

LAPORAN TUGAS BESAR III

IF2211 STRATEGI ALGORITMA



Disusun oleh:

Kelompok 25 Kunyat Gadab

Farhan Nafis Rayhan	13522037
Bagas Sambega Rosyada	13522071
Raden Francisco Trianto Bratadiningrat	13522091

Program Studi Teknik Informatika

Sekolah Teknik Elektro dan Informatika

Institut Teknologi Bandung

2024

DAFTAR ISI

DAFTAR ISI.....	2
DAFTAR GAMBAR.....	3
DAFTAR TABEL.....	4
BAB I	
DESKRIPSI TUGAS.....	5
1.1 Latar Belakang.....	5
1.2 Penjelasan Implementasi.....	6
1.3 Penggunaan Program.....	10
1.3 Spesifikasi Program.....	11
BAB II	
LANDASAN TEORI.....	13
2.1 Knuth-Morris-Pratt (KMP) Algorithm.....	13
2.2 Boyer-Moore (BM) Algorithm.....	14
2.3 Regular Expression (Regex).....	15
2.4 Persentase Kemiripan Menggunakan Levenshtein Distance.....	16
2.5 Desktop Application.....	17
BAB III	
ANALISIS PEMECAHAN MASALAH.....	18
3.1 Langkah-Langkah Pemecahan Masalah.....	18
3.2 Proses Penyelesaian Solusi dengan Algoritma KMP dan BM.....	19
3.3 Fitur Fungsional dan Arsitektur Aplikasi Desktop.....	20
3.4 Ilustrasi Kasus.....	21
BAB IV	
IMPLEMENTASI DAN PENGUJIAN.....	25
4.1 Spesifikasi Teknis Program.....	25
4.2 Tata Cara Penggunaan Program.....	29
4.3 Hasil Pengujian.....	31
4.4 Analisis Hasil Pengujian.....	37
BAB V	
KESIMPULAN DAN SARAN.....	40
5.1 Kesimpulan.....	40
5.2 Saran dan Tanggapan.....	40
5.3 Refleksi.....	41
LAMPIRAN.....	42
DAFTAR PUSTAKA.....	43

DAFTAR GAMBAR

Gambar 1.1 Ilustrasi fingerprint recognition pada deteksi berbasis biometrik

Gambar 1.2 Skema implementasi konversi citra sidik jari.

Gambar 1.3 Skema basis data yang digunakan.

Gambar 1.4 Referensi tampilan UI desktop-app.

Gambar 2.1.2 Contoh Pencocokan dengan algoritma KMP

Gambar 2.1.1 Contoh Border Function untuk KMP

Gambar 2.2.1 Contoh Pencocokan dengan algoritma BM

Gambar 2.4.1 Contoh transformasi string algoritma Levenshtein Distance

DAFTAR TABEL

Tabel 1.1 Hasil konversi citra sidik jari menjadi binary dan konversi ke ASCII 8-bits.

Tabel 1.2 Kombinasi ketiga variasi bahasa alay Indonesia.

Tabel 2.3.1 Notasi, Penjelasan dan Contoh Regex

Tabel 3.4.1 Ilustrasi Pencocokan Fingerprint

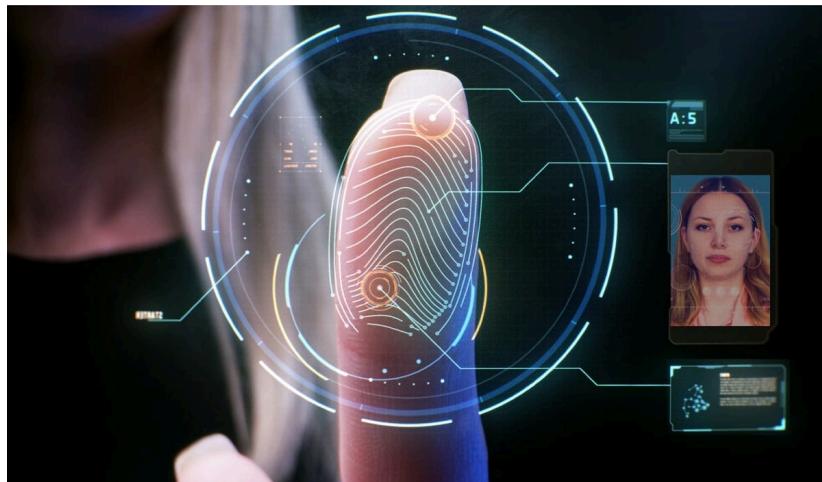
Tabel 3.4.2 Ilustrasi Konversi Gambar menjadi String Binary

