

LAPORAN TUGAS BESAR 3

IF2211 Strategi Algoritma

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



Disusun oleh:

10023478 Karunia Syukur Baeha

13522088 Muhamad Rafli Rasyiidin

13522101 Abdullah Mubarak

13522105 Fabian Radenta Bangun

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA

INSTITUT TEKNOLOGI BANDUNG

2024

Daftar Isi

BAB I.....	2
DESKRIPSI TUGAS.....	2
1.1 Deskripsi Tugas.....	2
Penggunaan Program.....	5
BAB II.....	7
LANDASAN TEORI.....	7
2.1 Pattern Matching.....	7
2.2 Algoritma Knuth-Morris-Pratt (KMP).....	7
2.3 Algoritma Boyer-Moore (BM).....	8
2.4 Hamming Distance.....	8
2.5 Regular Expression.....	8
2.6 Pengembangan Aplikasi Desktop dalam Bahasa C# Menggunakan Ekosistem .NET dan Framework GUI WPF.....	11
BAB III.....	12
IMPLEMENTASI DAN PENGUJIAN.....	12
3.1 Langkah-Langkah Umum Pemecahan Masalah.....	12
3.2 Proses Penyelesaian Masalah dengan Algoritma KMP.....	13
3.3 Proses Penyelesaian Masalah dengan Algoritma BM.....	13
3.4 Arsitektur Aplikasi dan Fitur Fungsionalnya.....	14
3.5 Ilustrasi Kasus.....	14
BAB IV.....	16
IMPLEMENTASI DAN PENGUJIAN.....	16
4.1 Spesifikasi Teknis Program.....	16
4.2 Tata Cara Penggunaan Program.....	24
4.3 Hasil Pengujian.....	24
4.2 Analisis Hasil Pengujian.....	25
BAB V.....	26
KESIMPULAN, SARAN, TANGGAPAN, DAN REFLEKSI.....	26
5.1 Kesimpulan.....	26
5.2 Saran.....	26
5.3 Tanggapan.....	26
5.4 Refleksi.....	26
LAMPIRAN.....	28
DAFTAR PUSTAKA.....	29

BAB I

DESKRIPSI TUGAS

1.1 Deskripsi Tugas

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.

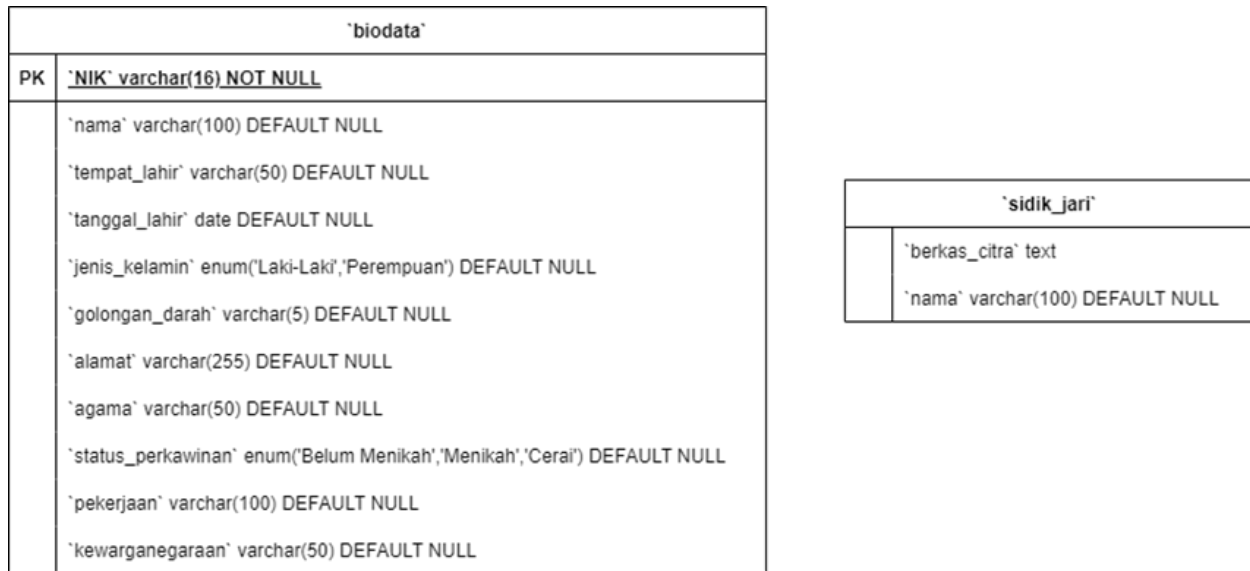


Gambar 2. Skema implementasi konversi citra sidik jari.

Sumber: Dokumentasi Penulis

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.



Gambar 3. Skema basis data yang digunakan.

Sumber: Dokumentasi Penulis

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 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 (silakan *review* kembali materi IF2240 bagian basis data relasional). Pada tugas besar kali ini, kita akan coba melakukan simulasi 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.

Sebuah data yang korupsi dapat memiliki berbagai macam bentuk. Pada tugas ini, jenis data korupsi 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

Tabel 2. Kombinasi ketiga variasi bahasa alay Indonesia.

Sumber: Dokumentasi Penulis

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.

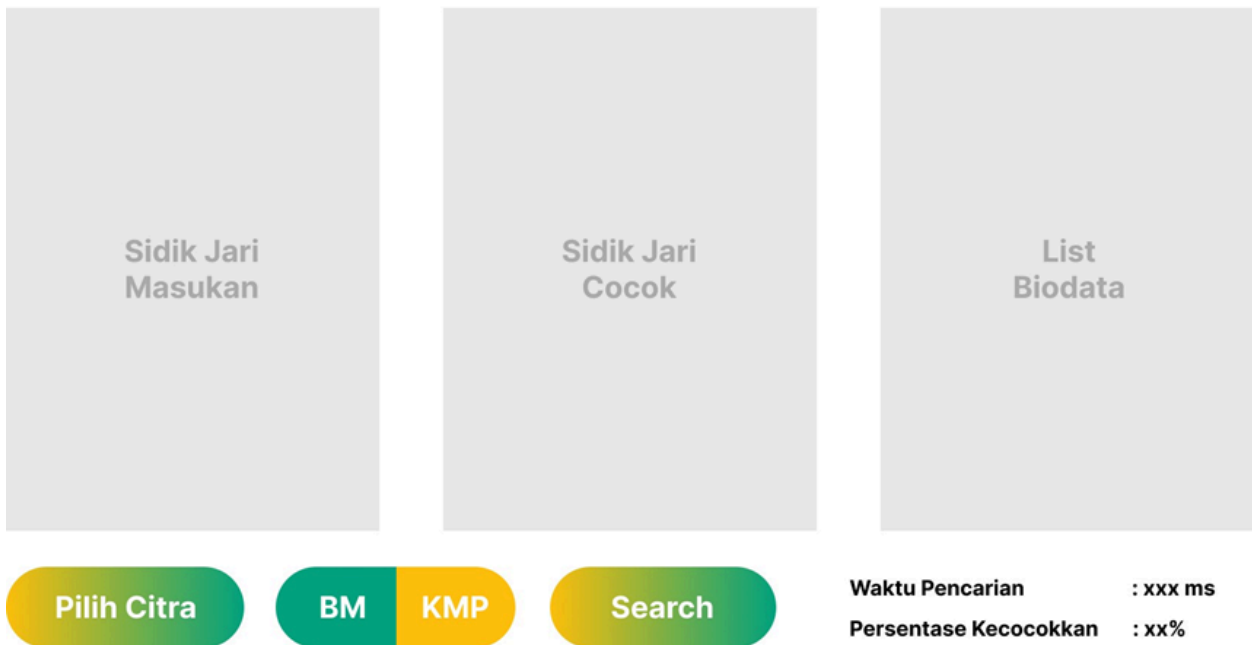
Setelah menemukan nama yang bersesuaian, Anda dapat menggunakan basis data untuk mengembalikan detail biodata lengkap individu. Jika seluruh prosedur diatas diimplementasikan dengan baik, maka sebuah sistem deteksi individu berbasis biometrik dengan sidik jari telah berhasil untuk diimplementasikan.

