

LAPORAN TUGAS BESAR 3

IF2211 STRATEGI ALGORITMA

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



Disusun oleh:

Adril Putra Merin - 13522068

Suthasoma Mahardhika Munthe - 13522098

Berto Richardo Togatorop - 13522118

Program Studi Teknik Informatika

Sekolah Teknik Elektro dan Informatika

Institut Teknologi Bandung

2024

DAFTAR ISI

DAFTAR ISI.....	2
BAB I.....	3
1.1 Penjelasan Persoalan.....	3
BAB II.....	5
2.1. Algoritma KMP, BM, dan Regex.....	5
2.1.1. Algoritma KMP (Knuth-Morris-Pratt).....	5
2.1.2. Algoritma BM (Boyer-Moore).....	6
2.1.3. Regular Expression (Regex).....	7
2.2. Teknik Pengukuran Persentase Kemiripan.....	8
2.3. Aplikasi Desktop yang dibangun.....	8
BAB III.....	10
3.1 Langkah-Langkah Pemecahan Masalah.....	10
3.2 Proses Penyelesaian Solusi dengan Algoritma KMP dan BM.....	11
3.3 Fitur fungsional dan arsitektur aplikasi Desktop yang dibangun.....	12
3.4 Contoh Ilustrasi Kasus.....	13
BAB IV.....	17
4.1 Spesifikasi Teknis Program (struktur data, fungsi, dan prosedur yang dibangun) Berikut adalah spesifikasi teknis program :	17
4.2 Penjelasan Tata Cara Penggunaan Program.....	31
4.3 Hasil Pengujian.....	34
4.4 Analisis Hasil Pengujian.....	39
BAB V.....	40
5.1 Kesimpulan.....	40
5.2 Saran.....	40
5.3 Refleksi.....	40
DAFTAR PUSTAKA.....	41
LAMPIRAN.....	42

BAB I

Deskripsi Tugas

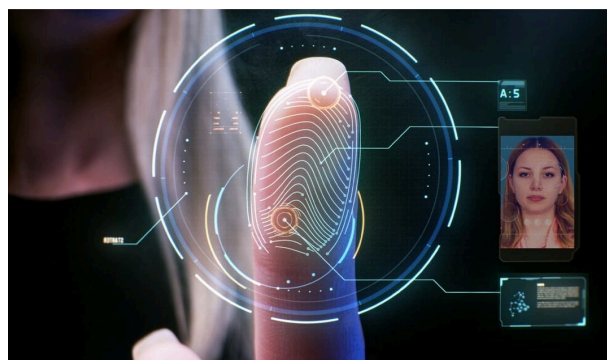
1.1 Penjelasan Persoalan

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.

Metode yang akan digunakan untuk melakukan deteksi sidik jari adalah Boyer-Moore dan Knuth-Morris-Pratt. Selain itu, sistem ini akan dihubungkan dengan identitas sebuah individu melalui basis data sehingga harapannya terbentuk sebuah sistem yang dapat mengenali identitas seseorang secara lengkap hanya dengan menggunakan sidik jari.



Gambar 1. Ilustrasi fingerprint recognition pada deteksi berbasis biometrik.
Sumber: <https://www.aratek.co/news/unlocking-the-secrets-of-fingerprint-recognition>

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.

Seorang pribadi dapat memiliki lebih dari satu berkas citra sidik jari (relasi one-to-many). Akan tetapi, keduanya tidak terhubung dengan sebuah relasi. Hal ini disebabkan karena pada kasus dunia nyata, data yang disimpan bisa saja mengalami korupsi. Dengan membuat atribut kolom yang mungkin korupsi adalah atribut nama pada tabel biodata, maka atribut nama pada tabel sidik_jari tidak dapat memiliki foreign-key yang mereferensi ke tabel biodata. Pada program ini, disimulasikan implementasi data korupsi yang hanya mungkin terjadi pada atribut nama di tabel biodata (asumsikan kolom lain pada setiap tabel tidak mengalami korupsi). 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 korupsi tersebut. Penanganan yang diimplementasikan adalah dengan menggunakan regular expression.

Data yang terdapat dalam KTP merupakan data yang privat dan tidak seharusnya mudah diakses orang yang tidak berwenang. Oleh karena itu, data yang ada di database merupakan hasil dekripsi. Dekripsi yang kami pakai adalah caesar cipher. Jadi, diperlukan dekripsi sebelum memproses data dengan algoritma yang ingin dipakai.

BAB II

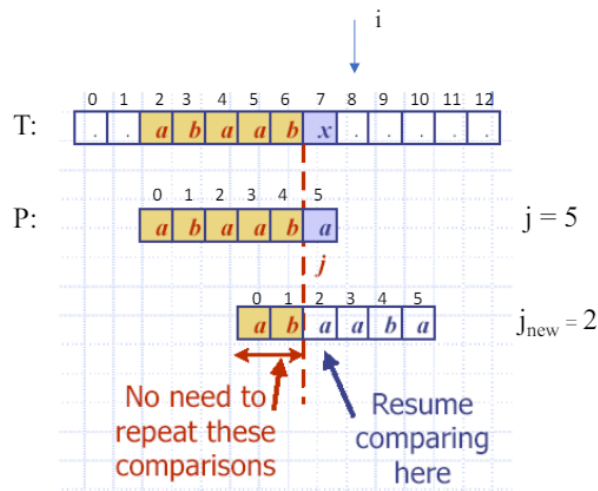
Landasan Teori

2.1. Algoritma KMP, BM, dan Regex

2.1.1. Algoritma KMP (Knuth-Morris-Pratt)

Algoritma Knuth-Morris (KMP) mencari *pattern* pada kumpulan text dari kiri ke kanan (sama seperti algoritma *brute force*). Namun, algoritma ini menggeser pattern dengan lebih “pintar”. Pergeseran yang dilakukan akan menghindari perbandingan karakter yang sudah pasti tidak akan cocok.

Jika terjadi ketidakcocokan pada index pola P pada index j dengan text T pada index i (atau $T[i] \neq P[j]$), maka kita tidak harus menggeser pattern sebanyak 1, melainkan kita bisa menggeser sepanjang *banyaknya prefix* $P[0...j-1]$ yang juga merupakan *suffix* dari $P[1...j]$.



Gambar 2.1.1.1. Ilustrasi pergeseran pada KMP

Banyaknya pergeseran ini dicari menggunakan fungsi pinggiran b (*KMP border function*). Pada algoritma ini, dilakukan preproses untuk mencari $b(k)$ dengan k adalah posisi sebelum terjadi ketidakcocokan pada pattern ($k = j - 1$). Langkah- langkah algoritma KMP :

1. Bandingkan $P[j]$ dengan $T[i]$. Jika $P[j]$ dan $T[j]$ sama, Tambahkan i dan j
2. Jika j mencapai panjang pola, berarti pola ditemukan di teks.
3. Jika $P[j]$ tidak sama dengan $T[i]$ dan $j \neq 0$, maka $j = b(k)$.
4. Jika $P[j]$ tidak sama dengan $T[i]$ dan $j = 0$, maka tambahkan i dengan 1

Dengan P adalah pola dan T adalah text yang dibandingkan.

Kompleksitas waktu untuk pre-proses fungsi pinggiran adalah $O(M)$ dan untuk pencarian string adalah $O(N)$ sehingga kompleksitas waktu keseluruhan adalah

$O(M+N)$ dengan M adalah panjang pola dan N adalah panjang teks dan M adalah panjang pola.

Kelebihan algoritma ini adalah tidak perlu mundur pada text sehingga baik untuk memproses file yang sangat besar. Namun, algoritma ini bekerja kurang baik jika jumlah alphabetnya besar sehingga akan lebih sering tidak cocok di awal.

2.1.2. Algoritma BM (Boyer-Moore)

Algoritma Boyer-Moore didasarkan atas 2 teknik, yaitu :

1. Teknik *looking-glass*

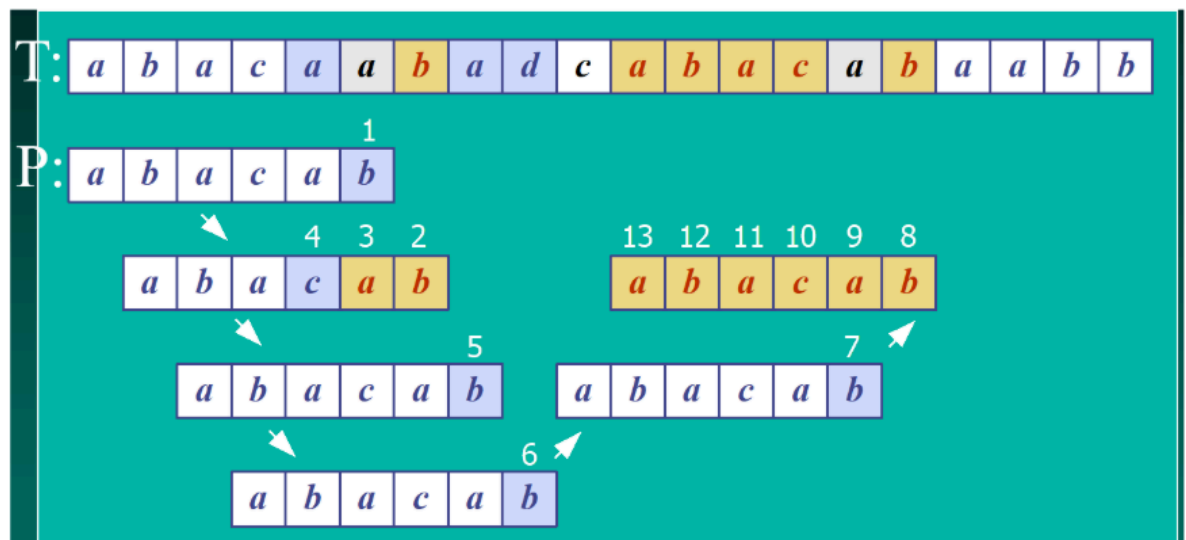
Cari P pada T dengan cara iterasi terbalik dimulai dari indeks akhir P.

2. Teknik *character jump*

Ketika terjadi $P[j] \neq T[i]$ dengan $T[i] = x$, maka ada 3 kemungkinan, coba secara berurutan.

- Kemungkinan 1: Jika P mengandung x, maka geser P untuk agar $T[i]$ sesuai dengan kemunculan terakhir x pada pola.
- Kemungkinan 2: Jika P mengandung x, tetapi pergeseran ke kanan tidak memungkinkan, maka geser P ke kanan sebanyak 1 karakter ke $T[i+1]$
- Kemungkinan 3: Jika kedua kemungkinan tidak terpenuhi, maka geser P sehingga $P[0]$ adalah posisi i yang lama.

Oleh karena itu, diperlukan juga *pre-processing* pada algoritma ini untuk mencari kemunculan terakhir dari tiap karakter pada pola dengan menggunakan fungsi $L()$. $L()$ akan memetakan setiap karakter pada P menjadi angka. $L(x)$ didefinisikan sebagai index terakhir dari x pada P atau -1 jika x tidak ada pada p.