Tabel 3.4.2 Ilustrasi Konversi Gambar menjadi String Binary

BAB I

DESKRIPSI TUGAS

1.1 Latar Belakang



Gambar 1.1 Ilustrasi fingerprint recognition pada deteksi berbasis biometrik
(sumber: <https://www.aratek.co/news/unlocking-the-secrets-of-fingerprint-recognition>)

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

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

Dengan menggabungkan teknologi identifikasi sidik jari dan pattern matching, dimungkinkan untuk membangun sistem identifikasi biometrik yang aman, handal, dan mudah

digunakan. Sistem ini dapat diaplikasikan di berbagai bidang, seperti kontrol akses, absensi karyawan, dan verifikasi identitas dalam transaksi keuangan.

Di dalam Tugas Besar 3 ini, Anda diminta untuk mengimplementasikan sistem yang dapat melakukan identifikasi individu berbasis biometrik dengan menggunakan sidik jari. Metode yang akan digunakan untuk melakukan deteksi sidik jari adalah Boyer-Moore dan Knuth-Morris-Pratt. Selain itu, sistem ini akan dihubungkan dengan identitas sebuah individu melalui basis data sehingga harapannya terbentuk sebuah sistem yang dapat mengenali identitas seseorang secara lengkap hanya dengan menggunakan sidik jari.

1.2 Penjelasan Implementasi

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.

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



Gambar 1.2 Skema implementasi konversi citra sidik jari.

(sumber: Tugas Besar IF2211 Strategi Algoritma Semester II tahun 2023/2024)

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 dan penanganan kasus ujung yang baik. Selanjutnya, data pixel tersebut akan dikonversi menjadi binary. Karena binary hanya memiliki variasi karakter satu atau nol, maka proses *pattern matching* akan membuat proses pencocokan karakter menjadi lambat karena harus sering mengulangi proses pencocokan *pattern*. Cara yang dapat dilakukan untuk mengatasi hal tersebut adalah dengan mengelompokkan setiap baris kode biner

per 8 bit sehingga membentuk karakter ASCII. Karakter ASCII 8-bit ini yang akan mewakili proses pencocokan dengan string data.

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

Citra Sidik Jari	Potongan nilai binary	Potongan ASCII 8-bits
	1010000010100000101000001 0100000101000001010000010 1000001010000010100000101 0000010100000101000001010 0000101000001010000010100 0001010000010100000101000 0010100000101000001010000 0101000001010000010100000 10100000101000001010	iÿÿÿÿÿÿÿÿÿIÿÿû:Âÿÿ<Iÿÿû wûÿkÿý,Pþÿû/ÿÿ' »ÿyD¾?ÿ&¼j¤Úa†È/Zíÿÿq ¼ÿúTý~Rûût`íÿÿÿÿ iÿÿÿÿÿÿÿÿöÿÿ"bÿýÁüþÿÊ 'þÿœ2ÿúYÿÿ\ ^ÿÿp fÿÿà !>ÿÿþÿÿÿÿÿÿ<ðÿÖ ²ÿÿQûÿ N°ÿášÿÿÿÿ iÿÿÿÿÿÿÿÿÿÿòrÛðÿÿÇ½ÿÿ‡å ý<>ÿÙþþ6Gÿÿ;Úÿÿÿfayý

Tabel 1.1 Hasil konversi citra sidik jari menjadi binary dan konversi ke ASCII 8-bits.