Penggunaan Program

Pada Tugas Besar kali ini, sistem yang dibangun akan diimplementasikan dengan basis *desktop-app* menggunakan **bahasa C#**. Masukan yang akan diberikan oleh pengguna saat menggunakan aplikasi adalah **sebuah citra sidik jari**. Selain itu program perlu untuk **memiliki basis data SQL** dengan struktur relasional seperti yang telah dijelaskan

sebelumnya. Tampilan layout dari aplikasi yang akan dibangun adalah sebagai berikut.

Aplikasi C# Tugas Besar 3 Strategi Algoritma 2023/2024



Gambar 4. Referensi tampilan UI *desktop-app*.

Sumber: Dokumentasi Penulis

BAB II

LANDASAN TEORI

2.1 Pattern Matching

Pattern Matching adalah metode dalam ilmu komputer yang digunakan untuk mengidentifikasi urutan tertentu dalam data yang sesuai dengan pola yang telah ditentukan. Teknik ini memiliki peran krusial dalam berbagai aplikasi seperti pengenalan teks, analisis data, pemrosesan bahasa alami, dan pencarian dalam basis data. Beberapa algoritma yang umum digunakan untuk Pattern Matching meliputi Knuth-Morris-Pratt (KMP), Boyer-Moore, dan Rabin-Karp, yang masing-masing menawarkan pendekatan unik untuk meningkatkan efisiensi dan kecepatan pencarian.

Dalam praktiknya, Pattern Matching memfasilitasi proses pencarian string dalam editor teks, mesin pencari, dan basis data, serta dalam pengenalan teks dan pemrosesan bahasa alami. Tantangan utama dalam Pattern Matching termasuk menangani ukuran data yang besar dan variasi dalam pola yang disebabkan oleh kesalahan atau mutasi. Implementasi algoritma pencocokan pola yang efisien sangat penting untuk mengatasi tantangan ini dan memastikan performa optimal dalam berbagai skenario penggunaan.

2.2 Algoritma Knuth-Morris-Pratt (KMP)

Algoritma Knuth-Morris-Pratt (KMP) adalah salah satu algoritma yang efisien untuk mencari substring dalam string utama. Algoritma ini dikembangkan oleh Donald Knuth, Vaughan Pratt, dan James H. Morris, dan dipublikasikan pertama kali pada tahun 1977. KMP bertujuan untuk mengoptimalkan proses pencarian dengan memanfaatkan informasi dari pola yang sudah diketahui untuk menghindari pencarian ulang yang tidak perlu. Algoritma ini menggunakan *border table* yang membantu menentukan posisi berikutnya untuk melanjutkan pencocokan tanpa harus memulai dari awal lagi.

Border table dalam algoritma KMP penting untuk meningkatkan efisiensi pencarian. Tabel ini mencatat panjang dari awalan terpanjang dari pola yang juga merupakan akhiran dari sub-pola tersebut. Dengan menggunakan tabel ini, algoritma KMP dapat "melompat" ke posisi tertentu dalam pola ketika terjadi ketidaksesuaian, sehingga menghindari pemeriksaan ulang karakter yang telah dibandingkan sebelumnya. Proses ini memungkinkan algoritma mencapai kompleksitas waktu $O(n + m)$, di mana n adalah panjang teks dan m adalah panjang pola. Hal ini membuat KMP lebih efisien dibandingkan metode pencocokan pola sederhana dengan kompleksitas $O(nm)$, karena tidak perlu melakukan pencarian ulang dari awal setiap kali ada ketidaksesuaian.

Dalam pengolahan teks, KMP sering digunakan dalam editor teks untuk mencari dan mengganti kata atau frasa, serta dalam mesin pencari untuk menemukan kata kunci dalam dokumen besar. Selain itu, algoritma ini juga digunakan dalam pemrosesan bahasa alami, bioinformatika untuk pencarian urutan DNA atau protein, dan banyak aplikasi lain yang memerlukan pencocokan pola yang cepat dan akurat. Dengan kemampuannya untuk menangani teks besar dengan kompleksitas yang lebih rendah, algoritma KMP tetap menjadi salah satu algoritma pencocokan pola yang bisa diandalkan dan sering digunakan dalam pemrograman.

2.3 Algoritma Boyer-Moore (BM)

Algoritma Boyer-Moore adalah salah satu algoritma pencocokan pola yang paling efisien dan banyak digunakan untuk mencari substring dalam string utama. Algoritma ini dikembangkan oleh Robert S. Boyer dan J Strother Moore, dan pertama kali dipublikasikan pada tahun 1977. Dalam pencocokannya, Algoritma BM membagi kasus menjadi 2. Algoritma BM bekerja dengan mengandalkan dua teknik. Yang pertama adalah *looking-glass technique*, maksud dari teknik ini adalah pencocokan dilakukan mundur mulai dari karakter paling kanan dari string yang dicari. Kemudian yang kedua adalah *character-jump technique*, artinya pencocokan dapat dilakukan dengan melompati indeks tertentu, tidak harus terurut seperti *brute-force*.

Waktu eksekusi paling buruk dari algoritma Boyer-Moore adalah $O(nm+A)$ dengan n mewakili panjang teks, m mewakili panjang *pattern*, dan A adalah ukuran alfabet. Dalam praktiknya, algoritma ini biasanya jauh lebih cepat karena sering dapat melompati sebagian besar karakter di teks utama, terutama ketika pola atau teks mengandung banyak karakter yang berulang. Hal ini dimungkinkan berkat dua heuristik utama yang digunakan dalam algoritma Boyer-Moore: *bad character heuristic* dan *good suffix heuristic*.

2.4 Teknik Perbandingan Dua String Menggunakan Hamming Distance

Hamming distance antara dua string yang memiliki panjang yang sama adalah jumlah huruf yang berbeda pada posisi yang sama. Hamming distance bisa digunakan untuk mengukur kemiripan antara dua string. Semakin kecil nilai hamming distance antar dua string, nilai kemiripan kedua string tersebut semakin besar. Contoh nilai hamming distance antara dua kata berikut adalah:

- Kancil dan Kacung : 4
- 0011 dan 0101 : 2
- Aku dan Kan : 3
- 389302 and 323572 : 3

Penentuan kemiripan dihitung dengan mencari persentase huruf yang benar dari string. Dengan kata lain, semakin sedikit nilai hamming distance antara dua string, semakin sedikit nilai kemiripan keduanya.

