jari. Hasil ini yang akan dijadikan dasar pencarian sidik jari yang sama dengan daftar sidik jari yang terdapat pada basis data. Pencarian sidik jari yang paling mirip dengan sidik jari yang menjadi masukan pengguna dilakukan dengan algoritma pencocokan *string* Knuth-Morris-Pratt (KMP) dan Boyer-Moore (BM). Jika tidak ada satupun sidik jari pada basis data yang *exact match* dengan sidik jari yang menjadi masukan melalui algoritma KMP ataupun BM, maka gunakan sidik jari paling mirip dengan kesamaan diatas nilai tertentu (*threshold*). Anda diberikan **kebebasan untuk**

**menentukan nilai batas persentase** kemiripan ini, silakan melakukan pengujian untuk menentukan nilai *tuning* yang tepat dan **jelaskan pada laporan**. Metode perhitungan tingkat kemiripan **jugadibebaskan** kepada Anda asalkan dijelaskan di laporan. Akan tetapi, asisten sangat menyarankan untuk menggunakan salah satu dari algoritma **Hamming Distance**, **Levenshtein Distance**, ataupun **Longest Common Subsequence (LCS)**.

Fungsi dari deteksi biometrik adalah mengidentifikasi seseorang, oleh sebab itu, pada proses implementasi program ini, sebuah citra sidik jari akan dicocokkan dengan biodata seseorang. Biodata yang dicocokkan terdiri atas data-data yang terdapat pada KTP, antara lain : NIK, nama, tempat/tanggal lahir, jenis kelamin, golongan darah, alamat, agama, status perkawinan, pekerjaan, dan kewarganegaraan. Relasi ini dibuat dalam sebuah basis data dengan skema detail seperti yang tertera pada bagian di bawah ini. Sebagai tambahan, struktur relasi basis data telah disediakan, silakan gunakan dump sql berikut. Atribut *berkas\_citra* yang disimpan pada tabel *sidik\_jari* adalah alamat dari citra dalam repositori (test/...), lokasi penyimpanan citra dalam folder **test**, bisa dilihat pada struktur repositori di bagian Pengumpulan Tugas.

'biodata'	
PK	'NIK' varchar(16) NOT NULL
	'nama' varchar(100) DEFAULT NULL
	'tempat_lahir' varchar(50) DEFAULT NULL
	'tanggal_lahir' date DEFAULT NULL
	'jenis_kelamin' enum('Laki-Laki','Perempuan') DEFAULT NULL
	'golongan_darah' varchar(5) DEFAULT NULL
	'alamat' varchar(255) DEFAULT NULL
	'agama' varchar(50) DEFAULT NULL
	'status_perkawinan' enum('Belum Menikah','Menikah','Cerai') DEFAULT NULL
	'pekerjaan' varchar(100) DEFAULT NULL
	'kewarganegaraan' varchar(50) DEFAULT NULL

'sidik_jari'	
	'berkas_citra' text
	'nama' varchar(100) DEFAULT NULL

**Gambar 4.** Skema basis data yang digunakan.

Sumber: Spesifikasi Tugas Besar 3

Seorang pribadi dapat memiliki lebih dari satu berkas citra sidik jari (**relasi one-to-many**). Akan tetapi, seperti yang dapat dilihat pada skema relasional di atas, keduanya tidak terhubung dengan sebuah relasi. Hal ini disebabkan karena pada kasus dunia nyata, data yang disimpan bisa saja mengalami korup. Dengan membuat atribut kolom yang mungkin korup adalah atribut *nama* pada tabel *biodata*, maka atribut *nama* pada tabel *sidik\_jari* **tidak dapat memiliki foreign-key** yang mereferensi ke tabel *biodata* (silakan *review* kembali materi IF2240 bagian basis data relasional). Pada tugas besar kali ini, kita akan coba melakukan simulasi implementasi data korup yang **hanya mungkin terjadi pada atribut nama di tabel biodata** (asumsikan kolom lain pada setiap tabel tidak mengalami korup). Akan tetapi, karena tujuan utama program adalah mengenali identitas seseorang secara lengkap hanya dengan menggunakan sidik jari, maka harus dilakukan sebuah skema untuk menangani data korup tersebut.

Sebuah data yang korup dapat memiliki berbagai macam bentuk. Pada tugas ini, jenis data korup adalah bahasa alay Indonesia. Terdengar lucu, tetapi para pendahulu kita sudah membuat bahasa ini

untuk berkomunikasi kepada sesamanya. Dengan mengutip dari [berbagai sumber](#), Anda akan diminta untuk menangani **kombinasi dari tiga buah variasi** bahasa alay, yaitu kombinasi huruf besar-kecil, penggunaan angka, dan penyingkatan. Contoh kasus bahasa alay dijelaskan dengan detail sebagai berikut.

Variasi	Hasil
Kata orisinil	Bintang Dwi Marthen
Kombinasi huruf besar-kecil	bintanG DwI mArthen
Penggunaan angka	B1nt4n6 Dw1 M4rthen
Penyingkatan	Bntng Dw Mrthen
Kombinasi ketiganya	b1ntN6 Dw mrthn

**Gambar 5.** Kombinasi ketiga variasi bahasa alay Indonesia.

Sumber: Spesifikasi Tugas Besar 3

Cara yang dapat digunakan untuk menangani ini adalah dengan menggunakan Regular Expression (Regex). Lakukan konversi pola karakter alay hingga dapat dikembalikan ke bentuk alfabetik yang bersesuaian. Setelah menggunakan Regex, Anda akan diminta kembali untuk melakukan *pattern matching* antara nama yang bersesuaian dengan algoritma KMP dan BM dengan ketentuan yang sama seperti saat [pencocokan sidik jari](#). Sebagai referensi bahasa alay, Anda dapat menggunakan *website* alay generator yang terdapat pada bagian referensi.

Setelah menemukan nama yang bersesuaian, Anda dapat menggunakan basis data untuk mengembalikan detail biodata lengkap individu. Jika seluruh prosedur diatas diimplementasikan dengan baik, maka sebuah sistem deteksi individu berbasis biometrik dengan sidik jari telah berhasil untuk diimplementasikan.

## BAB 2

### Landasan Teori

#### 2.1 Algoritma Knuth-Morris-Pratt

Algoritma Knuth-Morris-Pratt merupakan salah satu algoritma untuk melakukan pencocokan *string*. Algoritma ini memeriksa karakter dari kiri ke kanan seperti pada algoritma *brute force*. Meskipun memeriksa dari kiri ke kanan seperti algoritma *brute force*, algoritma ini berbeda dengan algoritma *brute force*. Algoritma Knuth-Morris-Pratt atau Algoritma KMP menggeser posisi karakter pencocokan dengan lebih sangkil.

Algoritma KMP memiliki cara agar pergeseran dilakukan dengan lebih sangkil dibandingkan dengan algoritma *brute force*. Sebelum melakukan pencocokan, algoritma ini melakukan *preprocessing* dengan menggunakan *border function*. *Border function* merupakan suatu fungsi yang akan menentukan sejauh apa pergeseran dari perbandingan karakter terjadi. Fungsi ini diterapkan pada *substring* yang akan dicari. Fungsi akan bernilai nol jika karakter yang diperiksa dari *string* adalah karakter yang pertama kali ditemukan dan akan memiliki nilai dari fungsi satu karakter sebelumnya ditambah dengan satu. Misalnya akan dicari kata “george” pada suatu *string*.

G	E	O	R	G	E

Pertama, periksa karakter dari indeks nol. Periksa apakah karakter tersebut sudah pernah muncul sebelumnya. Karena karakter tersebut adalah karakter pertama, maka masukkan angka nol pada indeks tersebut.

G	E	O	R	G	E
0					

Lalu, periksa karakter selanjutnya. Karakter “E” belum pernah muncul sebelumnya. Isi angka nol pada indeks tersebut. Lakukan langkah tersebut sampai pengecekan karakter yang sudah pernah muncul sebelumnya.

G	E	O	R	G	E
0	0	0	0		

Sampai pada indeks ke-4, karakter “G” sudah pernah muncul sebelumnya. Karena karakter tersebut sudah pernah muncul, isi nilai pada indeks tersebut dengan nilai dari indeks sebelumnya

dijumlahkan dengan satu. Indeks ke-3 bernilai nol dan karakter “G” sudah pernah muncul sebelumnya maka nilai indeks ke-4 diisi dengan satu.

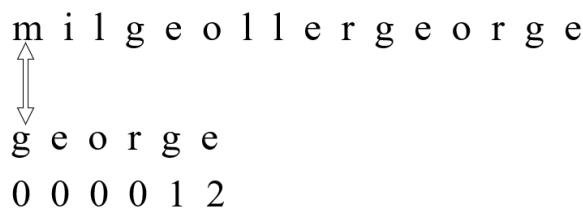
G	E	O	R	G	E
0	0	0	0	1	

Langkah pada karakter terakhir masih sama, jika karakter sudah pernah muncul tambah satu dari nilai karakter sebelumnya dan akan bernilai nol jika karakter belum pernah muncul. Karena karakter “E” sudah pernah muncul, maka indeks tersebut memiliki elemen bernilai dua.

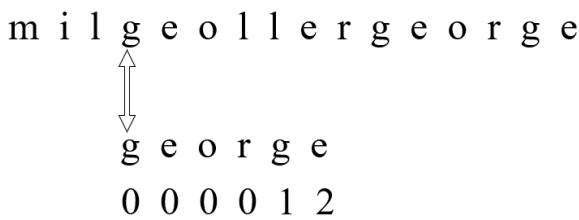
G	E	O	R	G	E
0	0	0	0	1	2

Setelah melakukan *preprocessing*, pencocokan *string* dilakukan. Pencocokan *string* dilakukan dengan menggunakan dua penunjuk menuju indeks dari *substring* yang akan dari *string* dan juga indeks menuju *string* itu sendiri. Pertama, karakter dari kedua penunjuk diperiksa. Jika karakternya sama, lanjut ke karakter berikutnya. Jika karakter dari setiap penunjuk berbeda, lihat nilai dari penunjuk *substring*. Mundur satu indeks dari *substring* dan periksa nilainya. Lakukan lagi pemeriksaan dari kedua karakter melalui kedua penunjuk. Jika penunjuk dari *substring* sudah berada pada indeks ke-0 dan memiliki karakter yang berbeda dengan karakter dari penunjuk *string*, indeks dari *string* dimajukan satu kali. Seluruh langkah ini diulangi sampai penunjuk *substring* berhasil mencocokkan sampai karakter terakhir dari *substring*.

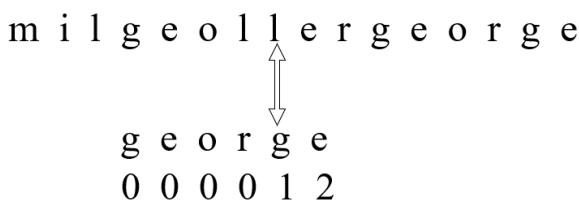
Berikut contoh pencarian menggunakan algoritma KMP, misalkan dicari kata “george” pada “milgeollergeorge”



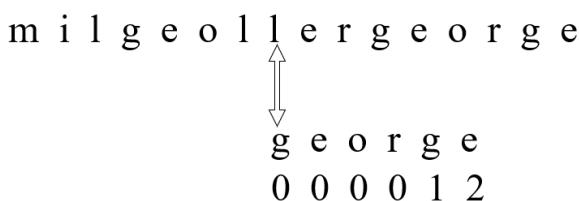
Karakter pertama diperiksa, karena tidak sama, karakter digeser sampai menemukan karakter yang sama.



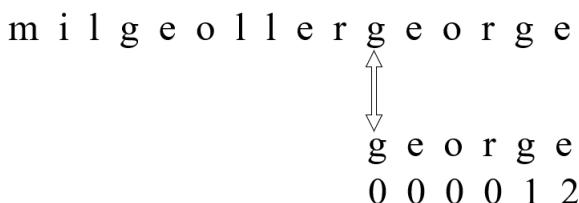
Penunjuk digeser setiap kali karakter yang diperiksa sama. Hal ini diulangi sampai penunjuk mencapai karakter terakhir dari *substring* sama antara *string* dengan *substring* atau penunjuk memiliki karakter yang berbeda.



Penunjuk memiliki karakter yang berbeda, maka lihat nilai dari *border function* satu karakter dari karakter *substring* yang ditunjuk. Satu karakter sebelum ‘‘g’’ adalah ‘‘r’’ dengan nilai nol. Maka penunjuk *substring* digeser ke karakter *substring* indeks ke-0. Ulangi langkah sampai menemukan satu karakter sama .