Jumlah perbandingan karakter: 13 kali

x	a	b	c	d
L(x)	4	5	3	-1

Gambar 2.1.2.1 Ilustrasi Pencocokan String menggunakan Algoritma BM

2.1.3. Regular Expression (Regex)

Regex merupakan sebuah teks (string) yang mendefinisikan sebuah pola pencarian sehingga dapat membantu kita untuk melakukan pencocokan dan pencarian teks. Regular Expression akan menentukan himpunan string yang diperlukan untuk hasil yang diinginkan. Cara termudah untuk hal ini adalah mendefinisikan semua elemen atau member dari himpunan itu. Namun, ada cara yang lebih ringkas untuk hal ini yang dapat digapai dengan menggunakan *metacharacter*.

Tabel 2.1.3.1. Penggunaan Metacharacter untuk Regex

Karakter	Deskripsi
^	Cocok dengan posisi awal dalam string. Dalam alat berbasis garis, ini cocok dengan posisi awal garis mana pun.
.	Cocok dengan karakter mana pun (banyak aplikasi mengecualikan baris baru, dan karakter mana yang dianggap sebagai baris baru adalah karakter rasa, pengkodean karakter, dan spesifik platform, namun dapat diasumsikan bahwa karakter umpan baris disertakan). Dalam ekspresi braket POSIX, karakter titik cocok dengan titik literal. Misalnya, <code>a.c</code> cocok dengan <code>"abc"</code> , dll., tetapi <code>[a.c]</code> hanya cocok dengan <code>"a"</code> , <code>"."</code> , atau <code>"c"</code> .
[]	Ekspresi dalam tanda kurung. Cocok dengan satu karakter yang ada di dalam tanda kurung. Misalnya, <code>[abc]</code> cocok dengan <code>"a"</code> , <code>"b"</code> , atau <code>"c"</code> . <code>[a-z]</code> menentukan rentang yang cocok dengan huruf kecil apa pun dari <code>"a"</code> hingga <code>"z"</code> . Bentuk-bentuk ini dapat dicampur: <code>[abcx-z]</code> cocok dengan <code>"a"</code> , <code>"b"</code> , <code>"c"</code> , <code>"x"</code> , <code>"y"</code> , atau <code>"z"</code> , seperti halnya <code>[a-cx-z]</code> . Karakter - diperlakukan sebagai karakter literal jika merupakan karakter terakhir atau pertama (setelah <code>^</code> , jika ada) di dalam tanda kurung: <code>[abc-]</code> , <code>[-abc]</code> , <code>^[^abc]</code> . Pelarian garis miring terbalik tidak diperbolehkan. Karakter <code>]</code> dapat disertakan dalam ekspresi tanda kurung jika karakter tersebut merupakan karakter pertama (setelah <code>^</code> , jika ada): <code>[]abc]</code> , <code>^[^]abc]</code> .
[^]	Mencocokkan satu karakter yang tidak terdapat dalam tanda kurung. Misalnya, <code>[^abc]</code> cocok dengan karakter apa pun selain <code>"a"</code> , <code>"b"</code> , atau <code>"c"</code> . <code>[^a-z]</code> cocok dengan karakter apa pun yang bukan huruf kecil dari <code>"a"</code> hingga <code>"z"</code> . Demikian pula, karakter literal dan rentang dapat dicampur.
\$	Cocok dengan posisi akhir string atau posisi sebelum baris baru yang mengakhiri string. Dalam alat berbasis garis, ini cocok dengan posisi akhir garis mana pun.
()	Mendefinisikan subekspresi yang ditandai. String yang cocok dengan

	induknya dapat dipanggil kembali nanti (lihat entri berikutnya, \n). Subekspresi yang ditandai juga disebut blok atau grup penangkap. Mode BRE memerlukan \ (\).
\$	Cocok dengan subekspresi yang ditandai ke-n, dengan n adalah digit dari 1 hingga 9. Konstruksi ini didefinisikan dalam standar POSIX.[35] Beberapa alat mengizinkan referensi lebih dari sembilan grup penangkap. Juga dikenal sebagai referensi kembali, fitur ini didukung dalam mode BRE.
*	Mencocokkan elemen sebelumnya sebanyak nol kali atau lebih. Misalnya, ab*c cocok dengan "ac", "abc", "abbbc", dll. [xyz]* cocok dengan "", "x", "y", "z", "zx", "zyx", "xyzzy", dan seterusnya. (ab)* cocok dengan "", "ab", "abab", "ababab", dan seterusnya.
{m,n}	Cocok dengan elemen sebelumnya setidaknya m dan tidak lebih dari n kali. Misalnya, a{3,5} hanya cocok dengan "aaa", "aaaa", dan "aaaaa". Ini tidak ditemukan di beberapa contoh regex lama. Mode BRE memerlukan \{m,n\}.

2.2. Teknik Pengukuran Persentase Kemiripan.

Sidik jari yang ingin dibandingkan belum tentu 100% mirip dengan sidik jari yang ada di database. Jika hal ini terjadi, maka dicari gambar lain yang persentase kemiripannya paling tinggi. Teknik pengukuran persentase kemiripan yang kami pakai adalah teknik Hamming Distance. Hamming distance antara dua string adalah banyaknya karakter yang berbeda pada posisi yang sama dari kedua string yang dibandingkan. Untuk mencari persentase kemiripan, akan dicari hamming distance antara string pola dan string minimal pada string sidik jari database. Karena itu, tidak perlu dilakukan validasi Langkah-langkah pencarian hamming distance adalah

1. Inisialisasi nilai counter = 0
2. Iterasi kedua string, misalkan S1 dan S2. Jika $S1[i] \neq S2[i]$, maka tambahkan ke counter
3. Kembalikan nilai counter. Jika counter = -1

Persentase kemiripan dari suatu gambar adalah:

$$\frac{\text{Banyaknya Karakter Pada Pola} - \text{Hamming Distance}}{\text{Banyaknya Karakter Pada Pola}} \times 100\%$$

2.3. Aplikasi Desktop yang dibangun

Aplikasi desktop dibangun menggunakan C#. Aplikasi terhubung dengan database dengan relasi “biodata” (yang berisi nama alay, tempat lahir, tanggal lahir, jenis kelamin, golongan darah, alamat, agama, status perkawinan, pekerjaan dan kewarganegaraan) dan relasi

“sidik_jari” yang berisi berkas citra serta nama asli. Relasi biodata merupakan relasi dengan data yang terenkripsi dengan metode caesar cipher. Aplikasi akan menerima sebuah gambar dari user dan menampilkan sidik jari serta biodata user yang sesuai dengan sidik jari masukan.

BAB III

Analisis Pemecahan Masalah

3.1 Langkah-Langkah Pemecahan Masalah

Pencarian sidik jari dapat diselesaikan dengan algoritma string matching. Hasil dari string matching adalah nama asli dan akan dipakai regex untuk menentukan biodata yang sesuai.

1. Terima input gambar

Langkah awal dari solusi ini adalah dengan menerima input gambar. Gambar inilah yang akan dicari apakah ada yang mirip dengan database yang ada. Gambar berukuran $m \times n$ pixel.

2. Ubah Gambar masukan menjadi binary dan ubah lagi menjadi karakter ASCII 8-bit.

Sebuah gambar tidak bisa langsung dibandingkan. Oleh karena itu, kami mengambil 40 pixel gambar. Pixel-pixel yang diambil ini kemudian diubah menjadi binary. Binary 0 menandakan hitam dan binary 1 menandakan putih. Namun, algoritma KMP dan BM lebih lambat untuk variasi karakter yang sedikit, maka binary tadi diubah menjadi karakter ASCII 8-bit.

3. Ambil List nama asli dan path ke gambar dari relasi “sidik_jari”

Selanjutnya, diambil list nama dan path ke gambar dari database pada relasi “sidik_jari”. Pengambilan data terlebih dahulu melalui proses dekripsi. Cara dekripsi adalah dengan menggunakan metode Caesar Cipher. Banyak shift nya bergantung pada panjang string.

4. Bandingkan ASCII 8-bit dengan Algoritma KMP atau BM

Selanjutnya, untuk seluruh path, akan diambil gambar sidik jarinya. Gambar ini dikonversi menjadi pixel, lalu binary, kemudian menjadi String ASCII 8-bit. Lalu, dilakukan *string matching* dengan hasil String ASCII 8-bit pada gambar masukan sebagai sebagai pola atau *pattern* dan String hasil konversi gambar database sebagai Text. Metode *pattern matching*nya menggunakan KMP atau BM, tergantung masukan user. Ketika ditemukan hasil yang *exact match*, maka dikembalikan nama asli yang bersesuaian dengan gambar tersebut.

5. Jika tidak ada yang tepat sama, cari gambar dengan kemiripan paling tinggi

Metodenya adalah dengan mencari hamming distance. Dilakukan iterasi terhadap semua gambar di database. Diambil terlebih dahulu string minimal pada gambar database. Setelah itu, dicari hamming distance dan persentasenya berdasarkan *hamming distance* tersebut. Untuk setiap iterasi, disimpan persentase dan path ke gambar untuk persentase paling tinggi. **Program pasti mengeluarkan biodata untuk sidik jari dengan persentase kemiripan paling tinggi, meskipun persentasenya rendah.** Hal ini

agar user lebih leluasa untuk menentukan apakah persentase tersebut sudah cukup mirip untuk dijadikan valid.

6. Ambil Pair nama alay dan NIK dari relasi biodata

Nama asli yang didapatkan tidak ada di relasi biodata. Melainkan, yang ada hanyalah nama alay. Oleh karena itu, akan diambil nama alay dan NIK yang bersesuaian. Nama asli diambil untuk dibandingkan dengan nama asli yang sudah didapat. Sementara itu, NIK diambil karena merupakan *primary key* sehingga nantinya bisa ditentukan biodata mana yang sesuai jika diketahui NIK nya.

7. Gunakan regex untuk mencari nama alay yang sesuai dengan nama asli yang didapatkan.

Selanjutnya, nama asli akan menghasilkan regex untuk mencari nama alay nama tersebut. List Pair yang sudah ada sebelumnya akan dibandingkan dengan regex tersebut. Jika didapatkan nama yang memenuhi regex, maka NIK pair tersebut akan dikembalikan. Hasil kontruksi nama alay merupakan kombinasi dari pergantian huruf besar-kecil, penggunaan angka, dan penyingkatan.

8. Tampilkan biodata yang didapatkan.

NIK yang didapatkan akan dipakai untuk query ke database. NIK merupakan *primary key* sehingga *determines* relasi. Oleh karena itu, bisa diambil biodata yang sesuai dan ditampilkan ke user.