2.5 Regular Expression

Regular expression, atau biasa dikenal dengan sebutan regex, adalah sebuah string teks yang bisa digunakan untuk membuat pola (*pattern*) untuk mencocokkan teks lain. Regex biasanya digunakan untuk manipulasi string dan pencocokan pola pada teks. Berikut adalah simbol-simbol yang digunakan pada regex beserta kegunaannya.

a. Quantifiers

Quantifier	Legend	Example	Sample Match
+	One or more	Version \w-\w+	Version A-b1_1
{3}	Exactly three times	\D{3}	ABC
{2,4}	Two to four times	\d{2,4}	156
{3,}	Three or more times	\w{3,}	regex_tutorial
*	Zero or more times	A*B*C*	AAACC
?	Once or none	plurals?	plural

Quantifier	Legend	Example	Sample Match
+	The + (one or more) is "greedy"	\d+	12345
?	Makes quantifiers "lazy"	\d+?	1 in 12345
*	The * (zero or more) is "greedy"	A*	AAA
?	Makes quantifiers "lazy"	A*?	empty in AAA
{2,4}	Two to four times, "greedy"	\w{2,4}	abcd
?	Makes quantifiers "lazy"	\w{2,4}?	ab in abcd

b. Character

Character	Legend	Example	Sample Match
\d	Most engines: one digit from 0 to 9	file_\d\d	file_25
\d	.NET, Python 3: one Unicode digit in any script	file_\d\d	file_9३
\w	Most engines: "word character": ASCII letter, digit or underscore	\w-\w\w\w	A-b_1
\w	.Python 3: "word character": Unicode letter, ideogram, digit, or underscore	\w-\w\w\w	字-ま_𐄂
\w	.NET: "word character": Unicode letter, ideogram, digit, or connector	\w-\w\w\w	字-ま_𐄂
\s	Most engines: "whitespace character": space, tab, newline, carriage return, vertical tab	a\s b\sc	a b c
\s	.NET, Python 3, JavaScript: "whitespace character": any Unicode separator	a\s b\sc	a b c
\D	One character that is not a <i>digit</i> as defined by your engine's \d	\D\D\D	ABC
\W	One character that is not a <i>word character</i> as defined by your engine's \w	\W\W\W\W\W	*-+=)
\S	One character that is not a <i>whitespace character</i> as defined by your engine's \s	\S\S\S\S	Yoyo

Character	Legend	Example	Sample Match
.	Any character except line break	a.c	abc
.	Any character except line break	.*	whatever, man.
\.	A period (special character: needs to be escaped by a \)	a\.c	a.c
\	Escapes a special character	*\\+\\? \\\$\\^\\\\	.*+? \$^\\
\	Escapes a special character	\\[\\{\\}\\]	[[{}]]

c. Logic

Logic	Legend	Example	Sample Match
	Alternation / OR operand	22 33	33
(...)	Capturing group	A(nt pple)	Apple (captures "pple")
\1	Contents of Group 1	r(\w)g\1x	regex
\2	Contents of Group 2	(\d\d)\+(\d\d)=\2\+\1	12+65=65+12
(?: ...)	Non-capturing group	A(?:nt pple)	Apple

2.6 Pengembangan Aplikasi Desktop dalam Bahasa C# Menggunakan Ekosistem .NET dan Framework GUI WPF

C# adalah bahasa pemrograman modern yang dikembangkan oleh Microsoft. Bahasa pemrograman ini didesain untuk menjadi bahasa yang simpel, *powerful*, dan multifungsi. C# memiliki lingkup penggunaan yang luas, mulai dari website, aplikasi desktop, pelayanan berbasis cloud, atau pun yang paling populer: pengembangan game. Bahasa ini menggunakan Object Oriented Programming sebagai basisnya, mirip seperti java. Bahasa C# memiliki beberapa *open-source library* yang dapat digunakan untuk menjalankan fungsi-fungsi tertentu sehingga *developer* tidak perlu untuk membangun suatu fungsionalitas dari awal.

“.NET” adalah salah satu framework yang paling umum digunakan dalam pembuatan perangkat lunak menggunakan bahasa C#, baik itu game, website, ataupun aplikasi desktop. Program yang dibuat dengan “.NET” dapat dijalankan di berbagai sistem operasi—Linux, macOS, Windows, dan lainnya.

Windows Presentation Foundation (WPF) adalah framework *user interface open-source* untuk aplikasi desktop berbasis Windows. WPF diperkenalkan oleh Microsoft sebagai bagian dari .NET Framework 3.0 pada tahun 2006. WPF menyediakan pembangunan UI baik dengan XAML (*markup*) ataupun menggunakan kode (*code-behind*).

Dengan menggunakan .NET dan WPF, pengembang dapat membuat aplikasi desktop yang efisien, responsif, dan memiliki antarmuka pengguna yang menarik menggunakan bahasa C#. Ekosistem .NET menyediakan dukungan yang cukup untuk pengembangan aplikasi, memungkinkan pengembang untuk fokus pada inovasi dan kualitas aplikasi.

2.7 Enkripsi Data Menggunakan Feistel Cipher

Feistel cipher adalah struktur simetris yang digunakan dalam konstruksi cipher blok, dinamai menurut kriptografer Horst Feistel. Ini adalah dasar dari banyak cipher blok terkenal, termasuk DES (Data Encryption Standard). Feistel cipher membagi blok teks biasa menjadi dua bagian dan memprosesnya melalui beberapa putaran enkripsi, menggabungkan operasi permutasi dan substitusi. Setiap putaran menerapkan sub-kunci yang dihasilkan dari kunci utama.

Langkah dasar enkripsi Feistel cipher:

1. Blok masukan dibagi menjadi dua bagian: Kiri (L_0) dan Kanan (R_0).
2. Iterasi:
 - a. Hitung $L_i = R_{i-1}$.
 - b. Hitung $R_i = L_{i-1} \text{ XOR } F(R_{i-1}, K_i)$.
 - c. F adalah fungsi putaran yang mengambil R_{i-1} dan sub-kunci putaran K_i sebagai masukan dan menghasilkan keluaran yang di-XOR dengan L_{i-1} .
 - d. Setelah iterasi terakhir, bagian kiri dan kanan digabungkan dalam urutan terbalik (R_n, L_n) untuk menghasilkan blok ciphertext terakhir.

Langkah deskripsi Feistel cipher: Karena sifat struktur Feistel yang simetris, dekripsi pada dasarnya adalah proses yang sama seperti enkripsi tetapi dengan kunci bulat diterapkan dalam urutan terbalik.

BAB III

IMPLEMENTASI DAN PENGUJIAN

3.1 Langkah-Langkah Umum Pemecahan Masalah

Program Deteksi Individu Berbasis Biometrik Melalui Citra Sidik Jari akan menerima file gambar sidik jari dari pengguna dengan format bitmap. Kemudian, gambar sidik jari masukan akan digunakan untuk mencari nama di database dengan mencocokkan gambar sidik jari masukan dengan gambar sidik jari dari path file yang ada di database. Nama yang didapat kemudian akan digunakan untuk mencari biodata yang bersesuaian di database. Jika nama tidak ditemukan, akan dilakukan pencarian dengan menggunakan bantuan regex.

Proses pengolahan gambar sidik jari pada database adalah sebagai berikut : gambar bitmap akan dibaca dan diubah menjadi matriks biner (0 jika rata-rata pixelnya < 128 dan 1 jika rata-rata pixelnya > 128). Rata-rata pixel yang dimaksud adalah nilai $(R + G + B) / 3$. Setelah itu, setiap delapan angka biner pada matriks akan diubah menjadi ASCII. Pengubahan dilakukan setiap baris matriks biner yang kemudian disimpan pada sebuah list string.