(sumber: Tugas Besar IF2211 Strategi Algoritma Semester II tahun 2023/2024)

Pada tahap implementasi di titik ini, telah dihasilkan serangkaian karakter ASCII 8-bit yang merepresentasikan sebuah sidik jari. Hasil ini yang akan dijadikan dasar pencarian sidik jari yang sama dengan daftar sidik jari yang terdapat pada basis data. Pencarian sidik jari yang paling mirip dengan sidik jari yang menjadi masukan pengguna dilakukan dengan algoritma pencocokan *string* Knuth-Morris-Pratt (KMP) dan Boyer-Moore (BM). Jika tidak ada satupun sidik jari pada basis data yang *exact match* dengan sidik jari yang menjadi masukan melalui algoritma KMP ataupun BM, maka gunakan sidik jari paling mirip dengan kesamaan diatas nilai tertentu (*threshold*). Anda diberikan kebebasan untuk menentukan nilai batas persentase kemiripan ini, silahkan melakukan pengujian untuk menentukan nilai *tuning* yang tepat. Metode perhitungan tingkat kemiripan juga dibebaskan. Namun sangat menyarankan untuk menggunakan salah satu dari algoritma Hamming Distance, Levenshtein Distance, ataupun Longest Common Subsequence (LCS).

Fungsi dari deteksi biometrik adalah mengidentifikasi seseorang, oleh sebab itu, pada proses implementasi program ini, sebuah citra sidik jari akan dicocokkan dengan biodata seseorang. Biodata yang dicocokkan terdiri atas data-data yang terdapat pada KTP, antara lain : NIK, nama, tempat/tanggal lahir, jenis kelamin, golongan darah, alamat, agama, status perkawinan, pekerjaan, dan kewarganegaraan. Relasi ini dibuat dalam sebuah basis data dengan skema detail seperti yang tertera pada bagian di bawah ini.

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

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

Gambar 1.3 Skema basis data yang digunakan.

(Sumber: Tugas Besar IF2211 Strategi Algoritma Semester II tahun 2023/2024)

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

Sebuah data yang korup dapat memiliki berbagai macam bentuk. Pada tugas ini, jenis data korup adalah bahasa alay Indonesia. Terdengar lucu, tetapi para pendahulu kita sudah membuat bahasa ini untuk berkomunikasi kepada sesamanya. Dengan mengutip dari berbagai sumber, Anda akan diminta untuk menangani kombinasi dari tiga buah variasi bahasa alay, yaitu kombinasi huruf besar-kecil, penggunaan angka, dan penyingkatan. Contoh kasus bahasa alay dijelaskan dengan detail sebagai berikut.

Variasi	Hasil
Kata orisinal	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 1.2 Kombinasi ketiga variasi bahasa alay Indonesia.
(sumber: Tugas Besar IF2211 Strategi Algoritma Semester II tahun 2023/2024)

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.

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.

1.3 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.



Gambar 1.4 Referensi tampilan UI desktop-app.
(sumber: Tugas Besar IF2211 Strategi Algoritma Semester II tahun 2023/2024)

Anda dapat menambahkan menu lainnya seperti gambar, logo, dan sebagainya. Tampilan front-end dari desktop-app tidak harus sama persis dengan layout yang diberikan di atas, tetapi dibuat semenarik mungkin dan wajib mencakup komponen-komponen berikut:

- Judul aplikasi
- Tombol Insert citra sidik jari, beserta display citra sidik jari yang ingin dicari
- Toggle Button untuk memilih algoritma yang ingin digunakan (KMP atau BM)
- Tombol Search untuk memulai pencarian
- Display sidik jari yang paling mirip dari basis data
- Informasi mengenai waktu eksekusi
- Informasi mengenai tingkat kemiripan sidik jari dengan gambar yang ingin dicari, dalam persentase (%)
- List biodata hasil pencarian dari basis data. Keluarkan semua nilai atribut dari individu yang dirasa paling mirip. Perlu diperhatikan pendefinisian batas kemiripan dapat memunculkan kemungkinan tidak ditemukan list biodata yang memiliki sidik jari paling mirip.

Secara umum, berikut adalah cara umum penggunaan program:

1. Pengguna terlebih dahulu memasukkan citra sidik jari yang ingin dicari biodatanya.
2. Setelah citra dimasukkan, pilih opsi pencarian, ingin melakukan pencarian apakah akan menggunakan algoritma KMP atau BM.
3. Tekan tombol search, program kemudian akan memproses, mencari citra sidik jari dari basis data yang memiliki kemiripan dengan citra sidik jari yang menjadi masukan.
4. Program akan menampilkan list biodata jika ditemukan citra sidik jari yang memiliki kemiripan dengan batas persentase tertentu. Program juga dapat mengeluarkan informasi tidak ada sidik jari yang mirip jika semua citra dalam basis data tidak memiliki kemiripan dengan masukan.
5. Terdapat informasi terkait waktu eksekusi program dan persentase kemiripan citra.