3.2 Proses Penyelesaian Solusi dengan Algoritma KMP dan BM

Gambar masukan dan gambar dari database akan di *pre-process* terlebih dahulu. Citra awalnya diubah menjadi bentuk grayscale. Citra yang sudah diubah ini kemudian diubah menjadi bentuk binary dengan 0 untuk grayscale yang lebih dari 128 untuk merepresentasikan terang dan binary 1 untuk grayscale yang kurang atau sama dengan 128. Lalu, untuk setiap baris, akan diambil 40 binary dengan rasio gelap terang yang seimbang agar tidak hanya mengambil background. Lalu, dipilih baris yang proporsinya paling seimbang untuk dijadikan sebagai pola atau *pattern*. Binary pada baris ini diubah terlebih dahulu menjadi ASCII-8 bit yang menjadi 5 karakter yang berguna sebagai *pattern*.

Sementara itu, citra yang berkasnya ada pada database juga diubah menjadi grayscale terlebih dahulu. Hasil grayscale ini kemudian diubah menjadi binary dengan mekanisme yang sama untuk *pattern*. Namun, keseluruhan gambar pada database diambil dan tidak dipotong. Jika hasil konversi binary tidak habis dibagi 8, maka akan ada sisa yang tidak bisa dijadikan ASCII-8 bit. Oleh karena itu, kami menambahkan padding berupa binary 0 untuk sisa binary tersebut. Hasil string dari database ini kemudian menjadi *teks*.

3.2.1. Proses penyelesaian dengan KMP

Karena masalah sudah dipetakan menjadi pattern dan teks, maka penggunaan KMP dilakukan seperti biasanya, yaitu membandingkan pattern dengan teks. Pattern yang dihasilkan akan dibandingkan dengan citra yang ada pada database. Perbandingan ini dilakukan dengan perulangan. Untuk setiap perbandingan, akan dicari list Lps yang merepresentasikan fungsi $b(k)$ untuk KMP. Setelah itu, akan dicari apakah pattern ada pada teks sesuai dengan algoritma KMP yang sudah dijelaskan pada bagian 2.1.1. Jika ada citra di database yang mengandung pattern, maka perulangan dihentikan dan langsung dikembalikan nama yang sesuai. Jika tidak, akan dicari citra dengan tingkat kemiripan paling tinggi dengan menggunakan hamming distance.

3.2.2 Proses Penyelesaian dengan BM

Karena masalah sudah dipetakan menjadi pattern dan teks, maka penggunaan algoritma BM dilakukan seperti biasanya, yaitu membandingkan pattern dengan teks. Pattern yang dihasilkan akan dibandingkan dengan citra yang ada pada database. Perbandingan ini dilakukan dengan perulangan. Untuk setiap perbandingan, akan dicari list last yang merepresentasikan fungsi *last occurrence* $L()$ untuk BM. Setelah itu, akan dicari apakah pattern ada pada teks sesuai dengan algoritma BM yang sudah dijelaskan pada bagian 2.1.2. Jika ada citra di database yang mengandung pattern, maka perulangan dihentikan dan langsung dikembalikan nama yang sesuai. Jika tidak, akan dicari citra dengan tingkat kemiripan paling tinggi dengan menggunakan hamming distance.

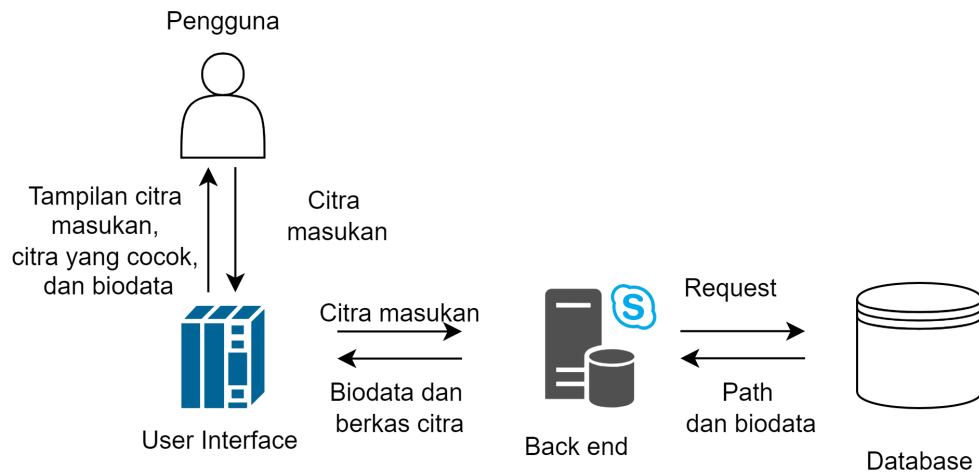
3.3 Fitur fungsional dan arsitektur aplikasi Desktop yang dibangun

Aplikasi desktop yang dibangun memiliki fitur fungsional sebagai berikut :

1. Input sidik jari
Sidik jari yang dimasukkan merupakan gambar dengan ekstension image bitmap.
2. Pemilihan Algoritma String Matching (KMP atau BM)
Komponen ini bertugas agar user bisa memilih algoritma yang ingin digunakan untuk *string matching* (KMP atau BM).
3. Starting Link
Komponen ini bertugas untuk memberikan pengguna tempat untuk memasukkan judul artikel Wikipedia yang akan dijadikan judul awal atau starting point dari penelusuran artikel Wikipedia.
4. Penampilan Sidik jari dan Biodata
Jika ditemukan sidik jari yang sesuai, maka sidik jari tersebut ditampilkan di samping

sidik jari masukan. Lalu, di ujung kanan akan ditampilkan biodata untuk pemilik sidik jari tersebut.

Aplikasi desktop yang dibangun memiliki arsitektur sebagai berikut :



Gambar 3.3.1 Arsitektur aplikasi yang dibangun

1. User interface berfungsi untuk menampilkan hal yang bisa langsung diakses oleh user. User bisa memasukkan citra masukan dan memerintahkan program untuk mencari orang yang memiliki sidik jari yang sesuai.
2. Back end berfungsi untuk memproses data masukan dan mencari sidik jari yang sesuai di database. Back end kemudian mengembalikan biodata untuk orang yang sidik jarinya paling mirip dengan citra masukan.
3. Database menyimpan dua relasi. Relasi pertama adalah relasi biodata yang komul namanya korup sehingga menjadi format nama alay. Relasi kedua adalah relasi sidik jari yang berisi nama asli dan berkas menuju citra sidik jari mereka.


3.4 Contoh Ilustrasi Kasus

Untuk memperjelas pemecahan masalah, penulis menyediakan beberapa ilustrasi kasus sebagai berikut.

A. Sidik Jari Memiliki *Exact Match*

Misalkan sidik jari berikut adalah sidik jari yang ingin dicari datanya. Kita akan mengambil 40 bit dengan banyak bit hitam dan bit putih yang paling tidak berbeda jauh sehingga kita memperoleh text pattern sebagai berikut.

Citra Sidik Jari	Potongan nilai binary	Potongan ASCII-8-bits
------------------	-----------------------	-----------------------

	<pre>111111111000000100010001 111111111000000000</pre>	$\ddot{y}?$ $\blacktriangleleft \ddot{y}$
---	--	---

Dengan menggunakan *pattern* tersebut, program akan melakukan pencarian terhadap seluruh gambar sidik jari pada database menggunakan algoritma yang dipilih pengguna. Algoritma tersebut bekerja sesuai dengan deskripsi cara kerjanya pada bagian sebelumnya. Pada kasus ini, gambar sidik jari tersebut terdapat di dalam *database* (memiliki *exact match*). Berikut adalah potongan nilai ASCII-8 bits yang bersesuaian dengan seluruh pixel pada gambar di atas.

[illegible]

Terlihat bahwa pada bagian yang di-highlight pada *string text*, terdapat kesesuaian dengan *string pattern*. Dengan demikian, program akan dapat menemukan gambar yang bersesuaian. Setelah menemukan gambar yang bersesuaian, program akan mencari nama yang bersesuaian dengan nama pemilik sidik jari diatas dengan menggunakan regex.

B. Sidik Jari Tidak Memiliki *Exact Match*

Pada kasus ini, string matching dengan KMP atau BM tidak dapat menemukan gambar sidik jari yang 100% cocok dengan gambar sidik jari yang dicari. Karena, itu program akan

mengubah semua pixel dari gambar sidik jari yang dicari ke dalam larik ASCII-8 bits. Larik tersebut kemudian akan dibandingkan untuk setiap gambar pada *database* yang juga sudah diubah ke dalam ASCII-8 bits. Program akan memilih gambar sidik jari dengan persentase kemiripan paling besar untuk didapatkan datanya. Berikut adalah citra sidik jari dan larik ASCII-8 bits dari gambar sidik jari yang diinginkan dan *match*-nya di *database*.

[illegible]

[illegible]

Program akan mencari hamming distance antara kedua larik tersebut dan menentukan persentase kemiripan sesuai dengan formula berikut.

$$\frac{\text{Banyaknya Karakter Pada Larik} - \text{Hamming Distance}}{\text{Banyaknya Karakter Pada Larik}} \times 100\%$$

sehingga didapatkan hasil persentase sebesar 91,262%. Setelah menemukan gambar yang bersesuaian, program akan mencari nama yang bersesuaian dengan nama pemilik sidik jari diatas dengan menggunakan regex.

BAB IV

Eksperimen

4.1 Spesifikasi Teknis Program (struktur data, fungsi, dan prosedur yang dibangun)

Berikut adalah spesifikasi teknis program :