Proses pengolahan gambar sidik jari masukan juga diubah menjadi ASCII dengan cara yang sama dengan langkah pengolahan gambar pada database. Penjelasan lebih detail ada pada subbab-subbab berikutnya. String ASCII yang didapat dari gambar sidik jari masukan digunakan sebagai pattern dan list string dari gambar yang ada di database digunakan sebagai teks. Berikut langkah-langkah algoritma utama:

1. Pattern akan dibuat terlebih dahulu dari gambar sidik jari masukan.
2. Found bernilai false.
3. File gambar yang pathnya ada di database diolah menjadi list string sesuai penjelasan sebelumnya.
4. Pattern akan dicocokkan dengan setiap string pada list string dengan KMP atau BM (sesuai masukan).
5. Jika dikembalikan nilai -1, akan diteruskan perhitungan dengan algoritma hamming-distance. Persentase kemiripan hasil kembalian hamming-distance akan dibandingkan dengan persentase kemiripan terbesar yang dicatat (nol persen untuk nilai awal). Jika persentase saat ini lebih besar, nama file saat ini akan dicatat dan nilai persentase kemiripan terbesar diubah menjadi nilai kembalian saat ini.
6. Langkah 2-4 akan terus diulangi untuk setiap gambar pada database. Jika dikembalikan nilai selain -1 pada pencocokan menggunakan algoritma KMP atau BM, iterasi pada list string dan iterasi file pada database akan dihentikan, found bernilai true, dan path file saat ini dicatat.
7. Jika found bernilai true, langsung mengembalikan path file dengan pesan kosong. Jika found bernilai false, akan dicek dahulu nilai persentase terbesar. Jika nilai persentase $<$

60%, akan dikembalikan pesan “Image not found”. Jika nilai persentase lebih dari itu, akan dikembalikan path file dengan pesan “Exact image not found”.

8. Jika path file tidak kosong, file gambar pada path tersebut ditampilkan di GUI.

3.2 Proses Penyelesaian Masalah dengan Algoritma KMP

Berikut merupakan langkah-langkah pencocokan menggunakan algoritma KMP :

1. Langkah pertama dalam pencocokan string *pattern* P dengan *text* T dengan menggunakan algoritma KMP adalah melakukan perhitungan *border function* ($b(k)$) untuk masing-masing nilai k dari $0..j-1$ (nilai j menyatakan indeks string dimulai dari 0 dan k menyatakan nilai $j-1$). Nilai $b(k)$ merupakan ukuran terbesar dari *prefix* P[0.. k] yang juga merupakan *suffix* P[1.. k]. Untuk $b(0)$ nilainya selalu 0. Hasil perhitungan *border function* disimpan pada sebuah *array of integer*.
2. Inisiasi variabel i dan j dengan nilai 0. Variabel i mewakili indeks T yang sedang diperiksa dan variabel j mewakili indeks P yang sedang diperiksa. Pencarian akan dimulai dengan nilai i dan j tersebut.
3. Selama nilai i lebih kecil dari panjang T, lakukan langkah pemeriksaan dengan langkah-langkah dibawah.
4. Jika karakter pada indeks ke- j pada P sama dengan karakter pada indeks ke- i pada T, periksa lagi apakah j adalah indeks terakhir pada P. Jika j adalah indeks terakhir pada P maka P ditemukan pada T, dikembalikan nilai $i - \text{panjang } P + 1$ yang menunjukkan letak di indeks mana P ditemukan pada T, dan pemeriksaan dihentikan. Jika j bukan indeks terakhir pada P maka pemeriksaan dilakukan kembali dengan nilai j dan i diinkremen.
5. Jika karakter pada indeks ke- j pada P tidak sama dengan karakter pada indeks ke- i pada T dan nilai j lebih dari 0, lakukan kembali pemeriksaan dengan nilai j diubah menjadi $b(j-1)$.
6. Jika karakter pada indeks ke- j pada P tidak sama dengan karakter pada indeks ke- i pada T dan nilai j kurang atau sama dengan 0, lakukan kembali pemeriksaan dengan nilai i di inkremen.
7. Jika nilai i sudah sama dengan panjang P dan pemeriksaan belum menemukan hasil, kembalikan nilai -1 dan hentikan pemeriksaan. Jika menyentuh langkah ini, artinya P tidak ditemukan pada T.

3.3 Proses Penyelesaian Masalah dengan Algoritma BM

Berikut merupakan langkah-langkah pencocokan menggunakan algoritma KMP :

1. Langkah pertama dalam pencocokan string *pattern* P dengan *text* T dengan menggunakan algoritma KMP adalah melakukan perhitungan *last occurrence function* ($L(x)$) untuk masing-masing nilai x dari $0..127$ (sesuai jumlah karakter ascii). Nilai *last occurrence function* untuk suatu x menyatakan pada indeks berapa kemunculan terakhir karakter yang memiliki nomor ascii x pada P. Karakter dengan nomor ascii x yang tidak muncul pada P nilainya diisi -1. Hasil perhitungan *last occurrence function* disimpan pada sebuah *array of integer*.

2. Inisiasi variabel i dengan nilai panjang P dikurangi 1 dan variabel j dengan nilai panjang T dikurangi 1. Variabel i menyatakan indeks P yang sedang diperiksa dan Variabel j menyatakan indeks T yang sedang diperiksa.
3. Selama nilai i tidak lebih besar dari panjang P dikurangi 1 (i belum sampai ujung T), pemeriksaan dilakukan dengan langkah dibawah. Pemeriksaan dimulai dari sebelah kanan P ke arah kiri.
4. Jika karakter pada indeks j pada P sama dengan karakter pada indeks i pada T, periksa apakah nilai j adalah 0. Jika j bernilai 0, kembalikan nilai i yang menunjukkan letak di indeks mana P ditemukan pada T dan hentikan pemeriksaan. Jika j tidak bernilai 0, lakukan pemeriksaan kembali dengan nilai i dan j di dekremen.
5. Jika karakter pada indeks j pada P tidak sama dengan karakter pada indeks i pada T, pemeriksaan dilakukan kembali dengan nilai i ditambahkan $panjang\ P - min(j, 1 + L(i))$ dan nilai j diubah menjadi panjang P dikurangi 1.
6. Jika nilai i sudah lebih dari panjang P dan pemeriksaan belum menemukan hasil, kembalikan nilai -1 dan hentikan pemeriksaan. Jika menyentuh langkah ini, artinya P tidak ditemukan pada T.

3.4 Arsitektur Aplikasi dan Fitur Fungsionalnya

1. Arsitektur Aplikasi

Arsitektur aplikasi kami meliputi *frontend* dan *backend*. *Frontend* bertugas untuk menerima gambar masukan dari pengguna, mengirim path file masukan ke *backend*, menerima path file dari *backend* dan menampilkan gambar hasil serta biodata yang didapat dari database. Bagian *backend* berisi algoritma utama: KMP, BM, hamming-distance, regex, dan database beserta algoritma penghubungnya. Algoritma utama digunakan untuk menentukan biodata dari pemilik gambar sidik jari masukan.


2. Fitur Fungsional

Aplikasi ini memiliki beberapa fitur fungsional. Pertama, aplikasi memiliki fitur untuk memilih dan mengunggah file, khususnya file dengan tipe gambar. Fitur ini akan digunakan oleh pengguna untuk memilih sidik jari yang akan dibandingkan. Kedua, aplikasi memiliki fitur untuk melakukan pencarian dengan memilih salah satu dari 2 metode yang disediakan, yaitu KMP dan BM. Fitur ini berfungsi untuk membandingkan dan mencari sidik jari pada *database* yang paling mirip dengan sidik jari masukkan pengguna. Ketiga, aplikasi memiliki fitur untuk menampilkan data dari *database*. Fitur ini berfungsi untuk menampilkan biodata yang tersimpan pada *database* sesuai dengan sidik jari masukkan pengguna.