1.3 Spesifikasi Program

Pada Tugas Besar ini, buatlah sebuah sistem yang dapat melakukan identifikasi individu berbasis biometrik dengan menggunakan sidik jari dengan detail sebagai berikut.

1. Sistem dibangun dalam bahasa C# dengan kakas Visual Studio .NET yang mengimplementasikan algoritma KMP, BM, dan Regular Expression dalam mencocokkan sidik jari dengan biodata yang berpotensi rusak.
2. Program dapat memiliki basis data SQL yang telah mencocokkan berkas citra sidik jari yang telah ada dengan seorang pribadi. Basis data yang digunakan dibebaskan asalkan bukan No-SQL (sebagai contoh, MySQL, PostgreSQL, SQLite).
3. Program dapat menerima masukan sebuah citra sidik jari yang ingin dicocokkan. Apabila citra tersebut memiliki kecocokan di atas batas tertentu (silakan lakukan tuning nilai yang tepat) dengan citra yang sudah ada, maka tunjukkan biodata orang tersebut. Apabila di bawah nilai yang telah ditentukan tersebut, memunculkan pesan bahwa sidik jari tidak dikenali.
4. Program memiliki keluaran yang minimal mengandung seluruh data yang terdapat pada contoh antarmuka pada bagian penggunaan program.
5. Pengguna dapat memilih algoritma yang ingin digunakan antara KMP atau BM.
6. Biodata yang ditampilkan harus biodata yang memiliki nama yang benar (gunakan Regex untuk memperbaiki nama yang rusak dan gunakan KMP atau BM untuk mencari orang yang paling sesuai).
7. Program memiliki antarmuka yang user-friendly. Anda juga dapat menambahkan fitur lain untuk menunjang program yang Anda buat (unsur kreativitas).

BAB II

LANDASAN TEORI

2.1 Knuth-Morris-Pratt (KMP) Algorithm

Knuth-Morris-Pratt disingkat KMP merupakan salah satu algoritma pencocokan string yang paling populer. KMP mencari pola kata pada suatu teks dengan urutan dari kiri menuju kanan. Namun berbeda dengan algoritma pencocokan string secara *brute-force*, KMP melakukan pergeseran pola dengan cara yang lebih baik.

KMP melakukan pergeseran pola dengan jumlah pergeseran sebaik mungkin untuk mencegah pencocokan yang tidak berguna sehingga pencocokan string masih akurat. Jumlah pergeseran terbaik untuk suatu pola P dari sebuah teks adalah ukuran prefix dari $P[0..j-1]$ yang merupakan suffix dari $P[1..j-1]$.