a. Modul GUI

Merupakan modul utama yang mengurus *behavior* GUI.

```
using Microsoft.Win32;
using System.Text;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Diagnostics;
using System.Windows.Shapes;
using System.Data.SQLite;
using treelilnick.connector;
using treelilnick.imageloader;
using treelilnick.preprocess;
using treelilnick.algorithm;
using treelilnick.regex;
using treelilnick.decryptor;
using System.Collections.Generic;
using System.Security.Cryptography;

namespace treelilnick
{
    /// <summary>
    /// Interaction logic for MainWindow.xaml
    /// </summary>
    public partial class MainWindow : Window
    {
        private BitmapImage? queryImage;
        // private BitmapImage? resultImage;
        public MainWindow()
        {
            queryImage = null;
            InitializeComponent();
        }

        private void ButtonSearch_Click(object sender, RoutedEventArgs e)
        {
            try
            {
                bool? isKMP = algoTypeToggle.IsChecked;
                listView.Items.Clear();
                if (queryImage == null)
                {
                    MessageBox.Show("Please insert your query image");
                    return;
                }
                // connect to database
                Database conn = new Database();
                List<Pair<string, string>> pairs = conn.GetDataSidikJari();

                // start the algorithm
                var stopwatch = new Stopwatch();
            }
        }
    }
}
```

```

        stopwatch.Start();
        BitmapSource queryGrayScale =
ImagePreprocess.ConvertToGrayscale(queryImage);
        string pattern =
ImagePreprocess.ConvertToASCIIIPattern(queryGrayScale);
        bool found = false;
        int indexFound = -1;
        double maxPercentage = 0;
        int i = 0;

        for (i = 0; i < pairs.Count; i++)
        {
            string imagePath = pairs[i].First;
            imagePath = "../test/" + imagePath;
            BitmapSource imageBitmap = ImageLoader.LoadImage(imagePath);
            BitmapSource imageGrayScale =
ImagePreprocess.ConvertToGrayscale(imageBitmap);
            string text =
ImagePreprocess.ConvertToASCIIString(imageGrayScale);
            if (isKMP.HasValue && (bool)isKMP)
            {
                found = KMP.FindPattern(text, pattern);
            }
            else
            {
                found = BayerMoore.FindPattern(text, pattern);
            }
            if (found)
            {
                indexFound = i;
                maxPercentage = 100;
                break;
            }
        }

        if (!found)
        {
            string queryAsciiString =
ImagePreprocess.ConvertToASCIIString(queryGrayScale);
            for (i = 0; i < pairs.Count; i++)
            {
                string imagePath = pairs[i].First;
                imagePath = "../test/" + imagePath;
                BitmapSource imageBitmap = ImageLoader.LoadImage(imagePath);
                BitmapSource imageGrayScale =
ImagePreprocess.ConvertToGrayscale(imageBitmap);
                string text =
ImagePreprocess.ConvertToASCIIString(imageGrayScale);
                int hammingDist = ClosestString.HammingDistance(text,
queryAsciiString);
                double percentage = (1 - (hammingDist /
(double)Math.Min(text.Length, queryAsciiString.Length))) * 100;
                if (percentage > maxPercentage)
                {
                    maxPercentage = percentage;
                    indexFound = i;
                }
            }
        }
        // end of algorithm
        stopwatch.Stop();
        long time = stopwatch.ElapsedMilliseconds;

        outputImage.Source = ImageLoader.LoadImage("../test/" +
pairs[indexFound].First);
        timeLabel.Text = "Waktu Pencarian: " + time.ToString() + "ms";
        percentageLabel.Text = "Persentase Kecocokan: " +
maxPercentage.ToString("F3") + "%";

        List<Pair<string, string>> listAlay = conn.GetNamaAndNIK();

```

```

        Pair<string, string> matched;
        try
        {
            matched = AlayRegex.SearchAlay(pairs[indexFound].Second, listAlay);
            List<string> data = conn.GetBioData(matched.Second);
            listView.Items.Add("NIK:\t\t\t"+data[Database.NIK]);
            listView.Items.Add("Nama:\t\t\t"+pairs[indexFound].Second);
            listView.Items.Add("Tempat
Lahir:\t\t"+Decrypt.DecryptCipher(data[Database.TEMPAT_LAHIR]));
            listView.Items.Add("Tanggal
Lahir:\t\t"+data[Database.TANGGAL_LAHIR]);
            listView.Items.Add("Jenis
Kelamin:\t\t"+data[Database.JENIS_KELAMIN]);
            listView.Items.Add("Golongan
Darah:\t\t"+data[Database.GOLONGAN_DARAH]);

            listView.Items.Add("Alamat:\t\t\t"+Decrypt.DecryptCipher(data[Database.ALAMAT]));

            listView.Items.Add("Agama:\t\t\t"+Decrypt.DecryptCipher(data[Database.AGAMA]));
            listView.Items.Add("Status
Perkawinan:\t"+data[Database.STATUS_PERKAWINAN]);

            listView.Items.Add("Pekerjaan:\t\t"+Decrypt.DecryptCipher(data[Database.PEKERJAAN]));

            listView.Items.Add("Kewarganegaraan:\t"+Decrypt.DecryptCipher(data[Database.KEWARGANEGA
RAAN]));
        }
        catch (Exception)
        {
            throw;
        }
    }
    catch (Exception)
    {
        throw;
    }
}

private void ButtonUpload_Click(object sender, RoutedEventArgs e)
{
    OpenFileDialog openFileDialog = new OpenFileDialog();
    openFileDialog.Filter = "Image files | *.bmp;*.jpg;*.png";
    openFileDialog.FilterIndex = 1;
    if (openFileDialog.ShowDialog() == true)
    {
        inputImage.Source = new BitmapImage(new Uri(openFileDialog.FileName));
        queryImage = new BitmapImage(new Uri(openFileDialog.FileName));
    }
    listView.Items.Clear();
}
}
}

```

b. Modul connector

i. Public class Database

Berguna untuk menghubungkan back-end program ke database. Kelas ini mengandung :

- Selektor dan Konstruktor kelas ini
- Atribut statik connection bertipe SQLiteConnection

- Metode `GetDataSidikJari` yang bertugas untuk mengembalikan nama asli dan berkas file citra dengan tipe berupa `List<Pair<string, string>>` . Data ini didapatkan dari relasi `sidik_jari`.
- Metode `GetNamaAndNIK` yang bertugas mengembalikan nama alay dan NIK yang berupa `List<Pair<string,string>>`. Data ini didapatkan dari relasi `biodata`.
- Metode `GetBioData` yang bertugas untuk mengambil biodata yang sesuai berdasarkan NIK. Biodata ini disatukan menjadi sebuah `List<string>`

```
using System;
using System.Collections.Generic;
using System.Data.SQLite;
using System.Security.Cryptography.X509Certificates;
using System.Windows;

namespace treelilnick.connector
{
    public class Database
    {
        private static SQLiteConnection connection;

        public static SQLiteConnection Connection
        {
            get { return connection; }
            set { connection = value; }
        }

        static Database()
        {
            connection = new SQLiteConnection("Data Source=SidikJari.db;Version=3;");
            connection.Open();

            Application.Current.Exit += (sender, e) => connection.Close();
        }

        public string GetName()
        {
            return "Connector";
        }

        // List<Pair<string, string>>
        public List<Pair<string, string>> GetDataSidikJari()
        {
            // Berkas citra : First, nama : Second
            List<Pair<string, string>> data = new List<Pair<string, string>>();

            using (SQLiteCommand selectCommand = new SQLiteCommand("SELECT * FROM
sidik_jari", Connection))
            {
                using (SQLiteDataReader reader = selectCommand.ExecuteReader())
                {
                    while (reader.Read())
                    {
                        #pragma warning disable CS8604 // Possible null reference argument.
                        #pragma warning disable CS8604 // Possible null reference argument.
                        data.Add(new Pair<string,
string>(reader["berkas_citra"].ToString(), reader["nama"].ToString()));
                        #pragma warning restore CS8604 // Possible null reference argument.
                        #pragma warning restore CS8604 // Possible null reference argument.
                    }
                }

                if (data.Count > 0)
                {
                    MessageBox.Show(data[0].First + " " + data[0].Second);
                }
            }
        }
    }
}
```

```

    }

    return data;
}

public List<Pair<string,string>> GetNamaAndNIK()
{
    // Nama: First, NIK : Second;
    List<Pair<string, string>> pairs = new List<Pair<string, string>>();
    using (SQLiteCommand selectCommand = new SQLiteCommand("SELECT nama,NIK FROM
biodata", Connection))
    {
        using (SQLiteDataReader reader = selectCommand.ExecuteReader())
        {
            while (reader.Read())
            {
                #pragma warning disable CS8604 // Possible null reference argument.
                #pragma warning disable CS8604 // Possible null reference argument.
                pairs.Add(new Pair<string, string>(reader["nama"].ToString(),
reader["NIK"].ToString()));
                #pragma warning restore CS8604 // Possible null reference argument.
                #pragma warning restore CS8604 // Possible null reference argument.
            }
        }
    }
    return pairs;
}

public List<string> GetBioData(string nik)
{
    List<string> biodata = new List<string>();
    try
    {
        using (SQLiteCommand selectCommand = new SQLiteCommand("SELECT * FROM
biodata WHERE NIK=@nik", Connection))
        {
            selectCommand.Parameters.AddWithValue("@nik", nik);

            using (SQLiteDataReader reader = selectCommand.ExecuteReader())
            {
                if (reader.Read())
                {
                    #pragma warning disable CS8604 // Possible null reference argument.
                    biodata.Add(reader["NIK"].ToString());
                    #pragma warning restore CS8604 // Possible null reference argument.
                    #pragma warning disable CS8604 // Possible null reference argument.
                    biodata.Add(reader["nama"].ToString());
                    #pragma warning restore CS8604 // Possible null reference argument.
                    #pragma warning disable CS8604 // Possible null reference argument.
                    biodata.Add(reader["tempat_lahir"].ToString());
                    #pragma warning restore CS8604 // Possible null reference argument.
                    #pragma warning disable CS8602 // Dereference of a possibly null reference.
                    biodata.Add(reader["tanggal_lahir"].ToString().Substring(0,10));
                    #pragma warning restore CS8602 // Dereference of a possibly null reference.
                    #pragma warning disable CS8604 // Possible null reference argument.
                    biodata.Add(reader["jenis_kelamin"].ToString());
                    #pragma warning restore CS8604 // Possible null reference argument.
                    #pragma warning disable CS8604 // Possible null reference argument.
                    biodata.Add(reader["golongan_darah"].ToString());
                    #pragma warning restore CS8604 // Possible null reference argument.
                    #pragma warning disable CS8604 // Possible null reference argument.
                    biodata.Add(reader["alamat"].ToString());
                    #pragma warning restore CS8604 // Possible null reference argument.
                    #pragma warning disable CS8604 // Possible null reference argument.
                    biodata.Add(reader["agama"].ToString());
                    #pragma warning restore CS8604 // Possible null reference argument.
                    #pragma warning disable CS8604 // Possible null reference argument.
                    biodata.Add(reader["status_perkawinan"].ToString());
                    #pragma warning restore CS8604 // Possible null reference argument.
                }
            }
        }
    }
}

```

```
#pragma warning disable CS8604 // Possible null reference argument.
        biodata.Add(reader["pekerjaan"].ToString());
#pragma warning restore CS8604 // Possible null reference argument.
#pragma warning disable CS8604 // Possible null reference argument.
        biodata.Add(reader["kewarganegaraan"].ToString());
#pragma warning restore CS8604 // Possible null reference argument.
    }
}
}
}
catch (Exception ex)
{
    MessageBox.Show("An error occurred while retrieving biodata: " +
ex.Message);
}

return biodata;
}

}

public class Pair<T1, T2>
{
    public T1 First { get; set; }
    public T2 Second { get; set; }

    public Pair(T1 first, T2 second)
    {
        First = first;
        Second = second;
    }
}
}
```

c. Modul preprocess

i. Public class ImagePreprocess.

Berguna untuk preprocess image, kelas ini mengandung :

- Metode static ConvertToGrayScale yang berfungsi untuk mengubah source gambar bertipe BitmapSource menjadi bentuk grayscalenya.
- Metode static ConvertToASCII berfungsi untuk mengubah tiap baris source gambar menjadi bentuk ASCII nya. Metode ini akan mengembalikan fungsi bertipe List<String>
- Metode static convertToASCIIString berfungsi untuk mengubah gambar menjadi bentuk ASCII nya. Metode ini dipakai untuk gambar yang ada di database.