3.5 Ilustrasi Kasus

Ketika seorang pengguna ingin mencari sidik jari masukan pada *database*, yang pertama dilakukan oleh program adalah mengambil bagian tengah citra untuk dibandingkan. Kemudian

program akan mengkonversi citra sidik jari masukan milik user menjadi sebuah potongan binary. Lalu, hasil potongan binary hasil konversi tadi dikonversi lagi menjadi ASCII 8 bit.

Citra Sidik Jari	Binary	ASCII
	<pre> 11111111111111111111111111111111 11111111111111111111111111111111 11111111111111111111111111111111 1111111111111111100001000000 000000000000000000000000000000 000000000000000000000000000000 000000000000000000000000000000... </pre>	O?◀I?Ax

String *pattern* ASCII yang dihasilkan kemudian dicari pada string ASCII dari gambar lain. Metode pencarian dapat dilakukan dengan algoritma Boyer-Moore atau Knuth-Morris-Pratt bergantung pada pilihan pengguna. Setelah pencarian berakhir, program akan menampilkan hasil pencarian kepada pengguna.

BAB IV

IMPLEMENTASI DAN PENGUJIAN

4.1 Spesifikasi Teknis Program

Struktur data yang digunakan dalam biodata pada program ini ada 11, yaitu nik (*long*), nama (*string*), tempat_lahir (*string*), tanggal_lahir (*string*), jenis_kelamin(*string*), golongan_darah (*string*), alamat (*string*), agama (*string*), status (*string*), pekerjaan (*string*), dan kewarganegaraan (*string*).

Berikut adalah fungsi dan prosedur yang digunakan pada program.

1. Fungsi BuildLast(string pattern) → *array of integer*

```
public static int[] BuildLast(string pattern) {  
    int[] last = new int[char.MaxValue+1];  
    for (int i = 0; i < last.Length; i++) {  
        last[i] = -1;  
    }  
    for (int i = 0; i < pattern.Length; i++) {  
        last[pattern[i]] = i;  
    }  
    return last;  
}
```

2. Fungsi BmMatch(string text, string pattern) → *integer*

```
public static int BmMatch(string text, string pattern) {  
    int[] last = BuildLast(pattern);  
    int n = text.Length;  
    int m = pattern.Length;  
    int i = m - 1;  
  
    if (i > n - 1) {  
        return -1;  
    }  
  
    int j = m - 1;  
    do {  
        if (pattern[j] == text[i]) {  
            if (j == 0) {
```

```

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

return -1;
}

```

3. Fungsi `hammingDist(string str1, string str2) → integer`

```

public static int hammingDist(String str1,
                               String str2)
{
    if (str1.Length == str2.Length) {
        int i = 0, count = 0;
        while (i < str1.Length)
        {
            if (str1[i] != str2[i])
                count++;

            i++;
        }
        return count;
    } else {
        return str1.Length;
    }
}

```

4. Fungsi `KMPCompare(string text, string pattern) → integer`

```

public static int KMPCompare(string text, string pattern) {
    int i = 0;

```

```

int j = 0;
int patternLength = pattern.Length;
int textLength = text.Length;
int[] b = computeBorder(pattern);

while (i < textLength) {
    if (pattern[j] == text[i]) {
        if (j == patternLength - 1) {
            return i - patternLength + 1;
        }
        i++;
        j++;
    } else if (j > 0) {
        j = b[j - 1];
    } else {
        i++;
    }
}
return -1;
}

```

5. Fungsi computeBorder(*string* pattern) → *array of integer*

```

public static int BmMatch(string text, string pattern) {
    int[] last = BuildLast(pattern);
    int n = text.Length;
    int m = pattern.Length;
    int i = m - 1;

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

    int j = m - 1;
    do {
        if (pattern[j] == text[i]) {
            if (j == 0) {
                return i;
            }
        } else {
            i--;
        }
    } while (i >= 0);
    return -1;
}

```

```

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

return -1;
}

```

6. Fungsi `ConvertToBinaryImage(string imagePath) → array of array of integer`

```

public static List<List<int>> ConvertToBinaryImage(string imagePath) {
    Bitmap inputImage = new Bitmap(imagePath);
    int width = inputImage.Width;
    int height = inputImage.Height;
    List<List<int>> binaryMatrix = new List<List<int>>(height);

    for (int y = 0; y < height; y++)
    {
        List<int> row = new List<int>(width);
        for (int x = 0; x < width; x++)
        {
            Color pixelColor = inputImage.GetPixel(x, y);
            int grayValue = (int) (pixelColor.R * 0.3 + pixelColor.G * 0.59 +
pixelColor.B * 0.11);
            row.Add(grayValue >= 128 ? 1 : 0);
        }
        binaryMatrix.Add(row);
    }

    return binaryMatrix;
}

```

7. Fungsi `IntToString(array of array of integer binaryMatrix) → array of string`

```

public static List<string> IntToString(List<List<int>> binaryMatrix) {

```

```

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

foreach (List<int> row in binaryMatrix)
{
    StringBuilder asciiString = new StringBuilder();

    for (int i = 0; i < row.Count - row.Count % 8; i += 8)
    {
        int asciiValue = 0;
        for (int j = 0; j < 8; j++)
        {
            asciiValue = (asciiValue << 1) | row[i + j];
        }
        asciiString.Append((char)asciiValue);
    }
    asciiRows.Add(asciiString.ToString());
}

return asciiRows;
}

```

8. Fungsi PatternMatching(*string* patternString, *string* targetString) → *boolean*

```

public bool PatternMatching(string patternString, string targetString) {
    string pattern = MakePattern(patternString);

    Regex regex = new Regex(pattern);
    return regex.IsMatch(targetString);
}

```

9. Fungsi MakePattern(*string* s) → *string*

```

public string MakePattern(string s)
{
    StringBuilder pattern = new StringBuilder();

    foreach (char c in s)
    {
        if (c == 'A' || c == 'a')
        {

```

```

        pattern.Append("(?:[Aa4])?");
    }
    else if (c == 'B' || c == 'b')
    {
        pattern.Append("(?:[Bb8])");
    }
    else if (c == 'D' || c == 'd')
    {
        pattern.Append("(?:[Dd])");
    }
    else if (c == 'E' || c == 'e')
    {
        pattern.Append("(?:[Ee3])?");
    }
    else if (c == 'I' || c == 'i')
    {
        pattern.Append("(?:[Ii1])?");
    }
    else if (c == 'U' || c == 'u')
    {
        pattern.Append("(?:[Uu])?");
    }
    else if (c == 'O' || c == 'o')
    {
        pattern.Append("(?:[Oo0])?");
    }
    else if (c == 'R' || c == 'r')
    {
        pattern.Append("(?:[Rr]|12)");
    }
    else if (c == 'G' || c == 'g')
    {
        pattern.Append("(?:[Gg]|6)");
    }
    else if (c == 'T' || c == 't')
    {
        pattern.Append("(?:[Tt7])");
    }
    else if (c == 'S' || c == 's')
    {
        pattern.Append("(?:[Ss5])");
    }

```

```

    }
    else
    {
        pattern.Append($"[{char.ToLower(c)}{char.ToUpper(c)}]");
    }
}

return pattern.ToString();
}