Periksa seperti pada langkah pertama. Penunjuk *string* digeser satu langkah kemudian diperiksa kembali. Karakter diperiksa satu per satu.



Jika sudah ditemukan semua sampai kedua penunjuk pada posisi sama dan penunjuk *substring* berada pada karakter terakhir, kembalikan indeks dari penunjuk *string* dikurangi dengan indeks dari penunjuk *substring*.

## **2.2 Algoritma BM**

Algoritma Boyer-Moore merupakan salah satu algoritma untuk melakukan string matching. Algoritma ini melakukan pemeriksaan secara maju dari belakang teks *pattern* kemudian pemeriksaan teks utama diperiksa maju. Pemeriksaan secara maju dan mundur inilah yang membuat pencocokan karakter menjadi lebih sangkil.

Sebelum melakukan pengecekan, dilakukan *preprocessing* terhadap *pattern* yang akan dicari. *Preprocessing* dilakukan dengan membuat larik dari *last occurrence* dari setiap karakter yang muncul dari teks yang dicari. Larik ini akan digunakan untuk membuat lompatan pemeriksaan dari karakter yang dicocokkan menjadi lebih sedikit.

Algoritma Boyer-Moore bekerja dengan melakukan pencocokan dari belakang teks *pattern* kemudian diperiksa apakah berbeda atau tidak. Jika karakter yang dicocokkan sama, pemeriksaan karakter dari *pattern* dan teks diperiksa secara maju. Hal ini dilakukan sampai pencocokan sudah sampai ke karakter pertama dari teks *pattern*. Jika sudah sampai ke karakter pertama dan tidak ada karakter yang berbeda, maka *pattern* sudah ditemukan pada teks. Jika berbeda, diperiksa apakah karakter yang diperiksa dari teks ditemukan pada teks *pattern*. Jika karakter tidak ditemukan dari teks *pattern*, maka teks *pattern* dimajukan ke karakter teks yang diperiksa. Sebaliknya, jika karakter ditemukan pada teks *pattern*, maka geser teks *pattern* menjadi sejajar dengan teks yang diperiksa, sejajar pada karakter yang sama dari pencocokan yang salah dengan karakter yang ada pada *pattern*. Pemeriksaan ini diulang sampai mencapai akhir dari teks utama atau *pattern* sudah ditemukan pada teks yang diperiksa.

## **2.3 Regex**

*Regular Expression* atau Regex merupakan sebuah sekvens karakter yang membuat spesifikasi dari pencocokan *pattern* pada suatu teks. Regex digunakan untuk melakukan suatu pencarian atau melakukan validasi input pada suatu teks string. Regex terdiri dari satu atau lebih karakter, metakarakter, *quantifier*, dan kelas karakter dalam membuat spesifikasi dari *string* yang akan diterima.

## **2.4 Penjelasan teknik pengukuran persentase kemiripan.**

### **2.4.1 Algoritma Fast Fourier Transform**

Algoritma *Fast Fourier Transform* atau FFT merupakan algoritma untuk mengkalkulasi *Discrete Fourier Transform* (DFT) dan *Inverse Discrete Fourier Transform* (IDFT) secara cepat dan sangkil. Algoritma ini dapat digunakan untuk menghitung perkalian dari dua buah polinomial dalam waktu  $O(n\log n)$ , yang merupakan perbaikan besar dari kompleksitas waktu algoritma naif  $O(n^2)$ .

Algoritma FFT dapat digunakan untuk menghitung kesamaan dari dua buah matriks *binary*. Secara formal, diberikan matriks  $T_A$  dengan banyak baris  $N_A$  dan banyak kolom  $M_A$  dan matriks  $T_B$  dengan banyak baris  $N_B$  dan banyak kolom  $M_B$ . Lebih lanjut, diberikan juga batasan  $N_A \geq N_B$  serta  $M_A \geq M_B$ . Perlu dicari suatu posisi  $k$ ,  $l$  sedemikian sehingga  $k < N_A$  dan  $l < M_A$  dan posisi  $k$ ,  $l$  merupakan posisi pada matriks  $T_A$  yang paling mirip dengan matriks  $T_B$ . Dengan kata

lain, jumlah posisi  $a, b$  yang memenuhi  $0 \leq a < N_B$ ,  $0 \leq b < M_B$  dan  $T_A[k + a, l + b] \neq T_B[a, b]$  minimum.

Persoalan di atas dapat diselesaikan dengan tahapan berikut.

1. Pertama-tama, tambahkan *padding* sebanyak  $M_A - M_B$  pada sebelah kanan matriks  $M_B$ .
2. Kemudian, buat 4 *array*,  $L_A, L'_A, L_B, L'_B$ .  $L_A$  akan berisi  $T_A$  yang di-*flatten*,  $L'_A$  akan berisi invers  $T_A$  yang di-*flatten*, sama halnya dengan  $L_B$  dan  $L'_B$ . Proses *flattening* merujuk pada proses konversi dari sebuah matriks 2D menjadi sebuah *array* 1D. Indeks  $a, b$  pada matriks akan menempati indeks  $aM+b$  pada *array*, dengan  $M$  merupakan banyak kolom.
3. Kemudian, *reverse* array  $L_B$  dan  $L'_B$ . Hal ini artinya piksel  $a, b$  pada  $T_A$  akan menempati posisi  $aM_A+b$  pada  $L_A$ , dan piksel  $a, b$  pada  $T_B$  akan menempati posisi  $N_AM_B-aM_A-b-1$  pada  $L_B$ .
4. Kemudian, lakukan kalkulasi polinomial  $L_{AB} = FFT(L_A, L_B) + FFT(L'_A, L'_B)$ . Perhatikan bahwa jika  $T_B$  diletakkan di indeks  $k, l$  pada  $T_A$ , maka jumlah indeks yang  $a, b$  yang memenuhi  $0 \leq a < N_B$ ,  $0 \leq b < M_B$  dan  $T_A[k + a, l + b] = T_B[a, b]$  terkandung di dalam  $L_{AB}[kN_A+l+N_A+M_B-1]$ .
5. Untuk menemukan indeks yang paling mirip, hanya perlu dicari indeks  $k, l$  yang memiliki  $L_{AB}[kN_A+l+N_A+M_B-1]$  maksimum.

#### **2.4.2 Density-based Clustering Algorithm**

*Density-based Clustering Algorithm* merupakan sebuah algoritma *data clustering* yang dikemukakan Martin Ester, Hans-Peter Kriegel, Jörg Sander and Xiaowei Xu pada tahun 1996. Diberikan himpunan titik, algoritma ini akan mengelompokkan titik-titik yang berdekatan dan menandai titik-titik outlier yang terletak sendirian di wilayah dengan kepadatan rendah. Algoritma ini melakukan proses dengan bantuan parameter  $\epsilon$ , yang menandakan jarak maksimum dari suatu titik dalam suatu *cluster* ke titik lain.

Algoritma ini memiliki tahapan sebagai berikut:

1. Diberikan himpunan titik  $H$ , parameter  $\epsilon$  yaitu jarak maksimum suatu titik ke titik terdekat dalam sebuah *cluster*, dan parameter *minpts* yang menandakan banyak titik minimum dalam suatu *cluster*.
2. Dari himpunan titik  $H$ , pilih sembarang poin  $P$ .
3. Cari seluruh titik dalam himpunan  $H$  yang berjarak kurang dari  $\epsilon$  dengan  $P$ . Jika jumlah titik yang ditemukan kurang dari *minpts*, ulangi ke langkah 2. Jika jumlah titik yang ditemukan lebih dari atau sama dengan *minpts*, lanjut ke tahap 4.
4. Label titik  $P$  dengan penanda area. Lakukan ekspansi pada titik-titik tetangga, yaitu melakukan langkah 3 pada setiap titik yang ditemukan secara rekursif.
5. Ulangi langkah 2 hingga seluruh titik pada himpunan  $H$  telah dikelompokkan ke dalam sebuah *cluster*.

#### **2.4.3 Convex Hull**

Algoritma Convex Hull digunakan untuk mencari *convex hull* dari sekumpulan titik dalam geometri komputasi. Sebuah *convex hull* adalah himpunan titik yang membentuk bentuk

konveks terkecil yang melingkupi semua titik. Pencarian *convex hull* dapat dilakukan dengan Graham Scan, dengan tahapan sebagai berikut:

1. Diberikan himpunan titik X, akan dicari *convex hull* dari himpunan titik.
2. Misal  $x_0$  adalah titik paling kiri dalam himpunan X. Urutkan titik dalam X berdasarkan derajat perbedaan dari  $x_0$ .
3. Buat sebuah *stack*, dan kemudian masukkan  $x_0$  dan  $x_1$ . Deklarasikan sebuah integer  $i = 2$ .
4. Jika  $x_i$  membuat sebuah *left turn* relatif terhadap 2 elemen pertama dalam *stack*, masukkan  $x_i$  ke dalam *stack*, dan tambahkan  $i$ . Jika tidak, pop elemen pertama dalam *stack*.
5. Ulangi langkah 4 hingga semua titik telah di-*consider*.

#### **2.4.4 Tahapan Pengukuran Persentase Kemiripan**

Akan digunakan 3 algoritma yang dijelaskan sebelumnya dalam mengukur persentase kemiripan. Tahapan yang dilakukan adalah sebagai berikut:

1. Diberikan gambar  $G_A$  dan gambar  $G_B$ , deklarasikan integer  $k, l$  yang menandakan posisi indeks dalam  $G_A$  yang paling mirip dengan  $G_B$ . Jika ukuran  $G_A$  dan  $G_B$  sama, maka pastilah  $k = 0, l = 0$ . Jika tidak, lakukan FFT untuk mencari  $k$  dan  $l$ .
2. Cari dan kumpulkan piksel yang berbeda, kemudian lakukan DBSCAN (*Data-clustering based algorithm*) untuk mengelompokkan piksel-piksel yang berbeda ke dalam area yang sama.
3. Untuk kumpulan area yang didapatkan, cari convex hull untuk tiap area dan hitung luasnya. Hitung jumlah luas, *similarity* dihitung dari luas gambar  $G_B$  dikurangi jumlah luas dari area dibagi dengan luas gambar  $G_B$ .

### **2.5 Penjelasan Enkripsi Basis Data**

Tiny Encryption Algorithm (TEA) adalah algoritma enkripsi simetris yang sederhana dan efisien, dirancang oleh David Wheeler dan Roger Needham dari Cambridge Computer Laboratory pada tahun 1994. TEA bekerja sebagai blok cipher, mengenkripsi data dalam blok berukuran 64 bit (8 byte) dan menggunakan kunci enkripsi sepanjang 128 bit (16 byte). Proses enkripsi dan dekripsi TEA melibatkan 32 siklus operasi sederhana seperti penjumlahan, pengurangan, XOR, dan pergeseran bit, membuatnya mudah diimplementasikan dan efisien dalam hal kecepatan.

TEA memulai proses enkripsinya dengan membagi data menjadi dua bagian 32 bit,  $v0$  dan  $v1$ , serta membagi kunci menjadi empat bagian 32 bit,  $k0, k1, k2$ , dan  $k3$ . Pada setiap putaran, nilai  $v0$  dan  $v1$  diperbarui melalui kombinasi operasi bit menggunakan bagian-bagian kunci dan nilai tetap Delta ( $0x9e3779b9$ ). Dekripsi dilakukan dengan cara yang sama, tetapi urutan operasinya dibalik. Untuk menangani data yang panjangnya tidak merupakan kelipatan 8 byte, TEA menggunakan padding, menambahkan byte tambahan ke akhir data sehingga panjang total data menjadi kelipatan 8 byte. Padding ini kemudian dihapus setelah proses dekripsi untuk mengembalikan data ke panjang aslinya.

Meskipun TEA cukup kuat untuk banyak aplikasi, kesederhanaannya membuatnya rentan terhadap beberapa jenis serangan, seperti analisis diferensial dan serangan kunci terkait. Kelemahan ini

membuatnya kurang cocok untuk aplikasi yang memerlukan keamanan sangat tinggi. Namun, karena kesederhanaan dan efisiensinya, TEA tetap menjadi pilihan yang populer untuk banyak aplikasi enkripsi. Implementasi TEA dalam kode menunjukkan bagaimana data dan kunci diproses dan dioperasikan untuk mencapai enkripsi dan dekripsi, memastikan data tetap terlindungi selama transmisi atau penyimpanan.