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Text;
using System.Windows;
```

```

using System.Windows.Controls;
using System.Windows.Media;
using System.Windows.Media.Imaging;

namespace treelilnick.preprocess
{
    public class ImagePreprocess
    {
        public static BitmapSource ConvertToGrayscale(BitmapSource originalImage)
        {
            PixelFormat format = originalImage.Format;
            if (format == PixelFormats.Gray8 || format == PixelFormats.Gray16 || format
== PixelFormats.Indexed8)
            {
                return originalImage;
            }
            int width = originalImage.PixelWidth;
            int height = originalImage.PixelHeight;
            int bytesPerPixel = 4;
            int stride = width * bytesPerPixel;

            byte[] pixels = new byte[height * stride];
            originalImage.CopyPixels(pixels, stride, 0);

            byte[] grayscalePixels = new byte[height * width];

            for (int y = 0; y < height; y++)
            {
                for (int x = 0; x < width; x++)
                {
                    int index = y * stride + x * bytesPerPixel;
                    byte blue = pixels[index];
                    byte green = pixels[index + 1];
                    byte red = pixels[index + 2];

                    byte gray = (byte)(red * 0.3 + green * 0.59 + blue * 0.11);

                    grayscalePixels[y * width + x] = gray;
                }
            }

            return BitmapSource.Create(width, height, originalImage.DpiX,
originalImage.DpiY, PixelFormats.Gray8, null, grayscalePixels, width);
        }
        public static List<string> ConvertToASCII(BitmapSource grayscaleImage)
        {
            List<string> result = [];
            int width = grayscaleImage.PixelWidth;
            int height = grayscaleImage.PixelHeight;
            int paddedWidth = (width + 7) / 8;

            byte[] grayscalePixels = new byte[height * width];
            grayscaleImage.CopyPixels(grayscalePixels, width, 0);

            byte[] binaryPixels = new byte[height * paddedWidth];

            for (int y = 0; y < height; y++)
            {
                for (int x = 0; x < width; x++)
                {
                    int grayscaleIndex = y * width + x;
                    int bitIndex = 7 - (x % 8);

                    byte grayscaleValue = grayscalePixels[grayscaleIndex];
                    byte thresholdValue = grayscaleValue <= 128 ? (byte)1 : (byte)0;
                    binaryPixels[y * paddedWidth + x / 8] |= (byte)(thresholdValue <<
bitIndex);
                }
            }
        }
    }
}

```

```

        for (int i = 0; i < height; i++)
        {
            StringBuilder temp = new();
            for (int j = 0; j < paddedWidth; j++)
            {
                temp.Append((char)binaryPixels[i * paddedWidth + j]);
            }
            result.Add(temp.ToString());
        }
        return result;
    }
}

public static string ConvertToASCIIPattern(BitmapSource grayscaleImage)
{
    int width = grayscaleImage.PixelWidth;
    int height = grayscaleImage.PixelHeight;
    int bitTakes = 40;

    byte[] grayscalePixels = new byte[height * width];
    grayscaleImage.CopyPixels(grayscalePixels, width, 0);

    int minDiff = bitTakes;
    byte[] result = new byte[bitTakes / 8];

    for (int i = height / 4; i < height * 3 / 4; i++)
    {
        // skip first two
        int j = 0;
        int whiteCnt = 0;
        int blackCnt = 0;
        // skip if black count < 3
        for (; j < width - bitTakes; j += 8)
        {
            int blk = 0;
            for (int k = 0; k < 8; k++)
            {
                if (grayscalePixels[i * width + j + k] <= 128) blk++;
            }
            if (blk > 2) break;
        }
        // pick 32 bit
        byte[] currBit = new byte[bitTakes / 8];
        for (int k = 0; k < bitTakes; k++)
        {
            int val;
            if (j + k >= width)
            {
                blackCnt++;
                val = 1;
            }
            else if (grayscalePixels[i * width + j + k] <= 128)
            {
                blackCnt++;
                val = 1;
            }
            else
            {
                whiteCnt++;
                val = 0;
            }
            int bitIdx = 7 - k % 8;
            currBit[k / 8] |= (byte)(val << bitIdx);
        }
        int diff = Math.Abs(whiteCnt - blackCnt);
        if (diff < minDiff)
        {
            minDiff = diff;
            for (int k = 0; k < bitTakes / 8; k++)
            {
                result[k] = currBit[k];
            }
        }
    }
}

```



```

    }
    }
    }
    StringBuilder res = new();
    for (int i = 0; i < bitTakes / 8; i++)
    {
        res.Append((char)result[i]);
    }
    return res.ToString();
}

public static string ConvertToASCIIString(BitmapSource grayscaleImage)
{
    int width = grayscaleImage.PixelWidth;
    int height = grayscaleImage.PixelHeight;
    int paddedWidth = (width + 7) / 8;

    byte[] grayscalePixels = new byte[height * width];
    grayscaleImage.CopyPixels(grayscalePixels, width, 0);

    byte[] binaryPixels = new byte[height * paddedWidth];

    for (int y = 0; y < height; y++)
    {
        for (int x = 0; x < width; x++)
        {
            int grayscaleIndex = y * width + x;
            int bitIndex = 7 - (x % 8);

            byte grayscaleValue = grayscalePixels[grayscaleIndex];
            byte thresholdValue = grayscaleValue <= 128 ? (byte)1 : (byte)0;
            binaryPixels[y * paddedWidth + x / 8] |= (byte)(thresholdValue <<
bitIndex);
        }
    }
    StringBuilder res = new();
    for (int i = 0; i < height; i++)
    {
        for (int j = 0; j < paddedWidth; j++)
        {
            res.Append((char)binaryPixels[i * paddedWidth + j]);
        }
    }
    return res.ToString();
}
}
}

```

ii. Public class KMP

Berguna untuk *string matching* dengan algoritma KMP, kelas ini mengandung

- Metode static `ComputeLps` yaitu metode yang bertugas untuk mencari array `b(k)` dari pattern yang dicari, yaitu String hasil konversi dari gambar masukan user. Metode ini mengembalikan `List<int>`
- Metode static `FindPattern` yang mengimplementasikan algoritma KMP antara pattern dan text. Metode ini mengembalikan boolean untuk menandakan apakah pattern terdapat di text.

```

public class KMP
{
    public static List<int> ComputeLps(string pattern)
    {
        List<int> lps = [];
        int n = pattern.Length;
        for (int idx = 0; idx < n; idx++)
        {
            lps.Add(0);
        }
        int i = 1, j = 0;
        while (i < n)
        {
            if (pattern[i] == pattern[j])
            {
                lps[i] = j + 1;
                i++;
                j++;
            }
            else if (j > 0)
            {
                j = lps[j - 1];
            }
            else
            {
                i++;
            }
        }
        return lps;
    }
    public static bool FindPattern(string text, string pattern)
    {
        List<int> lps = ComputeLps(pattern);
        int n = text.Length;
        int m = pattern.Length;
        int i = 0;
        int j = 0;

        while (i < n)
        {
            if (text[i] == pattern[j])
            {
                if (j == m - 1)
                {
                    return true;
                }
                i++;
                j++;
            }
            else if (j > 0)
            {
                j = lps[j - 1];
            }
            else
            {
                i++;
            }
        }
        return false;
    }
}

```

iii. Public class BayerMoore

Berguna untuk *string matching* dengan algoritma BM, kelas ini mengandung :

- Metode static BuildLast yang berguna untuk mencari nilai index kemunculan terakhir dari tiap karakter pada pattern.
- Metode static FindPattern yang mengimplementasikan algoritma KMP antara pattern dan text. Metode ini mengembalikan boolean untuk menandakan apakah pattern terdapat di text.

```
public class BayerMoore
{
    public static List<int> BuildLast(String pattern)
    {
        List<int> last = [];
        for (int i = 0; i < 256; i++)
        {
            last.Add(-1);
        }
        for (int i = 0; i < pattern.Length; i++)
        {
            last[pattern[i]] = i;
        }
        return last;
    }
    public static bool FindPattern(string text, string pattern)
    {
        List<int> last = BuildLast(pattern);
        int n = text.Length;
        int m = pattern.Length;
        int i = m - 1;

        if (i > n - 1)
        {
            return false;
        }

        int j = m - 1;
        do
        {
            if (text[i] == pattern[j]) {
                if (j == 0)
                {
                    return true;
                }
            }
            else
            {
                i--;
                j--;
            }
        }
        else
        {
            int lo = last[text[i]];
            i = i + m - Math.Min(j, 1 + lo);
            j = m - 1;
        }
    } while (i < n);

    return false;
}
}
```

iv. Public class ClosestString

Berguna untuk mencari nilai kecocokan String pattern dengan teks. Kelas ini mengandung :

- Metode HammingDistance yang berguna untuk mencari jumlah hamming distance dari dua string.

```
public class ClosestString
{
    public static int HammingDistance(string s1, string s2) {
        int len = Math.Min(s1.Length, s2.Length);
        int count = 0;
        for (int i = 0; i < len; i++) {
            if (s1[i] != s2[i]) {
                count++;
            }
        }
        return count;
    }
}
```

d. Modul imageLoader

i. Public class ImageLoader

Berfungsi untuk me-load image. Kelas ini mengandung :

- Metode static LoadImage yang berfungsi mengkonversi path berkas citra menjadi data bertipe BitmapImage.

```
using System.IO;
using System.Windows.Media.Imaging;

namespace treelilnick.imageloader
{
    public class ImageLoader
    {
        public static BitmapSource LoadImage(string filePath)
        {
            // Create a BitmapImage and set its properties
            Uri imgUri = new Uri(filePath, UriKind.Relative);
            BmpBitmapDecoder decoder2 = new BmpBitmapDecoder(imgUri,
                BitmapCreateOptions.PreservePixelFormat, BitmapCacheOption.Default);
            BitmapSource bitmapSource2 = decoder2.Frames[0];
            return bitmapSource2;
        }
    }
}
```

e. Modul Regex

i. Public class AlayRegex.

Berfungsi untuk meng-handle regex pada aplikasi. Regex digunakan untuk mencari nama alay ketika sudah diketahui nama asli. Kelas ini mengandung :

- Metode static CreateAlayRegex yang akan mengubah parameter nama asli menjadi regex untuk mencari nama alaynya.
- Metode static SearchAlay yang dengan parameter string nama asli dan List<Pair<string, string>> nama alay dan NIK. Metode ini akan mengembalikan NIK yang sesuai dengan nama asli atau men-throw bahwa nama tidak *match* jika tidak ada nama yang sesuai.