```

10. Fungsi `Search_FingerPrint(array of string ImageAscii, boolean isKMP) → array of string`

```

public static List<string> Search_FingerPrint(List<string> ImageAscii, bool
isKMP) {
    string ImageAsciiPartial = GetStringToMatch(ImageAscii);
    Database db = new Database();
    db.connect();
    List<string> allFiles = db.selectPathFromSidikJari();
    Console.WriteLine(allFiles[0]);

    int length = ImageAscii.Count;
    int width = ImageAscii.ElementAt(0).Length;
    int best_value_fp = 0;
    string result = "";
    string r = "";
    bool found = false;
    foreach (string file in allFiles) {
        List<string> ImageAscii2 =
ImageToString.IntToString(ImageToString.ConvertToBinaryImage(file));
        foreach (string row in ImageAscii2) {
            if (KMP.KMPCompare(row, ImageAscii) == -1) {
                for (int i = 0; i <= row.Length - ImageAscii.Length; i++) {
                    string substring = row.Substring(i, ImageAscii.Length);
                    int LCS_Result = Hamming.hammingDist(ImageAscii,
substring);

                    if (LCS_Result < best_value_fp)
                    {
                        best_value_fp = LCS_Result;
                        result = file;

```



```

        r = substring;
    }
}
} else {
    r = row;
    result = file;
    Console.WriteLine(file);
    Console.WriteLine(row);
    Console.WriteLine(ImageAscii);
    found = true;
    break;
}
}
if (found) {
    break;
}
}
List<string> res = new List<string>();
res.Add(result);
if (!found){
    if (best_value_fp < 55) {
        res.Add("\nNot found the image\n");
        res.Add(Convert.ToString(best_value_fp));
    } else {
        res.Add("\nNot found the exact same image\n");
        res.Add(Convert.ToString(best_value_fp));
    }
} else {
    res.Add("\n");
    res.Add("0");
}
return res;
}

```

11. Fungsi GetStringToMatch(array of string imageString) → string

```

public static string GetStringToMatch(List<string> imageString) {
    int colLength = imageString[0].Length;

    // Determine the row to use
    int selectedRow = imageString.Count / 2;
}

```

```

    int desiredLength = 8;
    int minLength = 4;
    int start = Math.Max(0, (collength - desiredLength) / 2);

    int end = Math.Min(collength, start + desiredLength);

    if (end - start < minLength)
    {
        start = Math.Max(0, (collength - minLength) / 4);
        end = Math.Min(collength, start + minLength);
    }

    // Build the result string
    StringBuilder result = new StringBuilder();
    for (int i = start; i < end; i++)
    {
        result.Append(imageString[selectedRow][i]);
    }

    return result.ToString();
}

```

12. (BONUS) Kelas FeistelCipher

```

public class FeistelCipher
{
    private const int NumRounds = 16;
    private readonly byte[] key;

    public FeistelCipher(byte[] key)
    {
        this.key = key;
    }

    public string Encrypt(string plaintext)
    {
        byte[] bytes = Encoding.UTF8.GetBytes(plaintext);
        if (bytes.Length % 2 != 0)
        {
            Array.Resize(ref bytes, bytes.Length + 1);
        }
    }
}

```

```

        byte[] encryptedBytes = Process(bytes, true);
        return Convert.ToBase64String(encryptedBytes);
    }

    public string Decrypt(string ciphertext)
    {
        byte[] bytes = Convert.FromBase64String(ciphertext);
        byte[] decryptedBytes = Process(bytes, false);
        return Encoding.UTF8.GetString(decryptedBytes).TrimEnd('\0');
    }

    private byte[] Process(byte[] data, bool isEncrypt)
    {
        int halfLength = data.Length / 2;
        byte[] left = new byte[halfLength];
        byte[] right = new byte[halfLength];
        Array.Copy(data, 0, left, 0, halfLength);
        Array.Copy(data, halfLength, right, 0, halfLength);

        for (int i = 0; i < NumRounds; i++)
        {
            byte[] temp = right;
            right = Xor(left, F(right, isEncrypt ? i : NumRounds - 1 - i));
            left = temp;
        }

        byte[] result = new byte[data.Length];
        Array.Copy(right, 0, result, 0, halfLength);
        Array.Copy(left, 0, result, halfLength, halfLength);

        return result;
    }

    private byte[] F(byte[] data, int round)
    {
        byte[] result = new byte[data.Length];
        for (int i = 0; i < data.Length; i++)
        {
            result[i] = (byte)(data[i] ^ key[round % key.Length]);
        }
    }

```

```

    }
    return result;
}

private byte[] Xor(byte[] a, byte[] b)
{
    byte[] result = new byte[a.Length];
    for (int i = 0; i < a.Length; i++)
    {
        result[i] = (byte)(a[i] ^ b[i]);
    }
    return result;
}
}

```

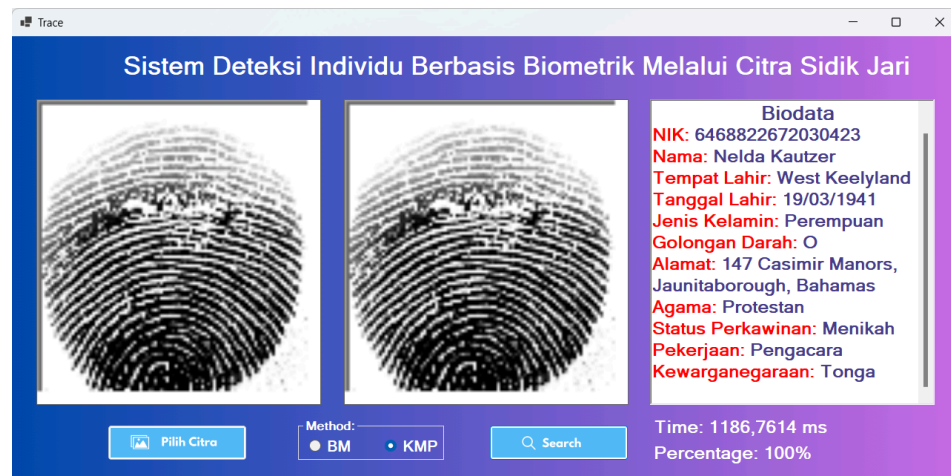
4.2 Tata Cara Penggunaan Program

Berikut adalah petunjuk penggunaan program.

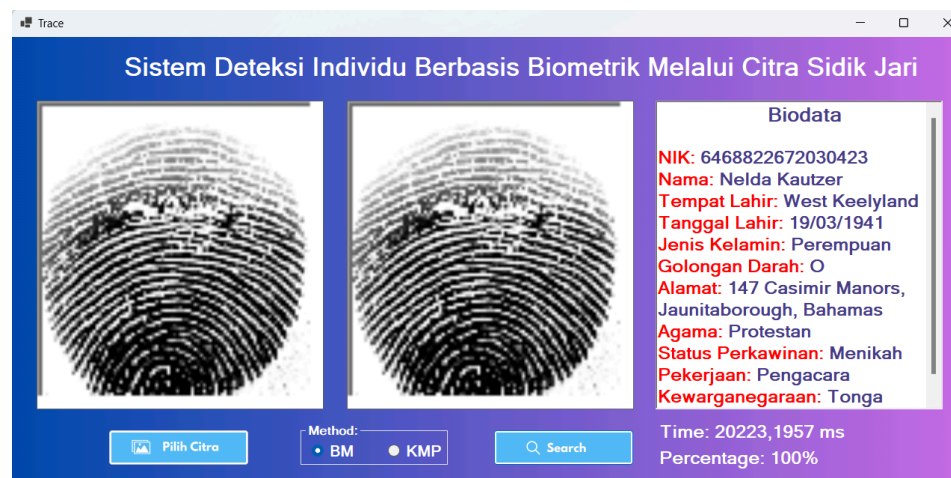
1. Clone repository github dengan perintah “git clone https://github.com/fabianradenta/Tubes3_Trace.git” pada terminal.
2. Pada terminal, masuk ke *directory* Tubes3_Trace/src/gui.
3. Jalankan program dengan perintah “dotnet run” pada terminal.
4. Setelah melakukan langkah 3, aplikasi desktop akan berjalan pada komputer.
5. Pada kolom input citra sidik jari, masukkan citra sidik jari yang ingin diuji.
6. Program akan melakukan pencarian pada *database* dan menampilkan hasil pencarian ke layar komputer.