## 2.6 Aplikasi desktop

Aplikasi yang dikembangkan merupakan sebuah aplikasi yang berfungsi untuk melakukan pencarian gambar sidik jari yang ada pada *database* dengan masukkan berupa gambar sidik jari. Aplikasi ini menyelesaikan persoalan tersebut dengan menggunakan Algoritma KMP ataupun dengan menggunakan Algoritma BM. Pengguna diberikan pilihan untuk menggunakan salah satu dari algoritma tersebut untuk melakukan pencarian. Jika tidak ditemukan *exact match*, program akan menampilkan hasil lain yang memiliki kemiripan tertinggi dengan nilai kemiripan di atas batas minimum. Aplikasi ini mendapatkan data dari *database* SQL dan kumpulan gambar yang berada dalam suatu folder.

Aplikasi yang dikembangkan merupakan sebuah aplikasi WPF berbasis .NET dengan menggunakan bahasa C#. Aplikasi ini dikembangkan dengan menggunakan kakas Visual Studio 2022. Behavior atau perilaku dari aplikasi ini diatur dengan menggunakan C# dan untuk *Graphical User Interface* (GUI) dari aplikasi ini dibuat dengan menggunakan XAML. Aplikasi ini menghubungkan C# dengan database melalui *package* tambahan SQLite.

## BAB 3

### Analisis Pemecahan Masalah

#### 3.1 Pemecahan Masalah

Untuk mengecek apakah gambar sidik jari A terdapat di dalam gambar sidik jari pada *database*, diperlukan suatu representasi dari kedua gambar yang dapat dibandingkan dengan mangkus dan sangkil. Pengecekan setiap *pixel* satu per satu akan memerlukan daya komputasi yang cukup besar. Maka dari itu, langkah pertama dari masalah ini adalah membuat representasi *binary string* dari gambar A dan gambar B. Representasi *binary* dari sidik jari adalah menggunakan *ridge* dan *valley* dari gambar yang diberikan. Representasi ini dipilih karena tingkat akurasinya yang jauh lebih tinggi dibandingkan dengan representasi 8-bit ASCII. Meskipun penggunaan representasi ASCII dapat membuat pemrosesan menjadi jauh lebih cepat, perubahan satu *pixel* pada gambar akan membuat *mismatch* delapan *pixel* pada saat pengecekan karena satu karakter direpresentasikan dari delapan bit *binary string*.

Setelah representasi *binary* dari gambar didapatkan, dilakukan pencocokan satu per satu sesuai dengan algoritma yang dipilih. Gambar dari masukan diperiksa dengan seluruh gambar yang ada pada *database*. Jika ditemukan *exact match*, maka akan dikeluarkan gambar dan biodata dari gambar sidik jari yang bersesuaian. Jika tidak ditemukan *exact match*, maka dilakukan pemeriksaan ulang satu per satu dari seluruh gambar yang ada pada *database* dengan menggunakan algoritma *Convex Hull*. Algoritma ini akan memeriksa tingkat kemiripan dari gambar yang ada pada *database* dengan gambar dari masukkan. Jika seluruh gambar sudah diperiksa, akan dikembalikan gambar dengan persentase kemiripan tertinggi yang memiliki persentase kemiripan di atas batas minimum kemiripan.

#### 3.2 Proses Penyelesaian Solusi dengan Algoritma KMP dan BM

##### 3.2.1 Proses Penyelesaian Solusi dengan Algoritma KMP

Proses penyelesaian dilakukan dengan membuat representasi *binary* dari gambar masukan dan gambar yang akan dibandingkan dari *database*. Setelah dibuat representasi *binary*, gambar masukan akan dibuat sebagai *pattern* dan gambar yang ada pada database akan dianggap sebagai “teks” yang akan dicari. Lalu, dibuat sebuah *border function* untuk representasi *binary* dari gambar yang akan dicari. Setelah *border function* dibuat, *pattern* dicocokkan dengan gambar yang dicari.

Pencocokan dimulai dari awal *pattern* dengan awal dari representasi *binary* yang dicari pada database. Karakter dari gambar yang dicari dicocokkan satu per satu dengan yang ada pada *pattern*. Jika terdapat ketidakcocokan karakter dari gambar, periksa *border function* yang telah dibuat. *Pattern* digeser sesuai dengan *border function* yang telah dibuat. Jika ditemukan bagian dari gambar yang seluruh karakternya sama pada *pattern*, dikembalikan gambar dan nama pemilik sidik jari yang bersesuaian. Jika tidak ada, pencarian ulang dilakukan dengan menggunakan algoritma *Convex Hull*. Pemeriksaan karakter dilakukan dengan maju sehingga proses pencarian gambar yang bersesuaian dapat dilakukan dengan kompleksitas waktu linear.

Proses *string matching* dengan algoritma KMP akan mengambil  $\frac{1}{2}$  baris bagian tengah dari gambar asli, yang dicocokkan dengan satu baris tengah dari gambar kueri. Jika ternyata *string matching* berhasil, program akan mencocokkan kedua gambar secara *brute force*. Proses ini memiliki efek positif dimana gambar yang di-*crop* dari gambar aslinya dapat ditemukan dengan cepat. Algoritma hanya akan mengambil setengah bagian gambar supaya waktu yang digunakan program tidak begitu lama.

### 3.2.2 Proses Penyelesaian Solusi dengan Algoritma BM

Sama seperti pada penyelesaian dengan algoritma KMP, proses penyelesaian dilakukan dengan membuat representasi *binary* dari gambar masukan dan gambar yang akan dibandingkan dari *database*. Setelah dibuat representasi *binary*, gambar masukkan akan dibuat sebagai *pattern* dan gambar yang ada pada database akan dianggap sebagai “teks” yang akan dicari. Berbeda dengan algoritma KMP yang menggunakan *border function*, algoritma BM menggunakan kemunculan terakhir dari setiap karakter yang ada pada representasi *binary* dari gambar.

Pencocokan dimulai dari bagian awal gambar dengan pemeriksaan karakter dimulai dari akhir *pattern*. Jika karakter sama, pemeriksaan karakter dimajukan sampai karakter terdepan dari *pattern* atau terdapat *mismatch* karakter. Jika terjadi *mismatch* karakter, karakter dari gambar diperiksa apakah terdapat karakter tersebut pada *pattern* di posisi lain. Jika ada, *pattern* digeser sejajar sesuai dengan *mismatch* tersebut dan jika tidak ada *pattern* digeser melewati karakter yang *mismatch* tersebut. Pengecekan dengan menggunakan algoritma ini dapat lebih lama karena penggunaan representasi karakter *binary* yang merupakan salah satu kasus terburuk untuk algoritma ini. Jika *pattern* ditemukan, dikembalikan gambar dan nama dari pemilik sidik jari. Jika tidak, dilakukan pencarian ulang dengan algoritma *Convex Hull*.

Proses *string matching* dengan algoritma BM hanya akan mengambil 5% baris bagian tengah dari gambar asli, yang dicocokkan dengan satu baris tengah dari gambar kueri. Jika ternyata *string matching* berhasil, program akan mencocokkan kedua gambar secara *brute force*. Proses ini memiliki efek positif dimana gambar yang di-*crop* dari gambar aslinya dapat ditemukan dengan cepat. Konsiderasi pemilihan baris yang sangat sedikit berdasar dari kompleksitas waktu terburuk algoritma BM yang bersifat kuadratik. Dikhawatirkan dengan memilih baris yang banyak program akan memakan waktu yang lama.

## 3.3 Fitur

### 3.3.1 Fitur Pencarian Biodata dengan Algoritma KMP

Pada fitur ini, pengguna dapat mencari biodata yang cocok dengan gambar dari input yang diberikan oleh pengguna. Pengguna diminta untuk memilih gambar terlebih dahulu, kemudian memilih algoritma. Setelah gambar dan algoritma KMP dipilih, pengguna dapat menekan tombol search untuk mencari biodata yang bersesuaian. Status loading akan muncul pada pojok kiri bawah untuk menunjukkan bahwa program sedang mencari biodata yang bersesuaian. Program akan menampilkan hasil pada bagian kanan *user interface*.

### **3.3.2 Fitur Pencarian Biodata dengan Algoritma BM**

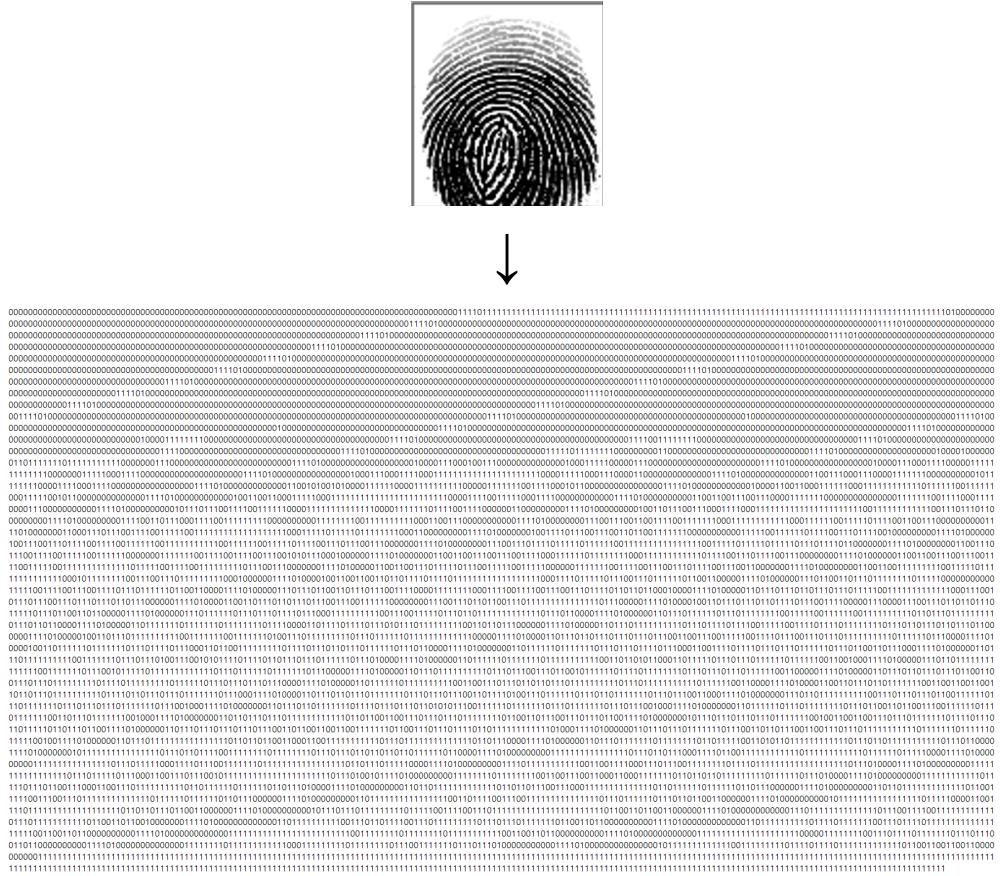
Pada fitur ini, pengguna dapat mencari biodata yang cocok dengan gambar dari input yang diberikan oleh pengguna. Pengguna diminta untuk memilih gambar terlebih dahulu, kemudian memilih algoritma. Setelah gambar dan algoritma BM dipilih, pengguna dapat menekan tombol search untuk mencari biodata yang bersesuaian. Status loading akan muncul pada pojok kiri bawah untuk menunjukkan bahwa program sedang mencari biodata yang bersesuaian. Program akan menampilkan hasil pada bagian kanan *user interface*.

### **3.3.3 Fitur Pencarian Non-Exact**

Fitur ini merupakan fitur untuk melakukan pencarian dengan gambar yang memiliki perbedaan dengan gambar asli pada *database*. Fitur ini tidak dipilih secara manual, akan tetapi dijalankan secara otomatis jika tidak ditemukan *exact match* pada *database*. Jika ditemukan hasil yang memiliki persentase kemiripan di atas batas minimum, akan ditampilkan biodata seperti pada fitur pencarian dengan algoritma KMP ataupun BM, tetapi dengan sedikit perbedaan. Bagian dari gambar sidik jari yang berbeda dengan masukkan pengguna akan ditandai dan kemudian akan ditampilkan juga tingkat kemiripannya. Jika tidak ada hasil yang memiliki persentase yang lebih dari batas minimum, tidak ada biodata dan gambar sidik jari yang akan ditampilkan.