```
using System;
using System.Collections.Generic;
using System.Text.RegularExpressions;
using treelilnick.connector;

namespace treelilnick.regex
{
    public class AlayRegex
    {
        // Generates a regex pattern to match an "alay" version of the provided name
        private static string CreateAlayRegex(string realName)
        {
            // Dictionary for character substitutions with regex patterns
            var substitution = new Dictionary<char, string>
            {
                {'a', "[a4]?"},
                {'e', "[e3]?"},
                {'i', "[i1]?"},
                {'o', "[o0]?"},
                {'u', "[u]?"},
                {'b', "[b68]?"},
                {'g', "[g69]?"},
                {'j', "[7j]?"},
                {'l', "[l1]?"},
                {'s', "[s5]?"},
                {'z', "[z2]?"},
                {' ', "\\s+"}
            };

            // Build regex pattern by substituting each character in the real name
            string regexPattern = "";
            foreach (char c in realName.ToLower())
            {
                if (substitution.ContainsKey(c))
                {
                    regexPattern += substitution[c];
                }
                else
                {
                    regexPattern += c;
                }
            }
        }
    }
}
```

```

        }
    }

    return regexPattern;
}

// Searches for a name in the list that matches the "alay" version of the
provided real name
public static Pair<string, string> SearchAlay(string realName, List<Pair<string,
string>> listAlay)
{
    // Create regex pattern from real name
    Regex regex = new Regex("^" + CreateAlayRegex(realName) + "$",
RegexOptions.IgnoreCase);

    // Search for a matching name in the list
    foreach (Pair<string, string> nameAndNIK in listAlay)
    {
        // Return the matching name and NIK
        if (regex.IsMatch(nameAndNIK.First))
        {
            return nameAndNIK;
        }
    }

    // Throw an exception if no match is found
    throw new Exception("No name match");
}
}
}

```

f. Modul Encryptor

i. Public class Decrypt

Berfungsi untuk mendekripsi data-data pada database.

- Atribut static string charlist menyimpan daftar karakter yang valid
- Atribut static Dictionary <char, int> charordo menyimpan map ke dari char ke int
- Atribut static Dictionary <int, char> charnum menyimpan map dari int ke char
- Konstruktor Decrypt berguna untuk konstruktor decryptor
- Method static DecrypCipher akan mendekripsi string menjadi string hasil dekripsi

```

using System;
using System.Collections.Generic;
using System.Text;

namespace treelilnick.decryptor
{
    public class Decrypt
    {
        private static readonly string charlist =
"ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789!@#%^&*()_.-=<>?/ ";
        private static readonly Dictionary<char, int> charordo;
        private static readonly Dictionary<int, char> charnum;
        private static readonly int length;
    }
}

```

```
static Decrypt()
{
    charordo = new Dictionary<char, int>();
    charnum = new Dictionary<int, char>();

    // Populate the dictionaries
    for (int i = 0; i < charlist.Length; i++)
    {
        charordo[charlist[i]] = i;
        charnum[i] = charlist[i];
    }