4.3 Hasil Pengujian

Test Case	Gambar Hasil
1	KMP

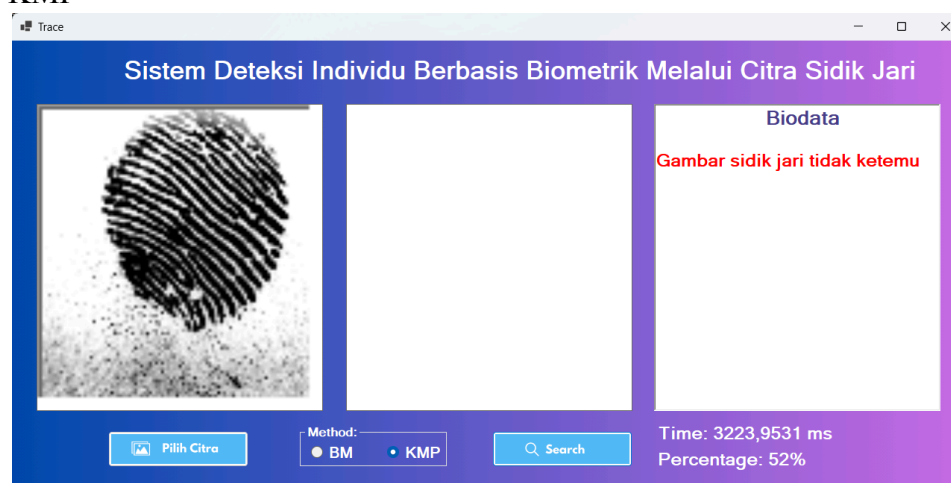


BM

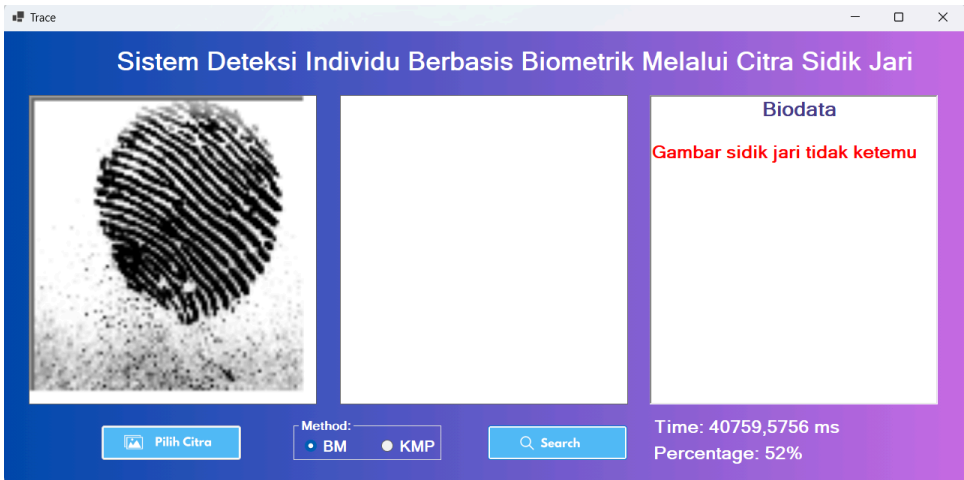
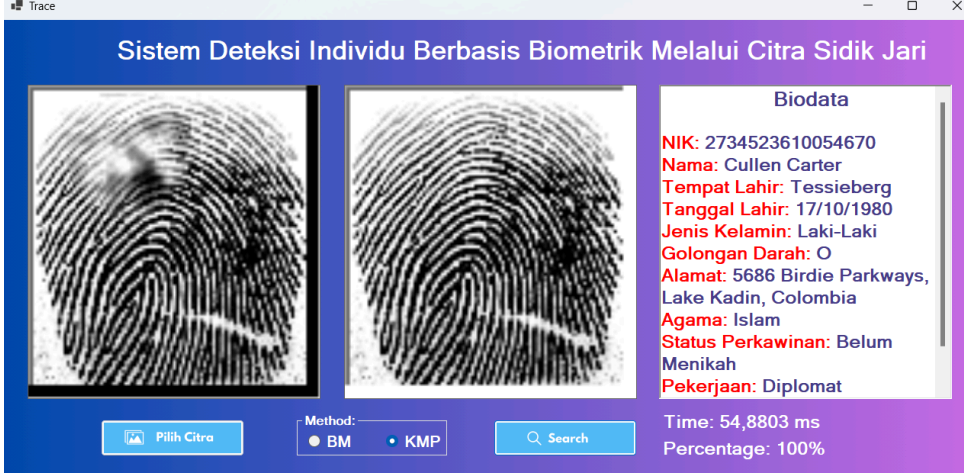



2

KMP





BM

	
3	<p>KMP</p> 
	<p>BM</p> 
4	<p>KMP</p>

Trace

Sistem Deteksi Individu Berbasis Biometrik Melalui Citra Sidik Jari

Biodata

NIK: 3696054948170641
 Nama: Oswaldo Grant
 Tempat Lahir: New Dollyfort
 Tanggal Lahir: 19/04/1932
 Jenis Kelamin: Perempuan
 Golongan Darah: O
 Alamat: 35704 Sandy Underpass, Tremainetown, Svalbard & Jan Mayen Islands
 Agama: Protestan
 Status Perkawinan: Menikah
 Pekerjaan: Programmer

Pilih Citra

Method: ☐ BM ☒ KMP

Search

Time: 3390,5543 ms
 Percentage: 78%

BM

Trace

Sistem Deteksi Individu Berbasis Biometrik Melalui Citra Sidik Jari




Biodata

NIK: 3696054948170641
 Nama: Oswaldo Grant
 Tempat Lahir: New Dollyfort
 Tanggal Lahir: 19/04/1932
 Jenis Kelamin: Perempuan
 Golongan Darah: O
 Alamat: 35704 Sandy Underpass, Tremainetown, Svalbard & Jan Mayen Islands
 Agama: Protestan
 Status Perkawinan: Menikah
 Pekerjaan: Programmer

Pilih Citra

Method: ☒ BM ☐ KMP

Search



Time: 42793,8267 ms
 Percentage: 78%

5

KMP

Trace

Sistem Deteksi Individu Berbasis Biometrik Melalui Citra Sidik Jari

Biodata

NIK: 2962400354475850
 Nama: Mary Bernhard
 Tempat Lahir: Rowefurt
 Tanggal Lahir: 27/08/1988
 Jenis Kelamin: Perempuan
 Golongan Darah: A
 Alamat: 20567 Marta Lake, Dixieport, Libyan Arab Jamahiriya
 Agama: Islam
 Status Perkawinan: Cerai
 Pekerjaan: Desainer grafis