## **3.4 Ilustrasi kasus**

Misalkan terdapat seorang pengguna yang ingin mencari data dari suatu gambar sidik jari dari *database*. Pengguna tersebut mengunggah gambar sidik jari pada aplikasi dan kemudian memilih algoritma yang akan digunakan. Gambar tersebut akan diubah menjadi suatu representasi *binary*, yaitu sebuah *string* dengan nilai 1 atau 0. Pada implementasi ini, pengubahan dari gambar menjadi *binary string* tidak dengan mengubah seluruh bagian dari gambar menjadi sebuah teks yang panjang, akan tetapi memecah gambar sidik jari menjadi *list of binary string*. Pemecahan ini dilakukan agar pemotongan dari gambar tidak akan membuat algoritma KMP atau BM langsung gagal karena menganggap satu gambar sebagai satu baris yang sama.



**Gambar 6.** Ilustrasi contoh pengubahan gambar sidik jari menjadi *binary string*

Setelah gambar diubah menjadi *list of binary string*, dilakukan pencocokan sesuai dengan algoritma yang dipilih. Jika dengan menggunakan algoritma KMP, dibentuk suatu *border function* dari *string pattern* yang akan dicari. Jika menggunakan algoritma BM, dicari kemunculan terakhir dari karakter 0 dan 1 pada *pattern*. Setelah *preprocessing* dilakukan, dilakukan pengecekan sesuai dengan algoritma yang dipilih. Sebagai contoh sederhana, misalkan terdapat *string* hasil konversi terdiri dari beberapa karakter 0 dan 1. Dilakukan pemeriksaan dari gambar sidik jari pada *database* yang sudah dalam bentuk *binary string*. Setiap karakter diperiksa satu per satu sesuai dengan algoritma KMP atau BM. Jika pemeriksaan dari setiap baris yang diperiksa ditemukan, maka akan dikembalikan nama dan gambar hasil pemeriksaan. Jika tidak, pencarian dilanjutkan dengan menggunakan algoritma *convex hull*.

## BAB 4

### Implementasi dan Pengujian

#### 4.1 Spesifikasi Teknis

##### 4.1.1 Struktur Database

Terdapat dua tabel yang digunakan dalam *database* pada program. *Database* pertama adalah *database* sidik\_jari untuk menyimpan data gambar sidik jari dan nama, dan *database* kedua adalah *database* biodata untuk menyimpan biodata dari nama-nama yang ada pada *database* sidik jari. Berikut adalah implementasi skema dari *database* yang digunakan.

Nama Tabel	Query Tabel
biodata	<pre>CREATE TABLE `biodata` (   `NIK` TEXT NOT NULL,   `nama` TEXT DEFAULT NULL,   `tempat_lahir` TEXT DEFAULT NULL,   `tanggal_lahir` TEXT DEFAULT NULL,   `jenis_kelamin` TEXT,   `golongan_darah` TEXT DEFAULT NULL,   `alamat` TEXT DEFAULT NULL,   `agama` TEXT DEFAULT NULL,   `status_perkawinan` TEXT,   `pekerjaan` TEXT DEFAULT NULL,   `kewarganegaraan` TEXT DEFAULT NULL,   PRIMARY KEY (`NIK`) );</pre>
sidik_jari	<pre>CREATE TABLE `sidik_jari` (   `berkas_citra` text,   `nama` TEXT DEFAULT NULL );</pre>

##### 4.1.2 Fungsi dan Prosedur

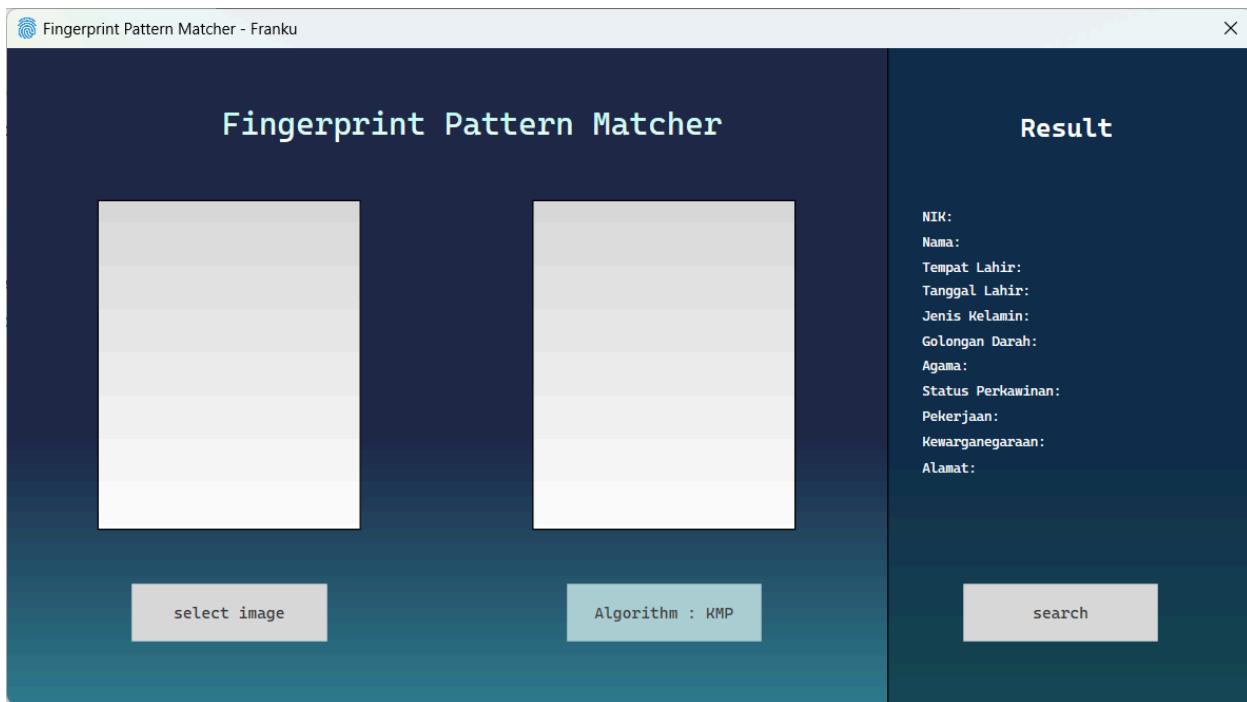
**Tabel 1.** Fungsi dan prosedur serta keterangannya yang digunakan dalam program.

Nomor	Fungsi/Prosedur	Keterangan
01	bool BMSearch(string pat, string txt)	Fungsi untuk melakukan pengecekan <i>pattern</i> dengan menggunakan algoritma BM

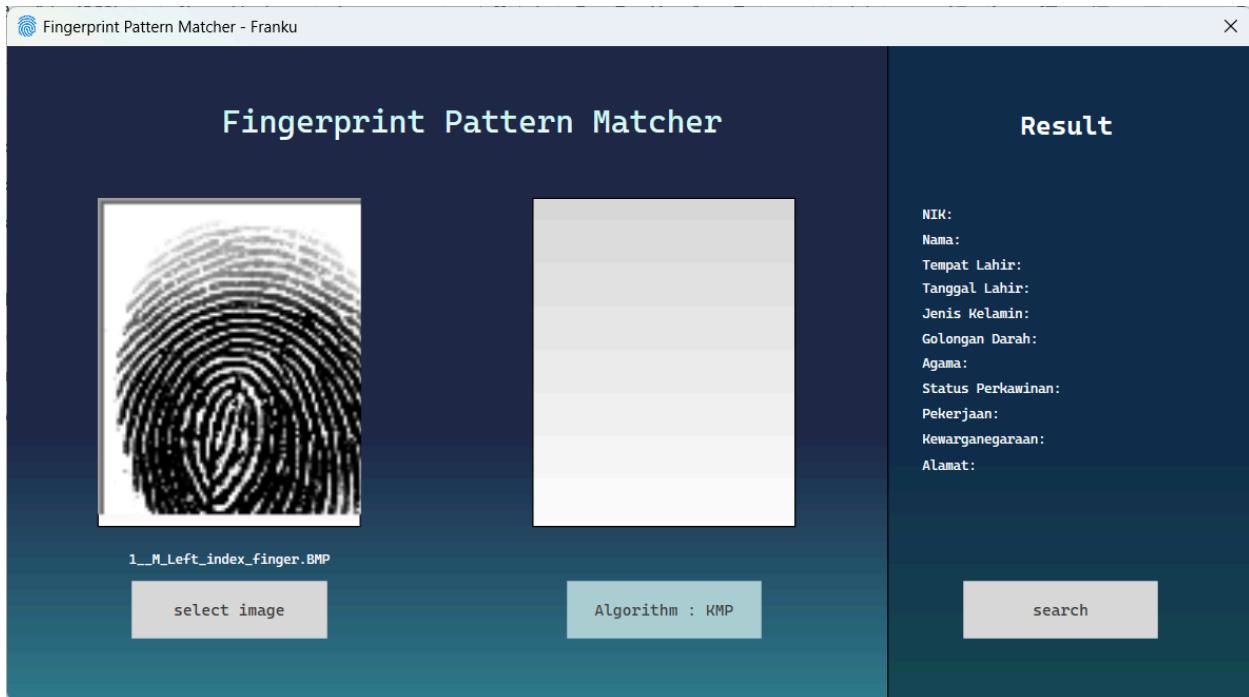
02	List<int> computeLPSArray(string pat, int M)	Fungsi untuk membentuk LPS/ <i>border function</i> pada algoritma KMP
03	bool KMPSearch(string pat, string txt)	Fungsi untuk melakukan pengecekan dengan menggunakan algoritma KMP
04	bool regexMatch(string plainText, string comparedText)	Fungsi untuk melakukan pengecekan dengan Regex yang disesuaikan dengan bahasa “alay”
05	BitmapImage Bitmap2BitmapImage(Bitmap bitmap)	Fungsi untuk mengubah data pada variabel bertipe data Bitmap menjadi BitmapImage
06	List<string> BmpToBinaryString(Bitmap bmp)	Fungsi yang mengubah data pada variabel bertipe Bitmap menjadi <i>List of string</i>
07	Bitmap BitmapImage2Bitmap(BitmapI mage bitmapImage)	Fungsi untuk mengubah data pada variabel bertipe data BitmapImage menjadi Bitmap
08	List<long> mul(ref List<long> a, ref List<long> b)	Fungsi untuk menghitung perkalian dari dua buah polinomial a dan b dengan metode FFT. Kedua polinomial dalam bentuk koefisien.
09	List<int> Cluster(List<Point> data)	Fungsi untuk mengelompokkan seluruh titik dalam <i>list</i> data menggunakan algoritma <i>data-based clustering</i> .
10	List<Point> GetConvexHull(List<Point> points)	Fungsi untuk mengembalikan titik-titik yang menjadikan sebuah <i>convex hull</i> dari titik pada <i>list</i> points.
11	byte[] Encrypt(byte[] data, byte[] key)	Fungsi untuk mengenkripsi.
12	byte[] Decrypt(byte[] data, byte[] key)	Fungsi untuk mendekripsi.