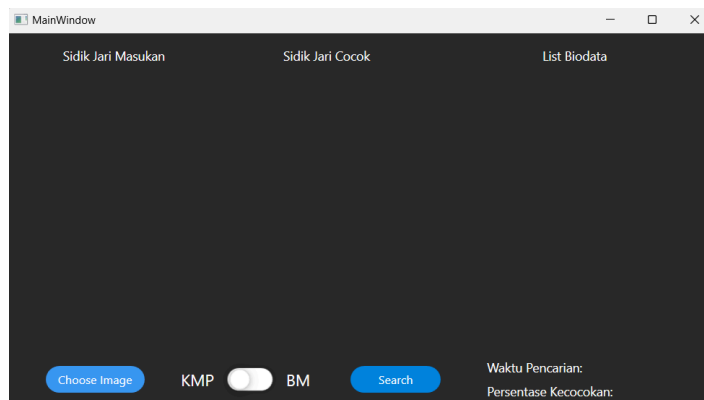
    length = charlist.Length;
}

public static string DecryptCipher(string cipher) {
    StringBuilder cstring = new StringBuilder(cipher);
    int size = cstring.Length;
    for (int i = 0; i < size; i++) {
        int coded = charordo[cstring[i]] - size;
        if (coded < 0) coded += length;
        cstring[i] = charnum[coded];
    }
    return cstring.ToString();
}
}
```

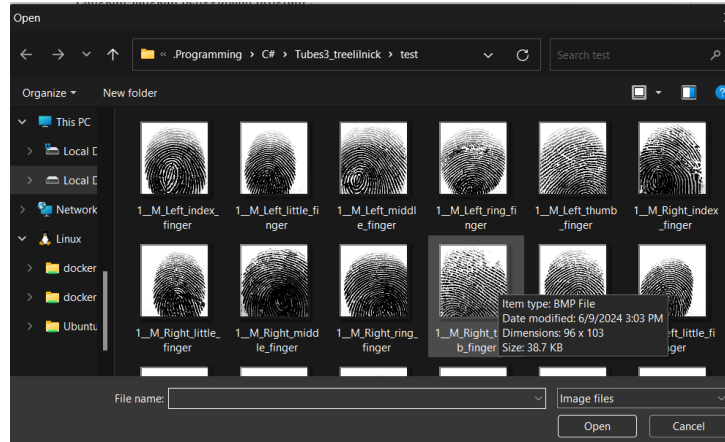
4.2 Penjelasan Tata Cara Penggunaan Program

Langkah-langkah penggunaan program :

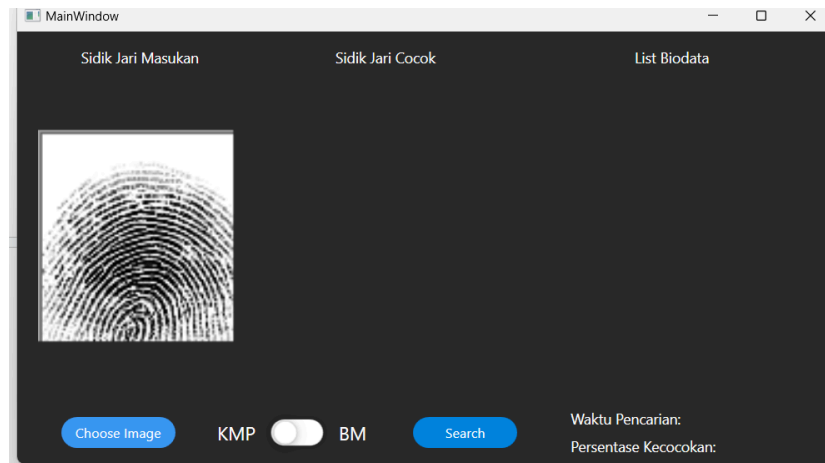
1. Memasukkan citra sidik jari yang ingin dicari biodata pemiliknya.



Gambar 4.2.1 Tampilan awal aplikasi



Gambar 4.2.2 Tampilan saat memilih gambar akan diarahkan ke eksplorer device



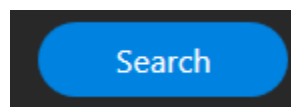
Gambar 4.2.3 Tampilan ketika gambar sudah dipilih

2. Memilih opsi algoritma pencarian (KMP atau BM)



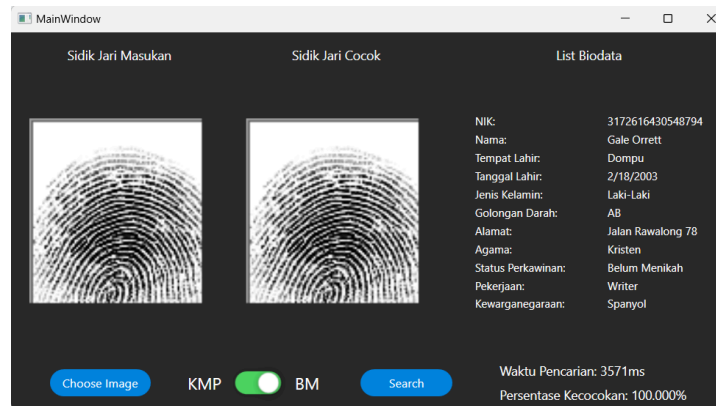
Gambar 4.2.4 Toggle untuk memilih algoritma

3. Menekan tombol search, program kemudian memproses untuk mencari citra sidik jari yang sesuai.

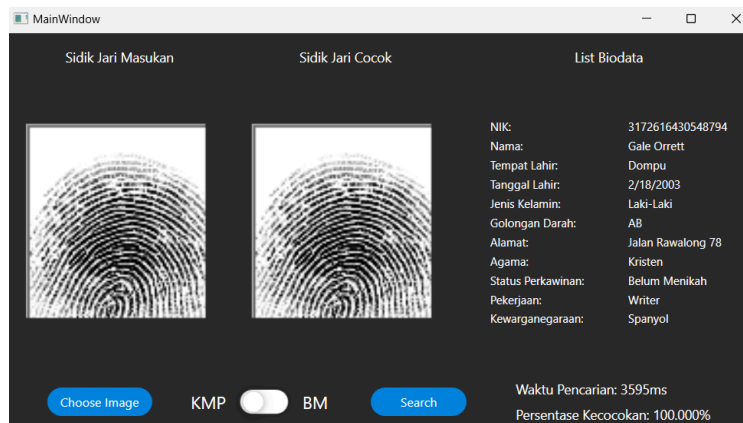


Gambar 4.2.5 Tombol untuk mencari biodata

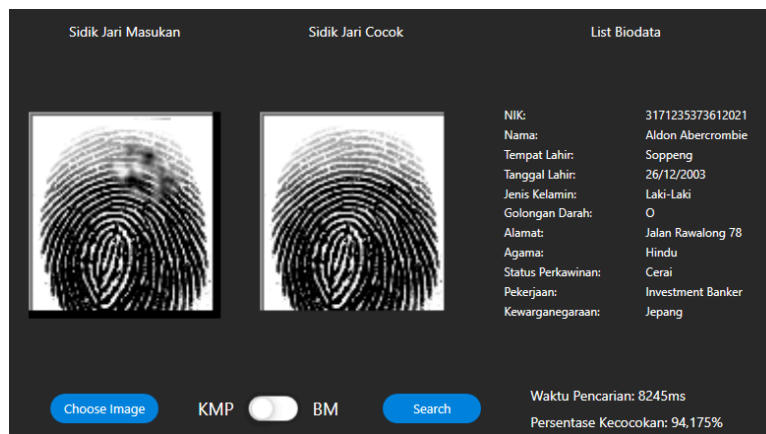
4. Program akan menampilkan biodata jika ditemukan citra sidik jari yang tepat mirip. Jika tidak ada yang tepat mirip, program akan menampilkan citra yang persentase kemiripannya paling tinggi dengan menggunakan *hamming distance*.



Gambar 4.2.6 Contoh hasil pencarian dengan algoritma BM



Gambar 4.2.7 Contoh hasil pencarian dengan algoritma KMP

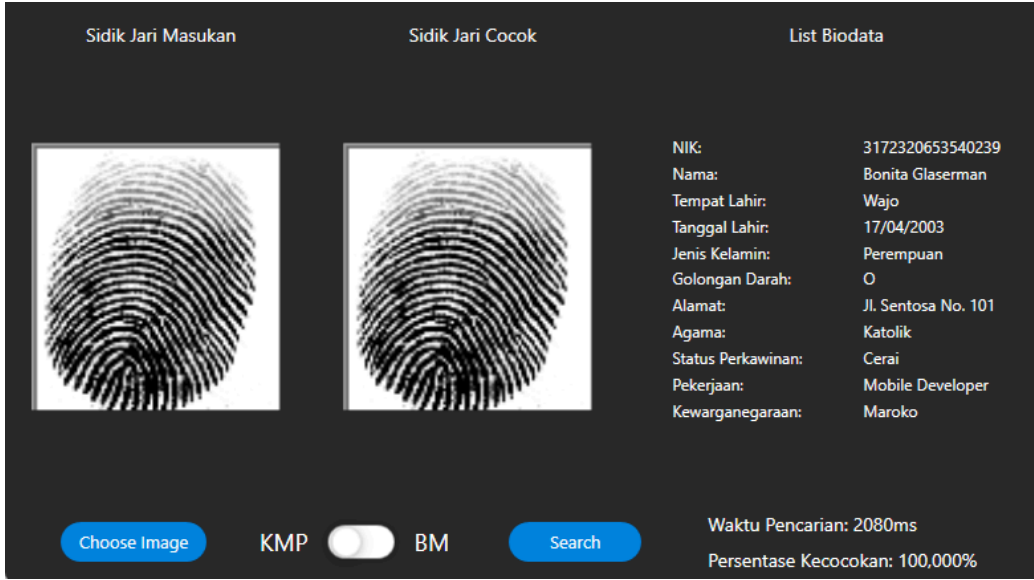
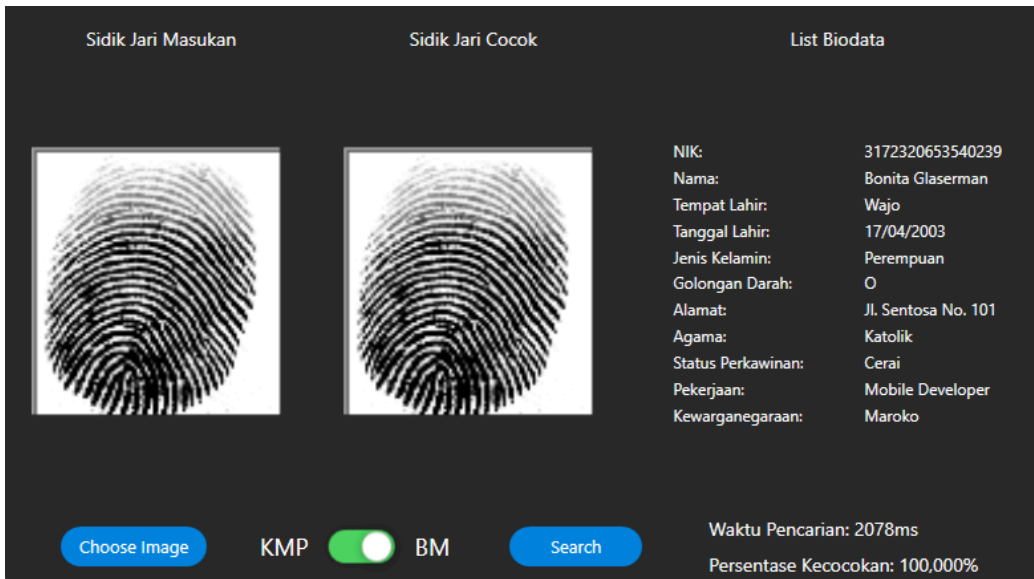


Gambar 4.2.8 Contoh hasil pencarian yang tidak *exact match*





4.3 Hasil Pengujian

Berikut adalah beberapa pengujian yang dilakukan terhadap program ini.





A. Pengujian gambar sidik jari 1 yang tidak *altered*

Algoritma	Hasil Pengujian
KMP	
BM	





B. Pengujian gambar sidik jari 2 yang tidak *altered*

Algoritma	Hasil Pengujian
KMP	<div> <div>Sidik Jari Masukan</div> <div>Sidik Jari Cocok</div> <div>List Biodata</div> <div>   <div> <div>NIK:</div>3174067338917495 <div>Nama:</div>Gustavo Hedger <div>Tempat Lahir:</div>Ambon <div>Tanggal Lahir:</div>26/06/2000 <div>Jenis Kelamin:</div>Perempuan <div>Golongan Darah:</div>B <div>Alamat:</div>Jl. Kencana No. 777 <div>Agama:</div>Buddha <div>Status Perkawinan:</div>Cerai <div>Pekerjaan:</div>Politician <div>Kewarganegaraan:</div>Indonesia </div> <div> <div>Choose Image</div> <div>KMP <input type="checkbox"/></div> <div>BM <input checked="" type="checkbox"/></div> <div>Search</div> <div>Waktu Pencarian: 3924ms</div> <div>Persentase Kecocokan: 100,000%</div> </div> </div> </div>
BM	<div> <div>Sidik Jari Masukan</div> <div>Sidik Jari Cocok</div> <div>List Biodata</div> <div>   <div> <div>NIK:</div>3174067338917495 <div>Nama:</div>Gustavo Hedger <div>Tempat Lahir:</div>Ambon <div>Tanggal Lahir:</div>26/06/2000 <div>Jenis Kelamin:</div>Perempuan <div>Golongan Darah:</div>B <div>Alamat:</div>Jl. Kencana No. 777 <div>Agama:</div>Buddha <div>Status Perkawinan:</div>Cerai <div>Pekerjaan:</div>Politician <div>Kewarganegaraan:</div>Indonesia </div> <div> <div>Choose Image</div> <div>KMP <input type="checkbox"/></div> <div>BM <input checked="" type="checkbox"/></div> <div>Search</div> <div>Waktu Pencarian: 4802ms</div> <div>Persentase Kecocokan: 100,000%</div> </div> </div> </div>

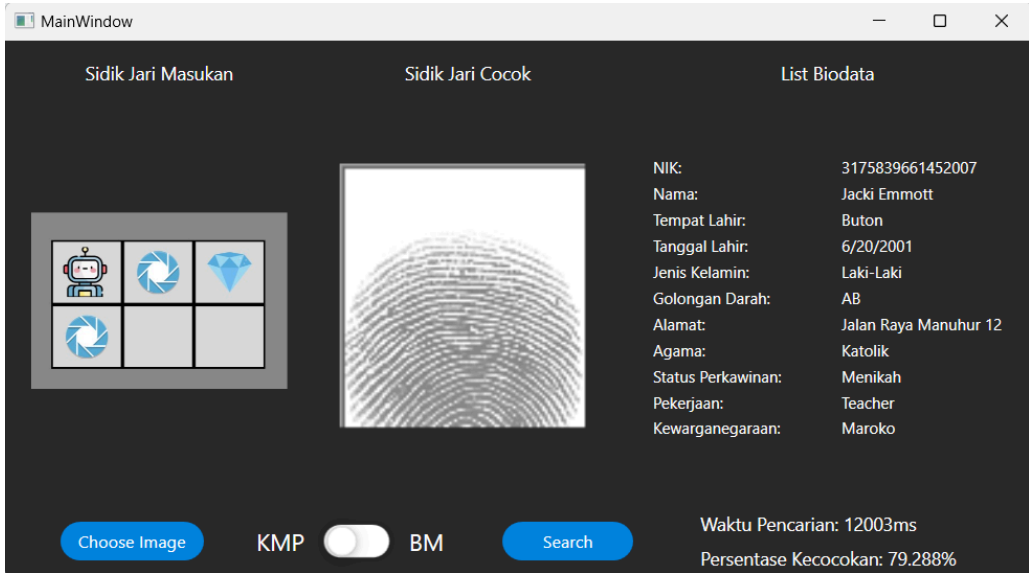
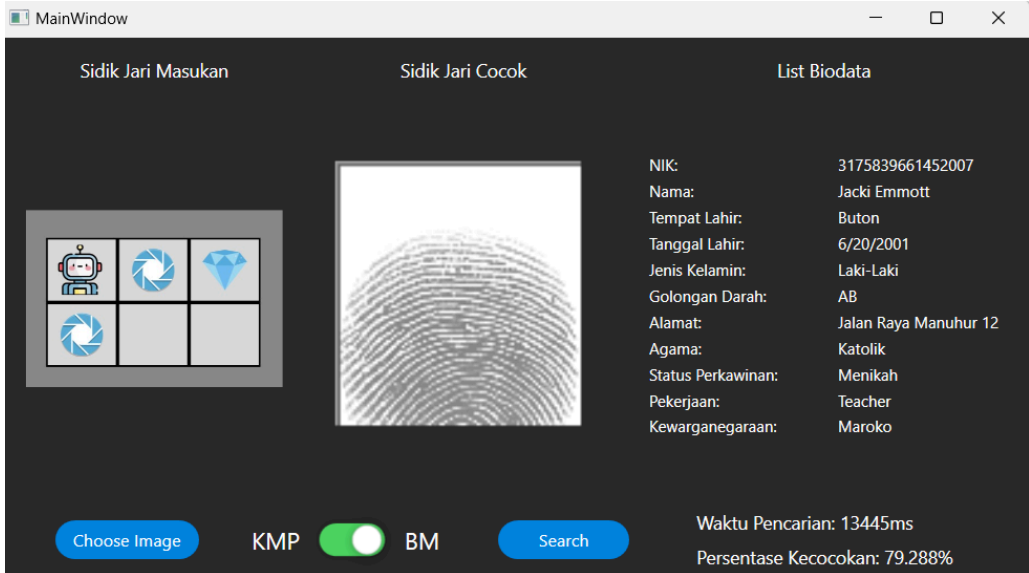
C. Pengujian gambar sidik jari 3 yang *altered*

Algoritma	Hasil Pengujian
KMP	<div> <div>Sidik Jari Masukan</div> <div>Sidik Jari Cocok</div> <div>List Biodata</div> <div>   <div> <div>NIK:</div>3171235373612021 <div>Nama:</div>Aldon Abercrombie <div>Tempat Lahir:</div>Soppeng <div>Tanggal Lahir:</div>26/12/2003 <div>Jenis Kelamin:</div>Laki-Laki <div>Golongan Darah:</div>O <div>Alamat:</div>Jalan Rawalong 78 <div>Agama:</div>Hindu <div>Status Perkawinan:</div>Cerai <div>Pekerjaan:</div>Investment Banker <div>Kewarganegaraan:</div>Jepang </div> <div> <div>Choose Image</div> <div>KMP <input type="checkbox"/></div> <div>BM <input type="checkbox"/></div> <div>Search</div> <div>Waktu Pencarian: 8245ms</div> <div>Persentase Kecocokan: 94,175%</div> </div> </div> </div>
BM	<div> <div>Sidik Jari Masukan</div> <div>Sidik Jari Cocok</div> <div>List Biodata</div> <div>   <div> <div>NIK:</div>3171235373612021 <div>Nama:</div>Aldon Abercrombie <div>Tempat Lahir:</div>Soppeng <div>Tanggal Lahir:</div>26/12/2003 <div>Jenis Kelamin:</div>Laki-Laki <div>Golongan Darah:</div>O <div>Alamat:</div>Jalan Rawalong 78 <div>Agama:</div>Hindu <div>Status Perkawinan:</div>Cerai <div>Pekerjaan:</div>Investment Banker <div>Kewarganegaraan:</div>Jepang </div> <div> <div>Choose Image</div> <div>KMP <input type="checkbox"/></div> <div>BM <input checked="" type="checkbox"/></div> <div>Search</div> <div>Waktu Pencarian: 8286ms</div> <div>Persentase Kecocokan: 94,175%</div> </div> </div> </div>

D. Pengujian gambar sidik jari 4 yang *altered*

Algoritma	Hasil Pengujian
KMP	<div> <div>Sidik Jari Masukan</div> <div>Sidik Jari Cocok</div> <div>List Biodata</div> <div>   <div> <div>NIK: 3173579109202365</div> <div>Nama: Sigismund Heinz</div> <div>Tempat Lahir: Parepare</div> <div>Tanggal Lahir: 29/01/2002</div> <div>Jenis Kelamin: Laki-Laki</div> <div>Golongan Darah: O</div> <div>Alamat: Jl. Ceria No. 321</div> <div>Agama: Kristen</div> <div>Status Perkawinan: Menikah</div> <div>Pekerjaan: Teacher</div> <div>Kewarganegaraan: Maroko</div> </div> </div> <div> <div>Choose Image</div> <div>KMP <input type="checkbox"/></div> <div>BM <input checked="" type="checkbox"/></div> <div>Search</div> <div>Waktu Pencarian: 8052ms</div> <div>Persentase Kecocokan: 69,256%</div> </div> </div>
BM	<div> <div>Sidik Jari Masukan</div> <div>Sidik Jari Cocok</div> <div>List Biodata</div> <div>   <div> <div>NIK: 3173579109202365</div> <div>Nama: Sigismund Heinz</div> <div>Tempat Lahir: Parepare</div> <div>Tanggal Lahir: 29/01/2002</div> <div>Jenis Kelamin: Laki-Laki</div> <div>Golongan Darah: O</div> <div>Alamat: Jl. Ceria No. 321</div> <div>Agama: Kristen</div> <div>Status Perkawinan: Menikah</div> <div>Pekerjaan: Teacher</div> <div>Kewarganegaraan: Maroko</div> </div> </div> <div> <div>Choose Image</div> <div>KMP <input type="checkbox"/></div> <div>BM <input checked="" type="checkbox"/></div> <div>Search</div> <div>Waktu Pencarian: 8217ms</div> <div>Persentase Kecocokan: 69,256%</div> </div> </div>

E. Pengujian untuk gambar yang tidak terkait

Algoritma	Hasil Pengujian