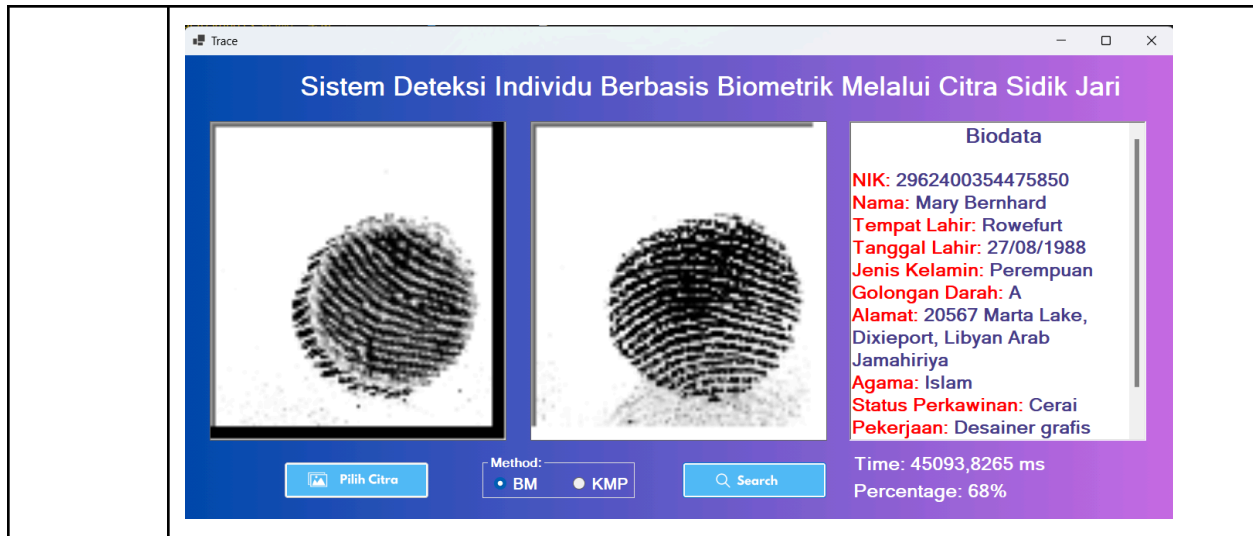
Pilih Citra

Method: ☐ BM ☒ KMP

Search

Time: 3057,2614 ms
 Percentage: 68%

BM



4.2 Analisis Hasil Pengujian

Berdasarkan hasil pengujian di atas, didapatkan bahwa algoritma Boyer-Moore (BM) selalu lebih lambat dari algoritma Knuth-Morris-Pratt (KMP). Hal ini sejalan dengan teori kompleksitas waktu kedua algoritma tersebut. Secara teori, algoritma KMP memiliki kompleksitas waktu sebesar $O(m + n)$ pada kasus terburuknya, sedangkan algoritma BM memiliki kompleksitas waktu sebesar $O(mn + A)$ pada kasus terburuknya. A pada BM merepresentasikan banyaknya jenis karakter yang diproses oleh algoritma tersebut. Semakin banyak jumlah karakter yang diproses, semakin cepat algoritma BM.

Pada *test case* 1, kedua algoritma tersebut dapat menemukan citra sidik jari yang sesuai pada *database*. Namun, pada *test case* 2, tidak ada satu pun algoritma yang dapat menemukan citra sidik jari yang sesuai. Hal tersebut dikarenakan tidak ada data sidik jari masukkan pada *database* dan tidak ada citra sidik jari pada *database* yang memiliki kemiripan di atas 55% dengan sidik jari tersebut sehingga program tidak menampilkan apapun. Pada *test case* 3, 4, dan 5, kedua algoritma berhasil menemukan sidik jari yang sesuai dengan masukkan, meskipun citra sidik jari tersebut rusak ataupun diubah (*altered*). Hal tersebut dikarenakan tingkat kemiripan kedua sidik jari tersebut berada di atas 55% sehingga hasilnya dapat ditampilkan pada layar.

BAB V

KESIMPULAN, SARAN, TANGGAPAN, DAN REFLEKSI

5.1 Kesimpulan

Dari tugas ini dapat disimpulkan poin-poin sebagai berikut.

1. Algoritma KMP dan BM dapat digunakan untuk melakukan *exact string matching*.
2. Regular Expression dapat digunakan untuk mendeteksi kesamaan dari sebuah *string* yang dimodifikasi penggunaan huruf besar kecilnya, penggunaan angka sebagai pengganti huruf dalam *string*, penyingkatan kata, ataupun kombinasi ketiganya.
3. Algoritma Hamming Distance dapat digunakan untuk mengetahui tingkat kemiripan dari dua string.
4. Pengembangan aplikasi desktop memerlukan bagian *user interface* (UI) dan *back-end* (BE) dengan fungsi dan pekerjaannya masing-masing.
5. Kombinasi dari penggunaan algoritma KMP, algoritma BM, Regular Expression, algoritma Hamming Distance, dan prinsip-prinsip pengembangan aplikasi desktop dapat membangun sebuah Sistem Deteksi Individu Berbasis Biometrik Melalui Citra Sidik Jari.

5.2 Saran

Dari pengerjaan tugas ini, saran dari penulis adalah sebagai berikut :

1. Algoritma pencocokan pada program sebaiknya dioptimasi lagi agar waktu eksekusinya dapat lebih cepat. Namun, hal untuk melakukan ini diperlukan waktu yang lebih banyak.
2. Untuk penggunaan yang lebih *advance*, sebaiknya program dimodifikasi dan dimatangkan terlebih dahulu.

5.3 Tanggapan

Nama	NIM	Tanggapan
Karunia Syukur Baeha	10023478	Keren bang
Muhamad Rafli Rasyiidin	13522088	Aman
Abdullah Mubarak	13522101	Mantep
Fabian Radenta Bangun	13522105	Asik bang

5.4 Refleksi

Pengerjaan tugas besar ini secara umum berjalan dengan lancar. Kendala yang dihadapi adalah waktu pengerjaan lebih banyak di dekat pengumpulan tugas karena pada *range* waktu pengerjaan, mahasiswa S1 Informatika juga menghadapi tugas besar-tugas besar dari mata kuliah lain yang batas waktu pengumpulannya lebih awal dari tugas besar ini dan Ujian Akhir Semester. Namun, dengan koordinasi yang baik dan komitmen yang kuat di antara semua anggota

kelompok untuk menyelesaikan tugas besar ini, tugas dapat terselesaikan dengan cukup baik. Berikut adalah poin-poin refleksi yang disadari agar kedepannya dapat menjadi lebih baik lagi.

1. Pembuatan rencana dan penetapan *timeline* pengerjaan tugas besar akan memudahkan pengerjaan tugas. Tujuan dari ini adalah supaya *progress* pengerjaan dapat di-*track*. Anggota kelompok juga harus dapat memenuhi target-target yang ditetapkan.
2. Penggunaan komentar yang lebih banyak dan lebih jelas dapat memudahkan penulis ataupun pembaca dalam melihat dan memahami *code*.

LAMPIRAN

- Tautan repository Github: github.com/fabianradenta/Tubes3_Trace
- Tautan video : <https://youtu.be/9PscQ586RGg>

DAFTAR PUSTAKA

<https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Route-Planning-Bagian1-2021.pdf>

<https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Route-Planning-Bagian1-2021.pdf>

Effendi Tono Hartono Andri Kurnaedi, D., 2013. Penerapan string matching menggunakan algoritma Boyer-Moore pada translator bahasa Pascal ke C. Majalah Ilmiah Unikom.

Astuti, W. (2017). Analisis String Matching Pada Judul Skripsi Dengan Algoritma Knuth-Morris-Pratt (KMP). Jurnal Ilmiah, 167-172.