## 4.2 Penggunaan Aplikasi

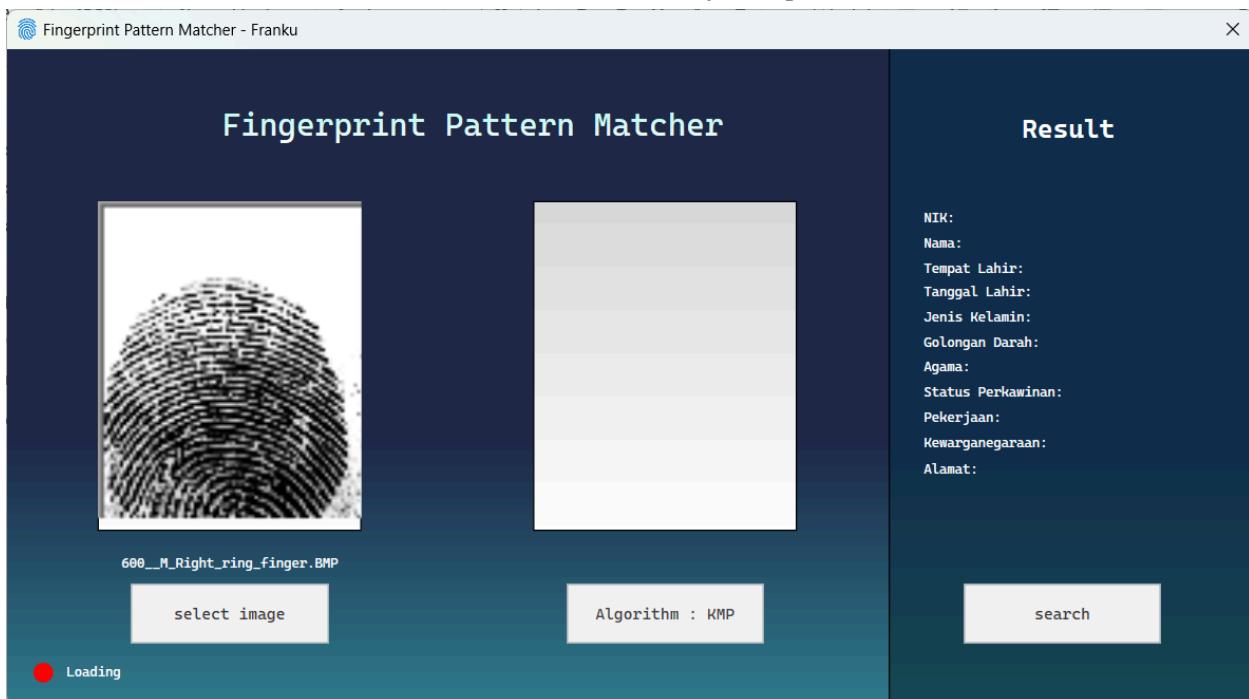
Pertama, pengguna akan diminta untuk memilih gambar terlebih dahulu. Untuk memilih gambar, tekan tombol “select image” yang ada pada bagian kiri program. Setelah ditekan, program akan memunculkan *pop-up* dialog untuk memilih gambar. Setelah gambar dipilih, program akan kembali ke tampilan sebelumnya. Setelah gambar dipilih, pilih algoritma yang ingin digunakan. Satu tombol yang sama dapat digunakan sebagai *toggle* untuk berganti algoritma dari KM ke BM ataupun sebaliknya. Setelah algoritma dipilih, tekan tombol search untuk mencari biodata dan hasil gambar yang bersesuaian. Pada bagian pojok kiri bawah akan muncul status *loading* untuk menandakan program sedang mencari gambar sidik jari yang bersesuaian. Penanda akan hilang setelah proses pencarian selesai. Setelah pencarian selesai akan muncul biodata dan gambar sidik jari pencarian jika gambar yang sama atau yang memiliki kemiripan di atas batas minimum ditemukan. Jika tidak, akan muncul *pop-up* bahwa hasil tidak ditemukan.