KMP	 <p>MainWindow</p> <p>Sidik Jari Masukan Sidik Jari Cocok List Biodata</p> <p>NIK: 3175839661452007 Nama: Jacki Emmott Tempat Lahir: Buton Tanggal Lahir: 6/20/2001 Jenis Kelamin: Laki-Laki Golongan Darah: AB Alamat: Jalan Raya Manuhur 12 Agama: Katolik Status Perkawinan: Menikah Pekerjaan: Teacher Kewarganegaraan: Maroko</p> <p>Choose Image KMP <input type="checkbox"/> BM <input checked="" type="checkbox"/> Search Waktu Pencarian: 12003ms Persentase Kecocokan: 79.288%</p>
BM	 <p>MainWindow</p> <p>Sidik Jari Masukan Sidik Jari Cocok List Biodata</p> <p>NIK: 3175839661452007 Nama: Jacki Emmott Tempat Lahir: Buton Tanggal Lahir: 6/20/2001 Jenis Kelamin: Laki-Laki Golongan Darah: AB Alamat: Jalan Raya Manuhur 12 Agama: Katolik Status Perkawinan: Menikah Pekerjaan: Teacher Kewarganegaraan: Maroko</p> <p>Choose Image KMP <input type="checkbox"/> BM <input checked="" type="checkbox"/> Search Waktu Pencarian: 13445ms Persentase Kecocokan: 79.288%</p>

4.4 Analisis Hasil Pengujian

Berdasarkan hasil pengujian di atas, yaitu pada uji bagian (A) dan (B), dapat dilihat bahwa waktu yang dibutuhkan untuk pencarian pada algoritma KMP dan BM relatif sama. Hal ini menunjukkan bahwa kompleksitas waktu yang pada kedua algoritma juga seharusnya relatif sama. Kedua algoritma tersebut juga menunjukkan hasil gambar yang sama. Hal ini merupakan reasuransi bahwa kedua algoritma bekerja dengan cukup baik.

Pada pengujian dengan *altered image*, yaitu pada uji bagian (C) dan (D), persentase yang didapat tidak 100%. Hal ini berarti algoritma KMP dan BM tidak menemukan gambar yang sesuai dengan sidik jari pencarian sehingga perlu digunakan *hamming distance* untuk melakukan perbandingan antara *string* ASCII-8 bits gambar pencarian dengan setiap gambar yang ada di database. Berdasarkan pengujian tersebut, hasil gambar sidik jari yang didapatkan sudah cukup akurat.

Sementara itu, pada pengujian gambar yang bukan sidik jari(d), ternyata persentase kecocokannya masih tinggi. Hal ini karena gambar pada masukan dan database cukup terang sehingga dikategorikan sebagai binary 0. Oleh karena itu, algoritma *hamming distance* akan menganggap bahwa kedua gambar cukup mirip.

BAB V

Analisis

5.1 Kesimpulan

Tugas Besar 3 IF2211 Strategi Algoritma adalah tugas untuk mencari sidik jari pada database yang sesuai dengan sidik jari masukan. Untuk menyelesaikan permasalahan ini, algoritma pencocokan string, yaitu KMP dan BM dapat menjadi suatu solusi. Aplikasi pencarian sidik jari ini berbasis desktop. Bahasa pemrograman yang dipakai adalah C# dengan memanfaatkan framework dotnet. Kerangka kerja UI yang dipakai adalah “WPF .NET”.

Dari semua yang kami kerjakan di Tugas Besar ini, kami bisa menyimpulkan bahwa algoritma KMP dan BM dapat digunakan untuk menyelesaikan pencarian sidik jari jika memang bagian dari sidik jari masukan terkandung di dalam sidik jari yang ada di database. Jika ada gambar yang *altered*, maka algoritma *hamming distance* merupakan alternatif yang baik untuk menentukan persentase kemiripan dua sidik jari. Namun, jika data masukan bukan merupakan gambar sidik jari, program tidak bisa mendeteksinya, karena hanya melihat gelap terang sehingga bisa salah interpretasi

5.2 Saran

Penggunaan *String Matching* untuk mendeteksi citra nampaknya tidak bisa digunakan begitu saja. Penggunaannya akan cukup baik jika memang citra masukan sudah ada di database karena algoritma ini dapat mendeteksinya dengan cukup cepat setelah citra diubah menjadi ASCII 8-bit. Namun, untuk citra yang bukan sidik jari, program tetap akan menganggap citra tersebut sama dengan yang ada di database karena hanya mengikuti gelap terang.

5.3 Refleksi

Algoritma *string matching* seperti Knuth-Morris-Pratt (KMP) dan Boyer-Moore (BM) adalah algoritma yang memiliki banyak sekali kegunaan seperti di bidang bioinformatika, analisis citra, *web search engine*, dan banyak lainnya. Salah satu pemanfaatan lain dari algoritma ini adalah untuk *fingerprint matching* seperti pada tugas ini. Hasil pencarian yang ditunjukkan oleh algoritma ini juga cukup baik. Walaupun demikian, algoritma ini tidak akan berjalan dengan baik jika terdapat rotasi atau *resize* terhadap gambar sidik jari yang ingin dicari.

DAFTAR PUSTAKA

Rinaldi Munir. Pencocokan String (String/Pattern Matching)

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

Rinaldi Munir. String Matching dengan Regular Expression

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

Rinaldi Munir. Aplikasi Teori Bilangan

<https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2023-2024/16-Teori-Bilangan-Bagian3-2023.pd> Digunakan untuk Dekripsi Caesar Cipher

Supatmi, S., and I. D. Sumitra. "Fingerprint Identification using Bozorth and BoyerMoore Algorithm." IOP Conference Series: Materials Science and Engineering, vol. 662, 2019, p. 022040. IOP Publishing, .

<https://iopscience.iop.org/article/10.1088/1757-899X/662/2/022040>

Proses identifikasi individu melalui citra sidik jari.

“Sokoto Coventry Fingerprint Dataset (SOCOFing)”

<https://www.kaggle.com/datasets/ruizgara/socofing> Dataset fingerprint yang bisa digunakan.

“Bahasa Alay Generator” <https://alaygenerator.blogspot.com/> Melakukan simulasi konversi bahasa alay Indonesia.

“How do I use SQL Server with C# and .NET?”

<https://learn.microsoft.com/en-us/shows/on-net/how-do-i-use-sql-server-with-csharp-anddotnet>

Cara membuat koneksi SQL dengan C#.

“Basic Database Operations Using C#”

<https://www.geeksforgeeks.org/basic-database-operations-using-c-sharp/> Operasi basis data dasar menggunakan C#.

“C# Documentation” <https://learn.microsoft.com/id-id/dotnet/csharp/> Dokumentasi C#

LAMPIRAN

Link Repository: https://github.com/sotul04/Tubes3_treelilnick

Link Video: <https://youtu.be/IyaAkxIMOzk>

Check List:

Poin	Ya	Tidak
Program berhasil dijalankan	✓	
Program dapat menampilkan data individu berdasarkan sidik jari menggunakan algoritma Knuth-Morris-Pratt (KMP)	✓	
Program dapat menampilkan data individu berdasarkan sidik jari menggunakan algoritma Boyer-Moore (KMP)	✓	
Program dapat menampilkan data individu berdasarkan sidik jari menggunakan <i>hamming distance</i>	✓	
[Bonus] Melakukan enkripsi terhadap data dalam basis data	✓	
[Bonus] Membuat video kelompok	✓	

Pembagian Tugas:

Nama	Nim	Pembagian Tugas
Adril Putra Merin	13522068	Image Loader, Image Preprocessing, KMP Algorithm, BM Algorithm, Perekaman Video
Suthasoma Mahardhika Munthe	13522098	Setup Project and Database, Seeding Database, Enkripsi dan Dekripsi database, GUI design and implementation
Berto Richardo Togatorop	13522118	Pencarian data dengan regex, Laporan, Seeding Database, Rekaman Video