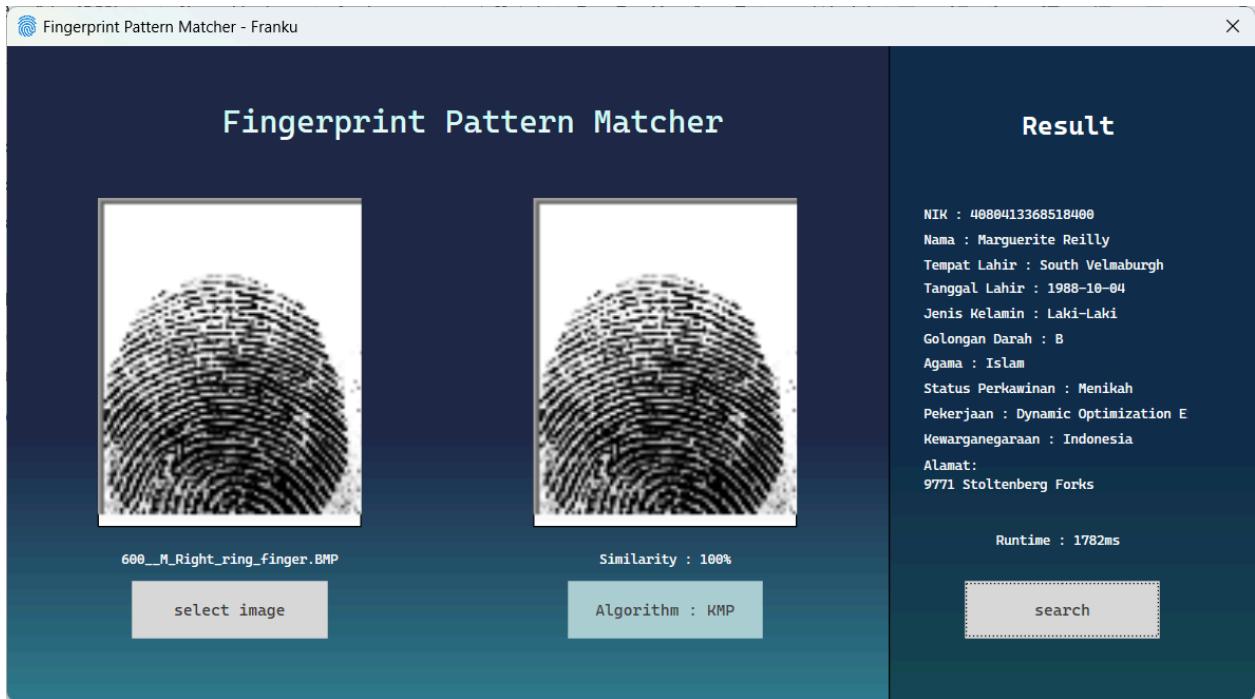
Gambar 7. Tampilan utama program



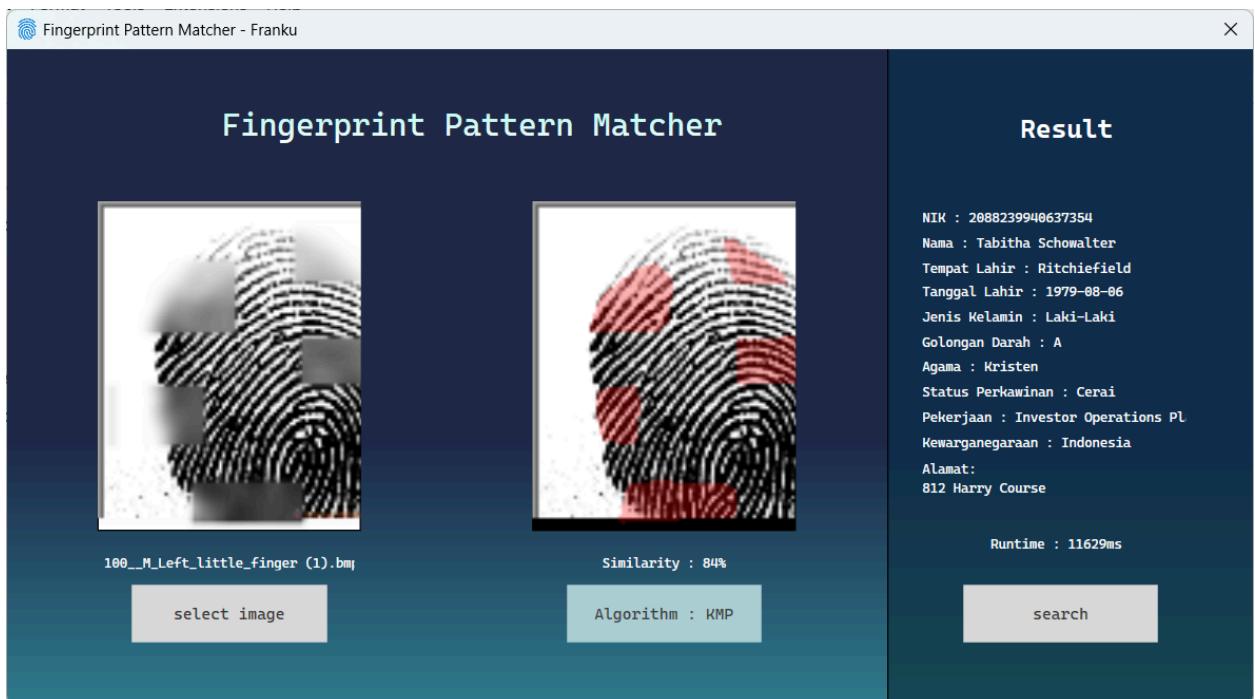
Gambar 8. Gambar sidik jari terpilih



Gambar 9. Program dalam proses pencarian

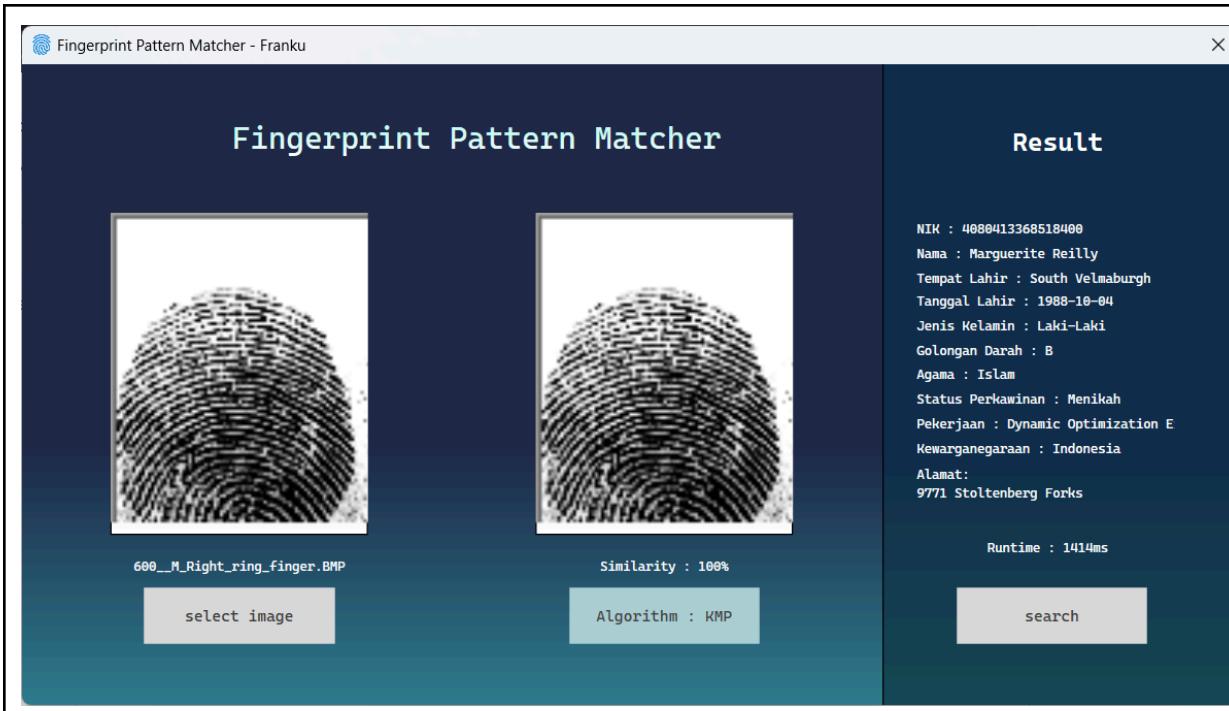


Gambar 10. Hasil untuk *exact match*

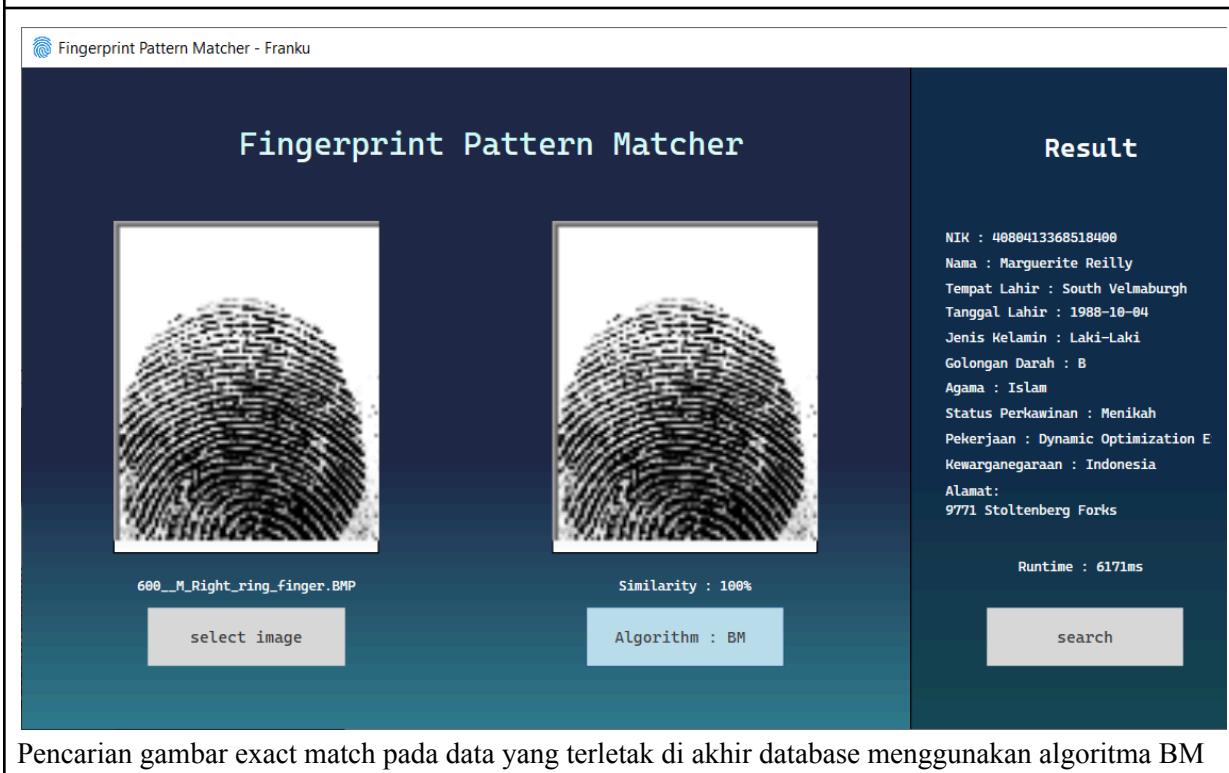


Gambar 11. Hasil untuk *non-exact match*

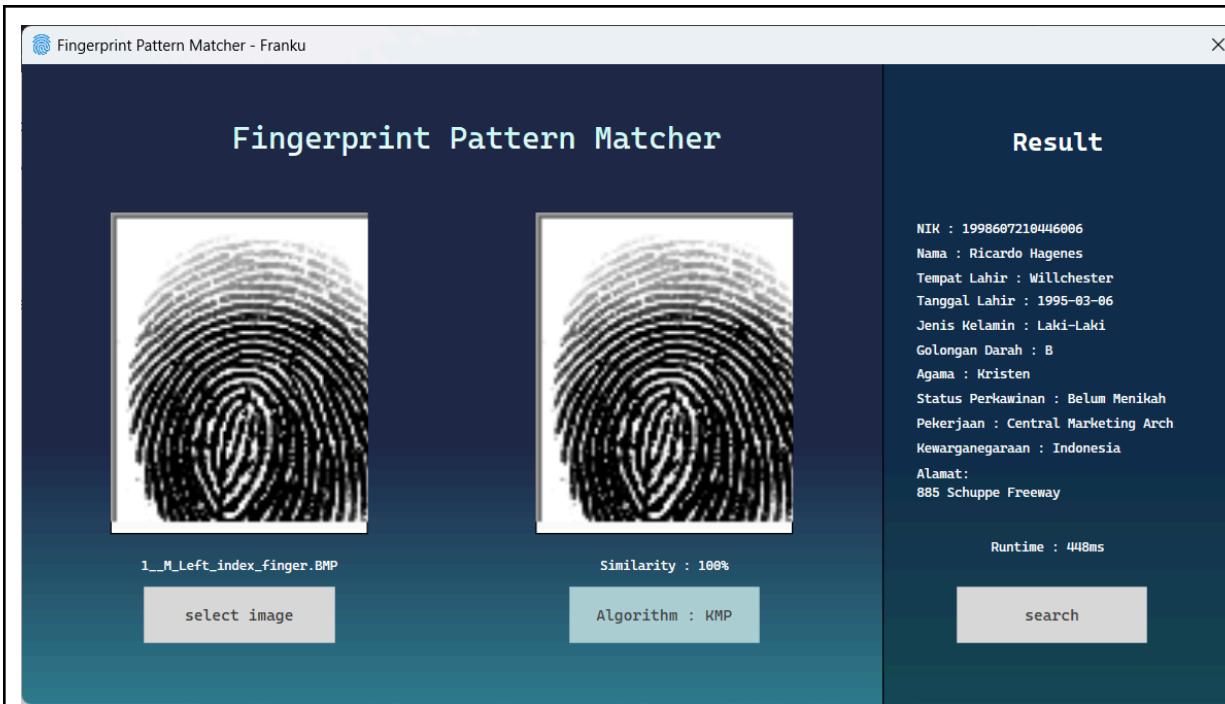
### 4.3 Hasil Pengujian



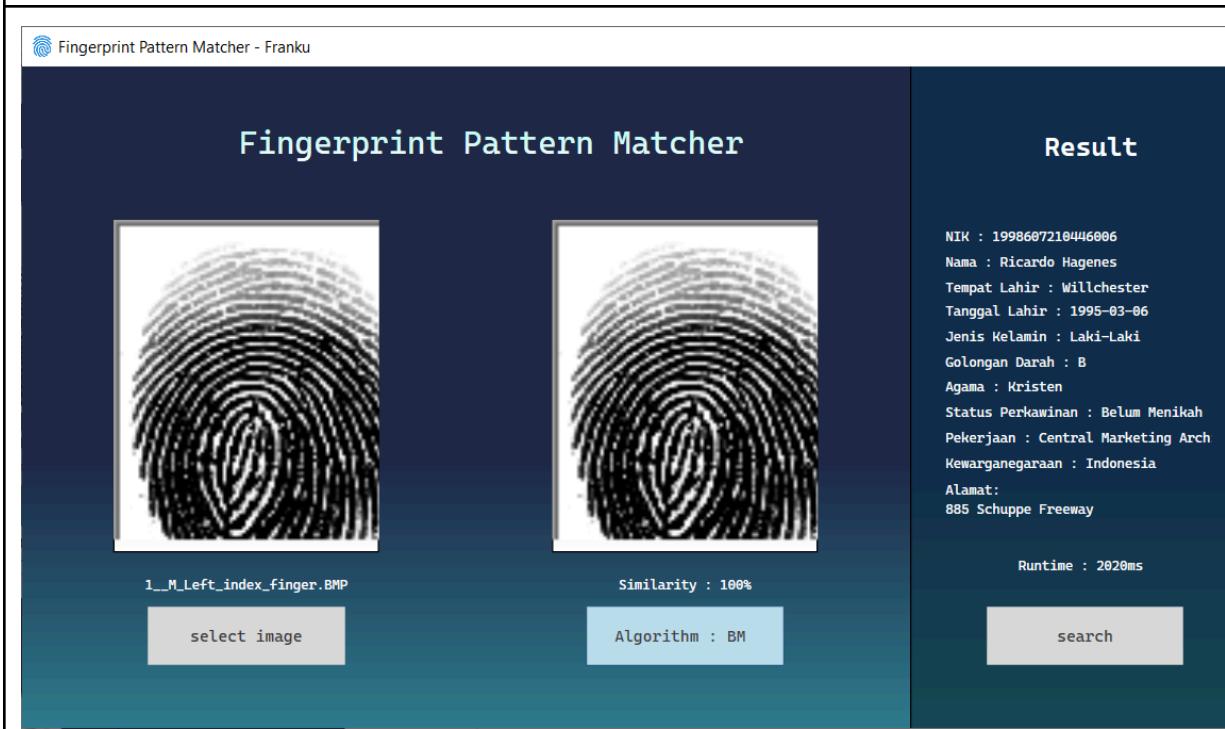
Pencarian gambar exact match pada data yang terletak di akhir database menggunakan algoritma KMP



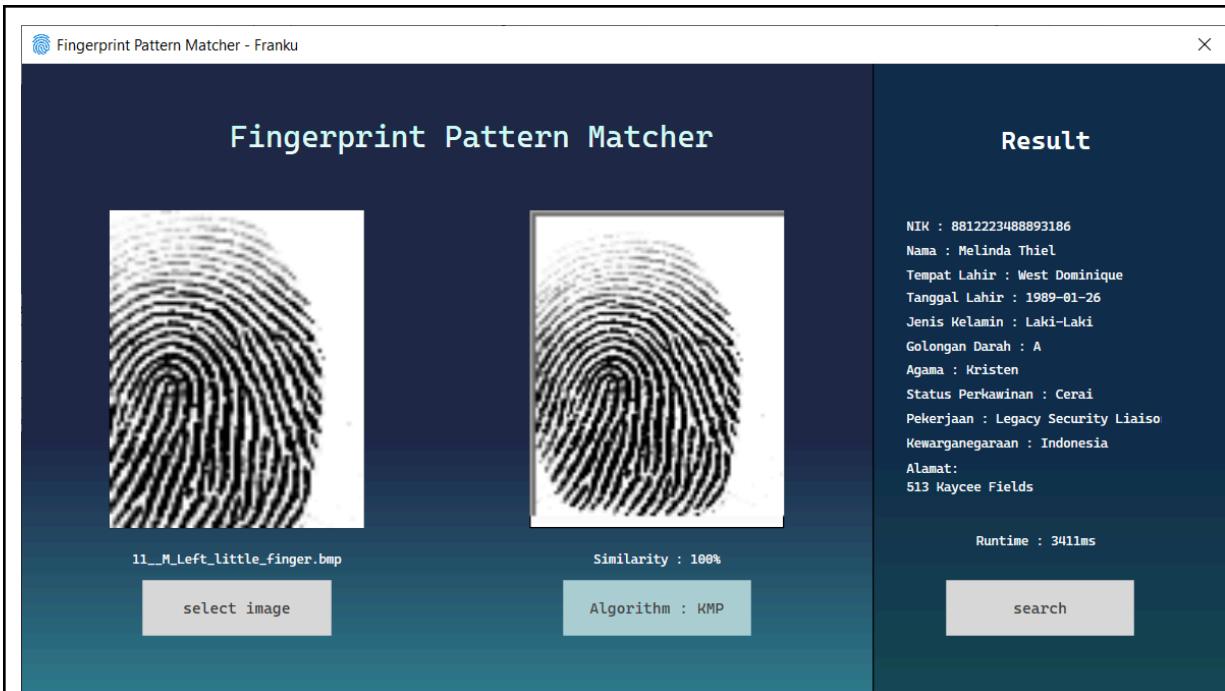
Pencarian gambar exact match pada data yang terletak di akhir database menggunakan algoritma BM



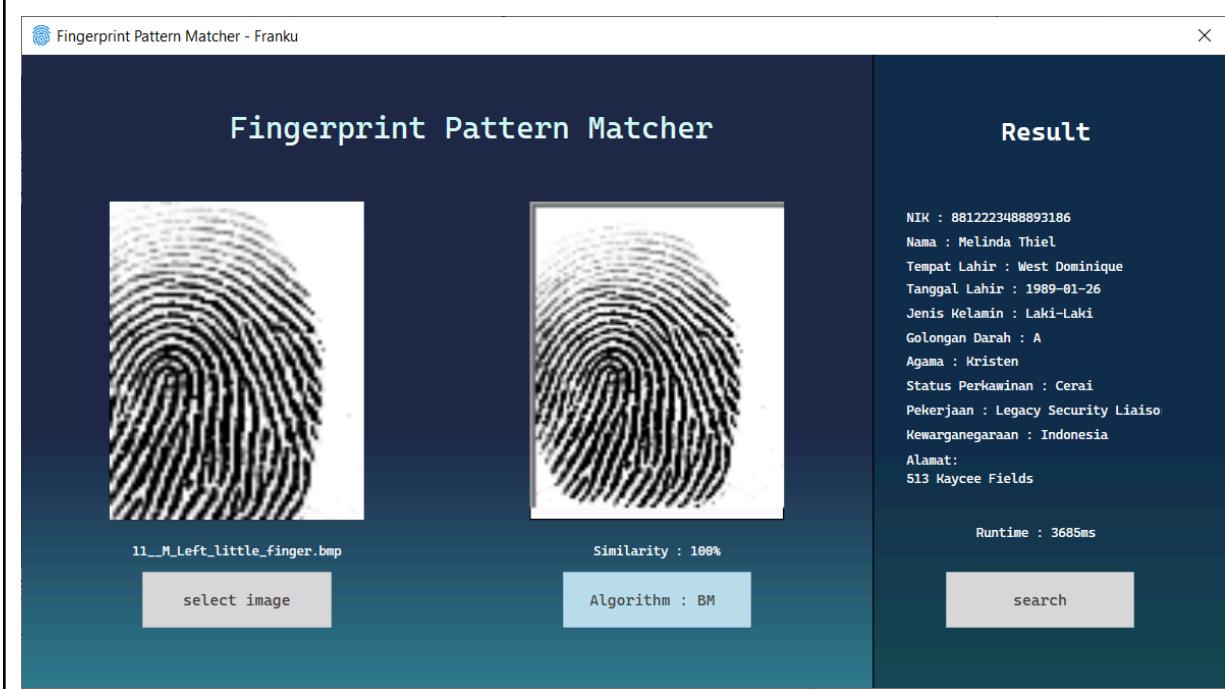
Pencarian gambar exact match pada data yang terletak di awal database menggunakan algoritma KMP



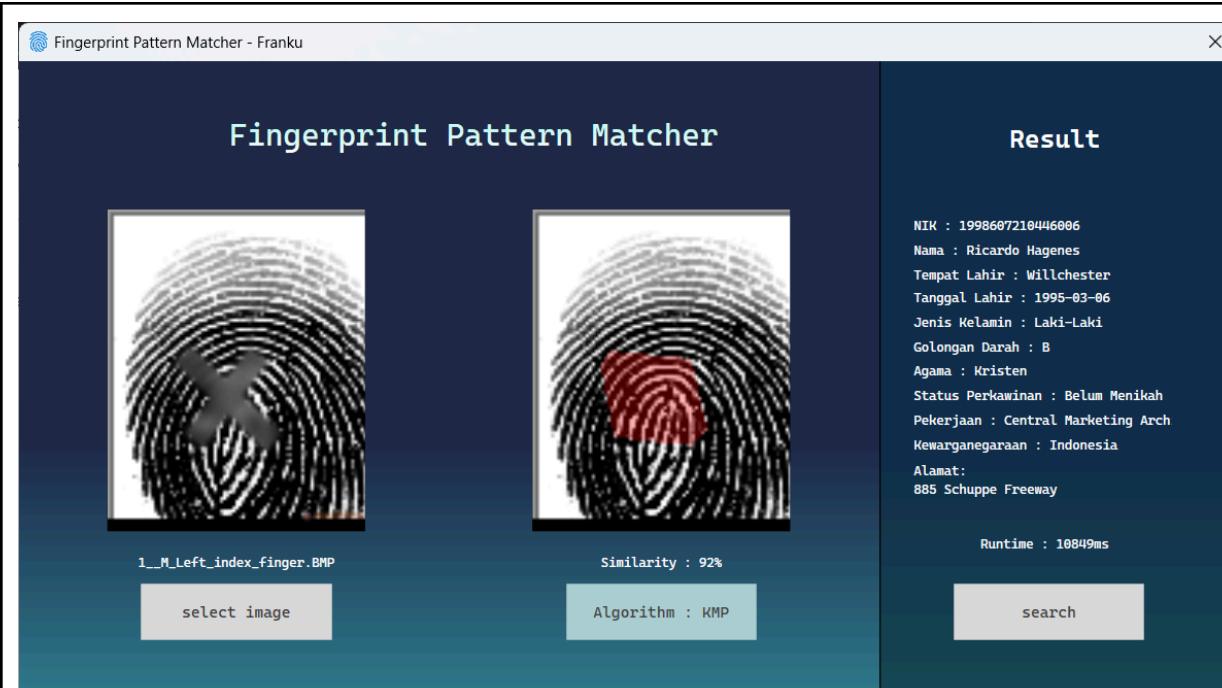
Pencarian gambar exact match pada data yang terletak di awal database menggunakan algoritma BM



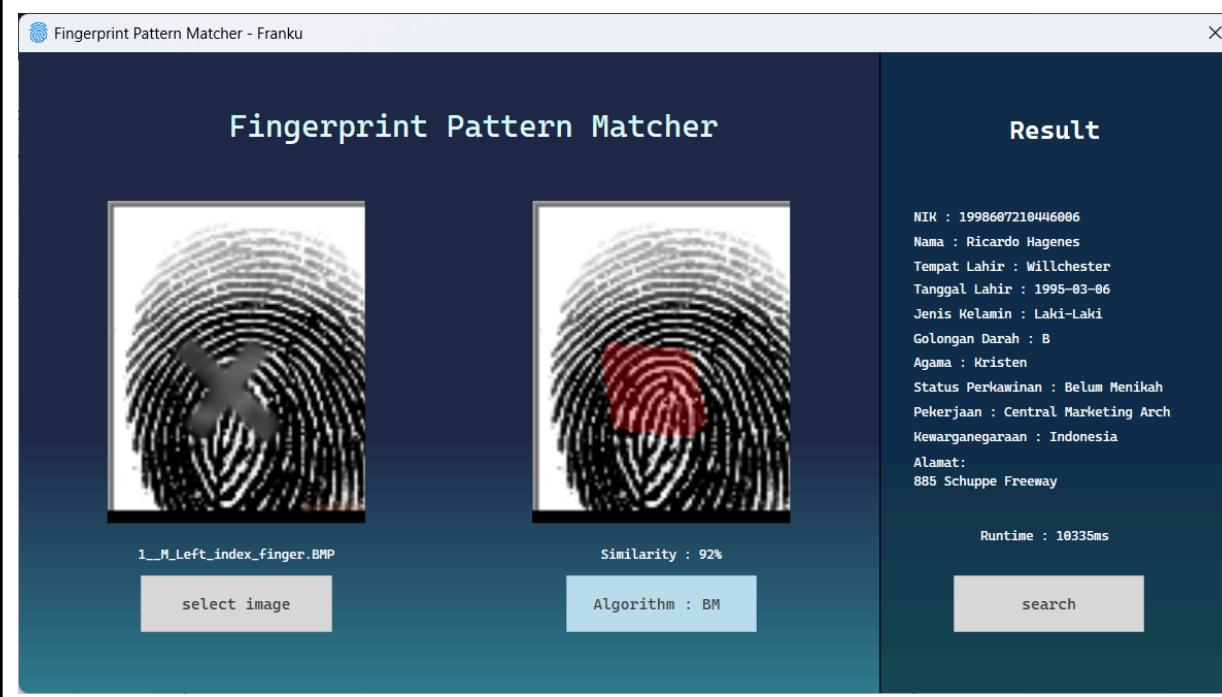
Pencarian gambar *exact match* yang dipotong menggunakan algoritma KMP



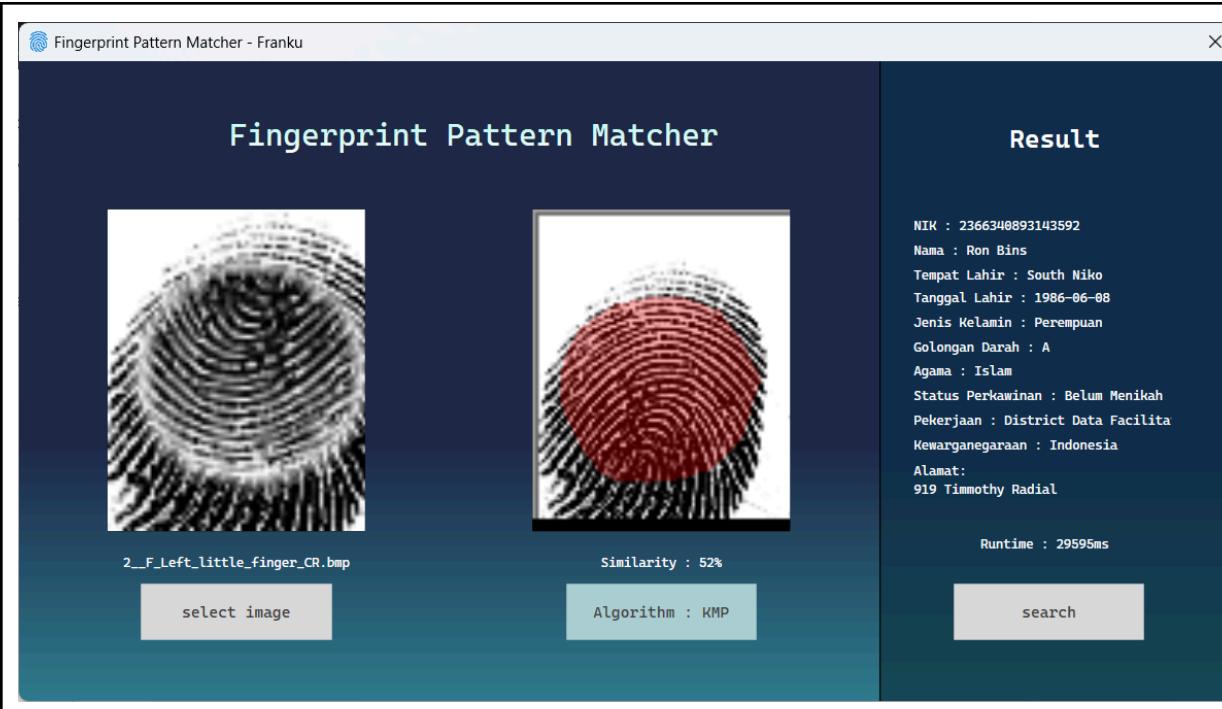
Pencarian gambar *exact match* yang dipotong menggunakan algoritma BM



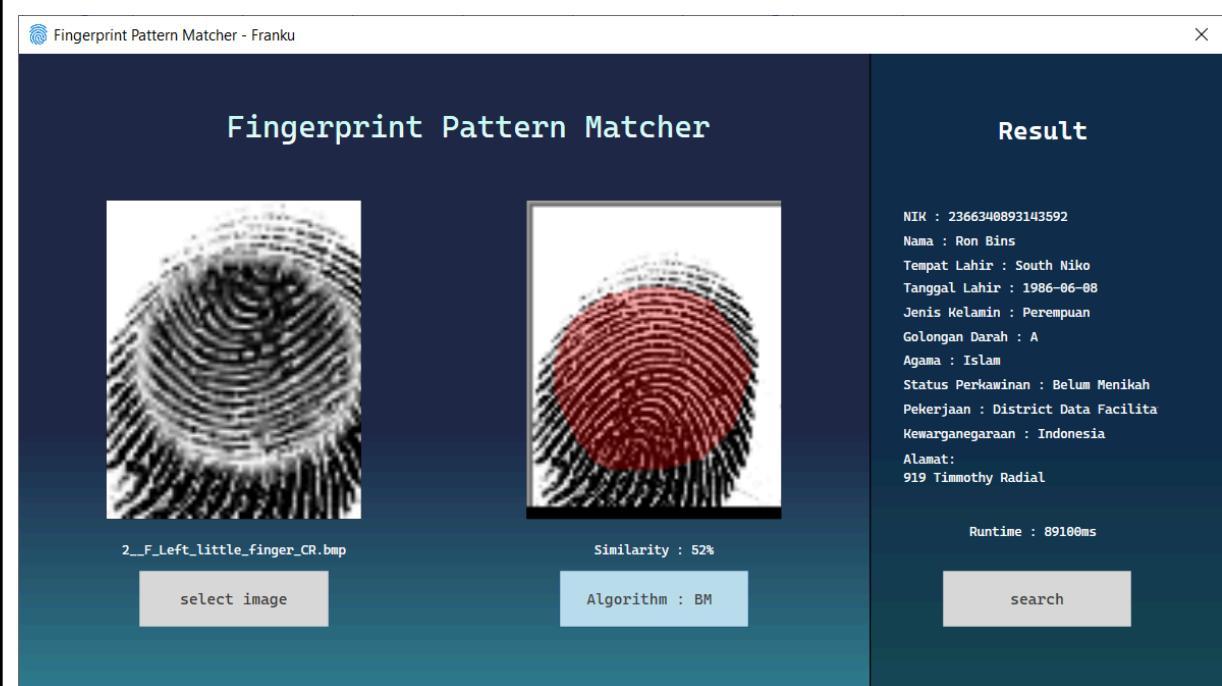
Pencarian gambar non-exact match dengan kerusakan ringan pada data yang terletak di awal database menggunakan algoritma KMP



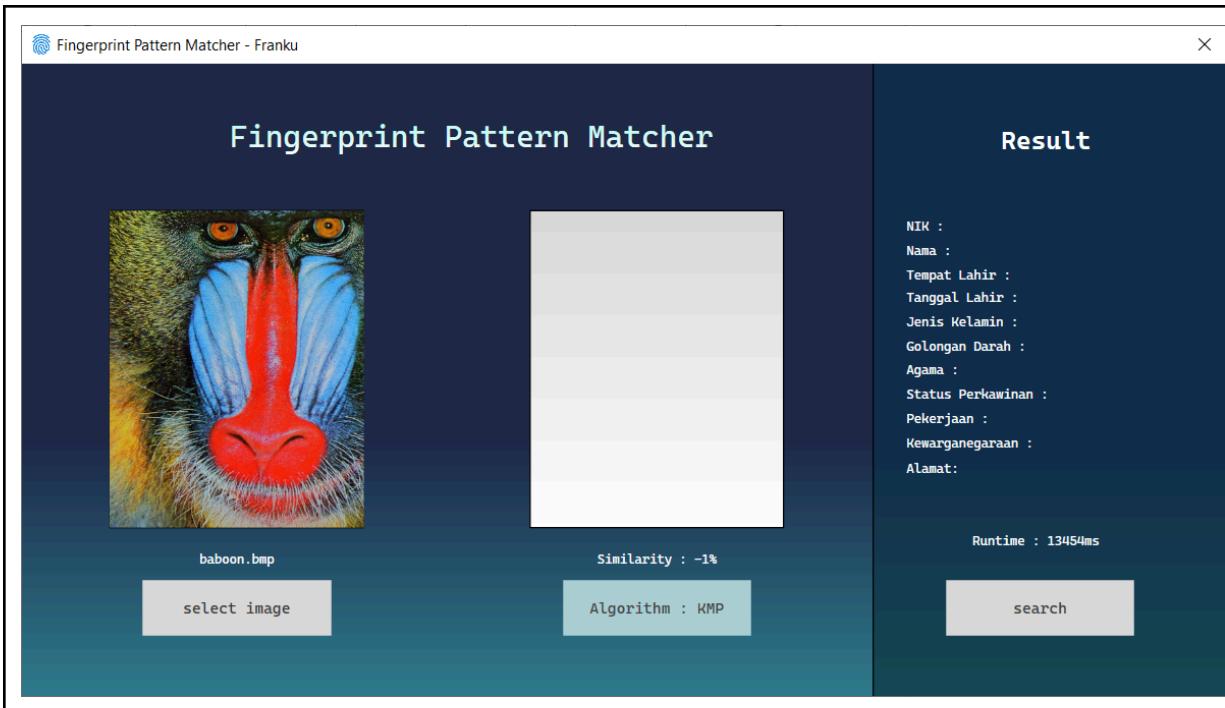
Pencarian gambar non-exact match dengan kerusakan ringan pada data yang terletak di awal database menggunakan algoritma BM



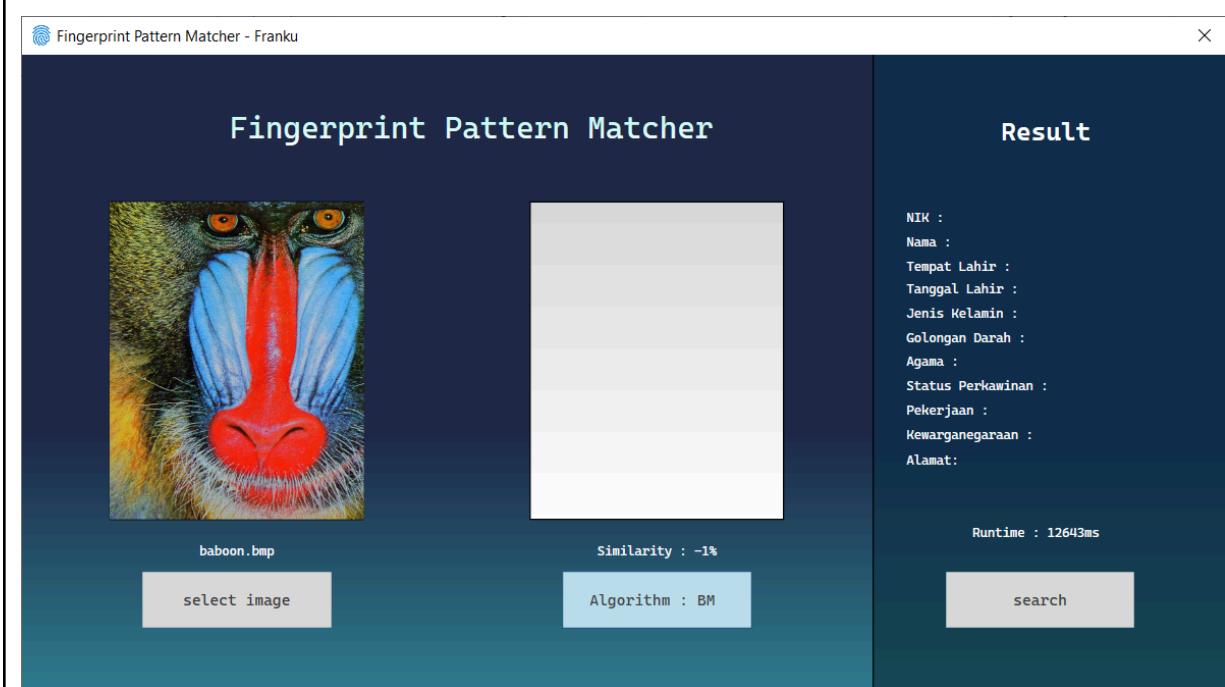
Pencarian gambar non-exact match dengan kerusakan berat pada data yang terletak di awal database menggunakan algoritma KMP



Pencarian gambar non-exact match dengan kerusakan berat pada data yang terletak di awal database menggunakan algoritma BM. Gambar uji terpotong dan memiliki kerusakan berat.



Pencarian gambar yang tidak ada pada *database* menggunakan algoritma KMP



Pencarian gambar yang tidak ada pada *database* menggunakan algoritma BM

#### 4.4 Analisis

**Tabel .** Hasil pengujian beserta deskripsi kasusnya untuk setiap algoritma.

Deskripsi kasus	Waktu algoritma KMP (ms)	Waktu algoritma BM (ms)
Pencarian gambar pada awal <i>database</i>	448	2020
Pencarian gambar pada akhir <i>database</i>	1414	6171
Pencarian gambar yang dipotong	3411	3685
Pencarian gambar dengan kerusakan ringan	10849	10335
Pencarian gambar dengan kerusakan berat	29595	89100
Pencarian gambar yang tidak ada dalam <i>database</i>	13454	12643
Rata-rata waktu	9861	20659

Dari hasil pengujian, dapat dilihat bahwa performa algoritma KMP relatif lebih cepat dibandingkan algoritma BM. Hal ini cukup wajar, mengingat algoritma KMP memiliki kompleksitas waktu linear dalam *worst case*-nya, sedangkan algoritma BM memiliki kompleksitas waktu kuadratik. Menariknya, panjang string *pattern* yang digunakan sebagai input algoritma KMP dan BM berbeda jauh. Algoritma BM akan mengambil  $\frac{1}{2}$  baris gambar sebagai input, sedangkan BM hanya akan mengambil 5% baris gambar. Input algoritma BM 10x lebih kecil dibandingkan algoritma KMP, namun algoritma BM tetap memakan waktu yang lebih lama dibanding algoritma KMP.

Secara rata-rata, algoritma BM memakan waktu 2 kali lebih lama dibandingkan algoritma KMP. Hal ini bisa juga disebabkan karena algoritma Boyer-Moore memiliki performa yang relatif buruk ketika ukuran alfabet kecil. Mengingat program menggunakan alfabet *binary*, algoritma BM dengan performa yang buruk tidak mengagetkan.

Dapat dilihat juga bahwa dalam kasus pencarian gambar yang dipotong tanpa kerusakan tambahan, algoritma KMP dan BM berhasil menemukan gambar asli dengan cepat. Hal ini membuktikan bahwa target program dapat menemukan gambar asli dari gambar yang dipotong dengan cepat berhasil diraih.

## BAB 5

### Penutup

#### 5.1 Kesimpulan

- Algoritma KMP dan Algoritma BM merupakan salah satu algoritma string matching yang dapat digunakan untuk mencari *exact match* dari *pattern* pada suatu *string*.
- Algoritma KMP memiliki kompleksitas waktu dan waktu pencarian yang lebih baik dibandingkan dengan Algoritma BM.
- Algoritma KMP dan Algoritma BM tidak dapat digunakan untuk mencari *partial match* dari *pattern* pada suatu *string*.
- Diperlukan penyesuaian agar Algoritma KMP dan BM dapat memeriksa *partial match* dari gambar yang terpotong di pinggir, yaitu dengan memeriksa satu baris gambar sebagai satu *binary string* daripada menganggap satu gambar sebagai sebuah *binary string* yang sangat panjang.
- Diperlukan algoritma lain selain Algoritma KMP dan Algoritma BM untuk mencari *partial match* dari satu *string* utuh, contohnya *convex hull*.
- Regex dapat digunakan sebagai salah satu cara untuk melakukan pencocokan *string* yang rusak/*corrupt* dengan *string* yang benar.

#### 5.2 Saran

Dari segi algoritma dan konsep dari algoritma KMP, BM, algoritma pencocokan lain dan juga regex sudah cukup menjelaskan dan dapat berjalan dengan baik pada penugasan kali ini. Namun, dari sisi teknis, masih banyak kekurangan dari program yang dibuat. Keterbatasan pengetahuan mengenai pengembangan aplikasi dengan C# dan keterbatasan waktu eksplorasi membuat pengalaman pengguna dalam menggunakan aplikasi ini menjadi sub-optimal. Kedepannya, kami harus lebih banyak eksplorasi mengenai cara kerja bahasa C#. Selain saran untuk kelompok, kami juga memiliki saran mengenai tugas besar 3 ini. Menurut kami, mendorong mahasiswa untuk bereksplorasi menggunakan bahasa baru merupakan hal yang sangat baik. Namun pemberian tugas GUI menggunakan C#, menyarankan untuk membuat project berdasarkan WPF, dan mewajibkan program dapat dijalankan di linux adalah hal yang cukup unik.

#### 5.3 Tanggapan

Tugas ini awalnya sangat menantang. Tugas ini diberikan ketika anggota kelompok kami sudah mendapatkan banyak tugas besar lain dan waktu pengerjaan tugas besar menjadi sangat terbatas. Namun, pemberian kelonggaran waktu tambahan sangat meringankan tugas ini. Tugas ini dapat diselesaikan dengan lebih baik dengan adanya penambahan waktu pengerjaan. Penulis mengucapkan terima kasih yang sebesar-besarnya kepada asisten mata kuliah IF2211 Strategi Algoritma karena telah memperpanjang waktu pengerjaan tugas besar ini .

#### **5.4 Refleksi**

Penugasan kali ini merupakan penugasan yang sangat unik. Berbeda dengan penugasan dari tahun-tahun sebelumnya yang kebanyakan adalah membuat *chatbot* dengan konsep *string matching*, kali ini diberikan penugasan berupa *string matching* untuk melakukan pencarian dan pencocokan sidik jari. Penugasan kali ini juga menggunakan bahasa yang baru bagi kami yaitu menggunakan bahasa C#. Kedepannya kami harus bisa lebih banyak eksplorasi agar penulisan kode C# dapat diimplementasi dengan lebih rapi dan terstruktur.

## **Daftar Pustaka**

- Andem, V. R. (2003). A cryptanalysis of the tiny encryption algorithm (Master's thesis). The University of Alabama, Tuscaloosa, Alabama.
- Dey, D. (2023, May 23). DBSCAN Clustering in ML | Density based clustering. Retrieved from <https://www.geeksforgeeks.org/dbscan-clustering-in-ml-density-based-clustering/>
- Kobe University. (2022, March 1). 2D Fast Fourier Transform Software FFT2D [Software]. Retrieved from [https://www2.kobe-u.ac.jp/~kuroki/OpenSoft/FFT2D/index\\_en.html](https://www2.kobe-u.ac.jp/~kuroki/OpenSoft/FFT2D/index_en.html)
- Rathi, A., & Vagmi, A. (2024, April 29). Convex Hull using Divide and Conquer Algorithm. GeeksforGeeks. Retrieved from <https://www.geeksforgeeks.org/convex-hull-using-divide-and-conquer-algorithm/>
- Thakur, P. (2022, November 8). DBSCAN Algorithm — Density Based Spatial Clustering of Application with Noise. Medium. Retrieved from <https://medium.com/@tpreethi/dbscan-algorithm-density-based-spatial-clustering-of-application-with-noise-a826538dcb42>

## **Lampiran**

Github : [https://github.com/FrancescoMichael/Tubes3\\_Franku](https://github.com/FrancescoMichael/Tubes3_Franku)