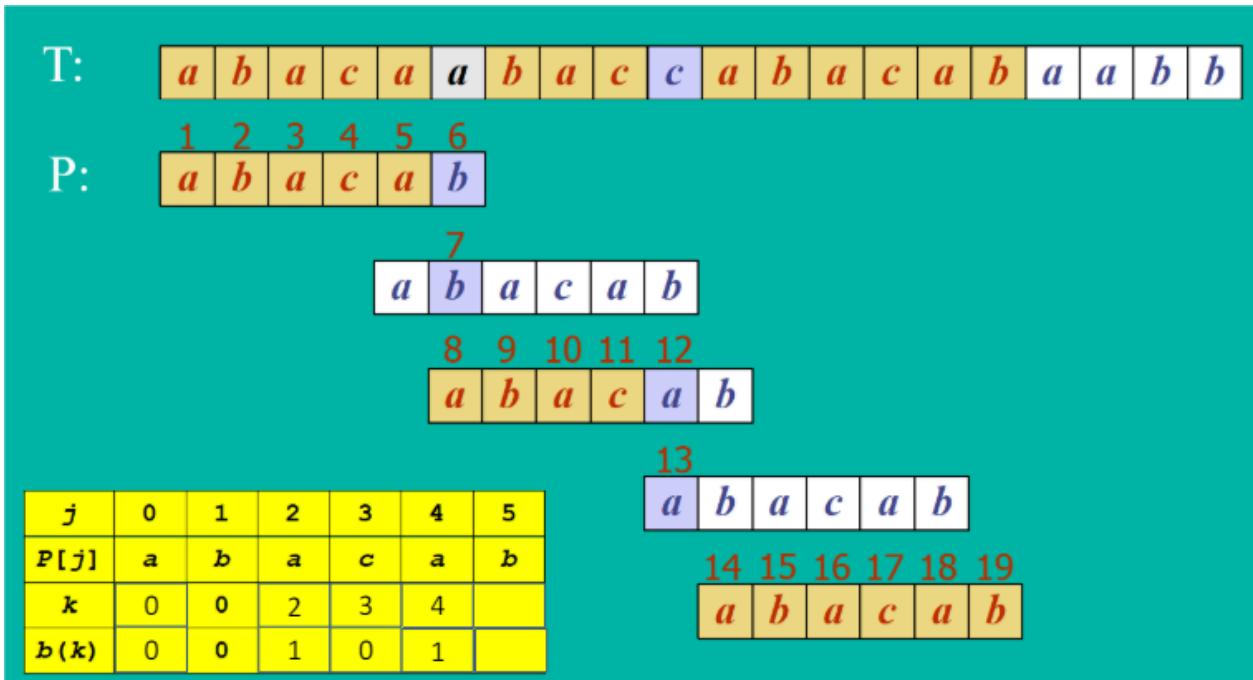
Dengan menggunakan *border function* yang merupakan fungsi yang memberikan ukuran terbesar prefix dari pola $[0..k]$ yang juga merupakan suffix dari pola $[1..k]$ dengan $k = j - 1$. *Border function* ini memberikan jumlah pergeseran pola sehingga tidak lagi terjadi pencocokan yang tidak berguna(*wasteful*).

j	0	1	2	3	4	5
$P[j]$	a	b	a	a	b	a

k	0	1	2	3	4
$b(k)$	0	0	1	1	2

Gambar 2.1.1 Contoh *Border Function* untuk KMP

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



Gambar 2.1.2 Contoh Pencocokan dengan algoritma KMP

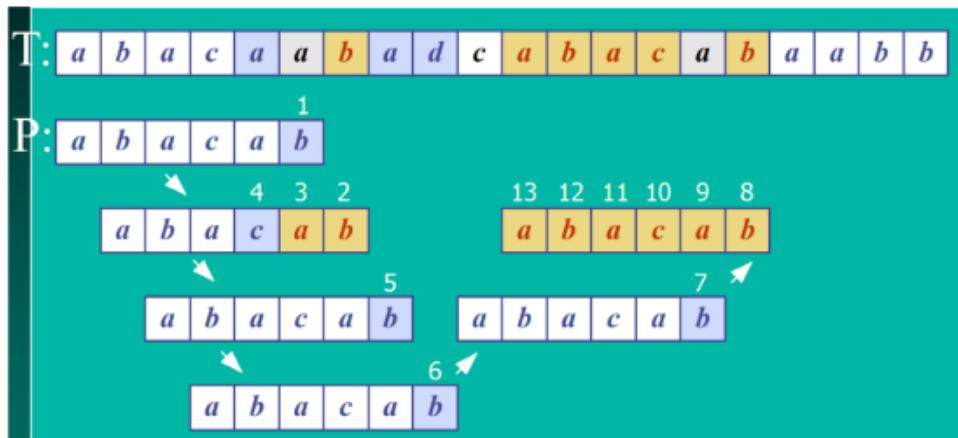
(sumber: <https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Pencocokan-string-2021.pdf>)

2.2 Boyer-Moore (BM) Algorithm

Algoritma Boyer-Moore atau BM adalah salah satu algoritma pencocokan string yang paling populer. Algoritma BM didasarkan dua teknik, yaitu teknik *looking-glass* dan teknik *character-jump*. Teknik *looking-glass* adalah teknik pencocokan string yang melakukan pencocokan pola dimulai dari ujung pola hingga awal dari pola atau bersifat terbalik. Teknik character-jump adalah teknik pergeseran ketika terjadi kesalahan pencocokan. Teknik ini memiliki 3 kasus yang mungkin yang dicoba secara berurutan untuk pola P dan teks T adalah sebagai berikut:

1. Jika terdapat suatu karakter x ada dalam P, maka lakukan pergeseran ke kanan hingga posisi kemunculan terakhir x dalam P sama dengan $T[i]$.
2. Jika terdapat suatu karakter x ada dalam P, tetapi pergeseran ke kanan tidak mungkin, maka lakukan pergeseran P ke kanan sebanyak 1 karakter menuju $T[i+1]$.
3. Jika kasus 1 dan 2 tidak terpenuhi, maka lakukan pergeseran P ke kanan hingga $P[0]$ dan $T[i+1]$ berposisi sama

Algoritma BM melakukan *pre-processes* pada pola P dan alfabet A untuk membuat fungsi *last occurrence function* L() yang memetakan semua huruf pada P menjadi bilangan bulat(integer) yang digunakan untuk melakukan pergeseran. L(x) didefinisikan sebagai indeks terbesar i sehingga $P[i] == x$, atau -1 jika indeks tidak ada.



Gambar 2.2.1 Contoh Pencocokan dengan algoritma BM

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

2.3 Regular Expression (Regex)

Regular expression (*regex*) adalah sebuah pola yang digunakan untuk mencocokkan kombinasi karakter pada sebuah string. *Regex* merupakan sebuah implementasi dan representasi dari bahasa formal (MDN Docs: Regex). Sebuah pola *regex* dapat membentuk berbagai kombinasi kata atau bahkan kalimat sesuai definisi pola tersebut. Pola *regex* tersebut digunakan untuk menentukan apakah sebuah kata/kalimat sesuai dengan pola *regex* yang digunakan atau tidak.

Regular expression menggunakan beberapa notasi tertentu untuk menentukan apakah sebuah string bersesuaian dengan pola *regex* yang digunakan. Beberapa karakter/notasi yang digunakan pada pola *regex* adalah sebagai berikut,

Tabel 2.3.1 Notasi, Penjelasan dan Contoh Regex

Notasi	Penjelasan dan Contoh
[abc]	Sebuah string dapat berisi salah satu dari karakter a, b, atau c
.	Sebuah karakter apapun, misalnya pada ‘res.si’, di antara karakter res- dan -us dapat menjadi karakter apapun, dapat menjadi resesi, resisi, dan sebagainya

[X]*	Karakter X dapat muncul lebih dari 1 kali bahkan tidak muncul sama sekali
[a-g]	Dapat berisi karakter apapun antara a-g
[Oo]	Dapat berisi huruf O besar atau o kecil
\t , \s , \n , *	Tab, space, newline, dan karakter ‘*’

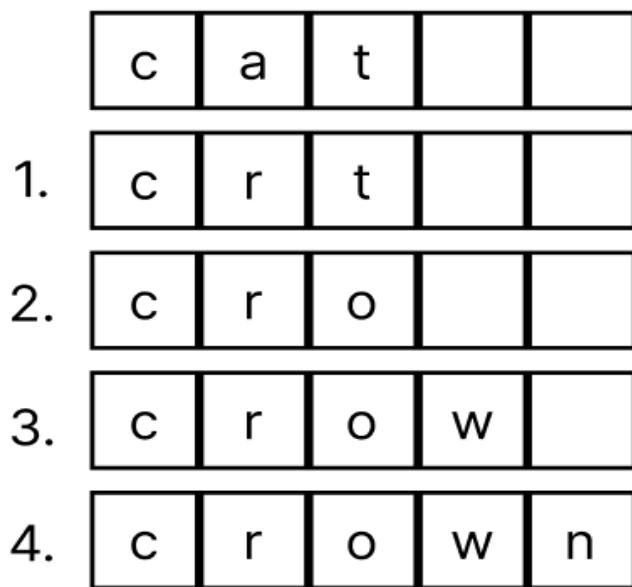
Sebagai sebuah contoh, diberikan sebuah pola *regex* sebagai berikut: [HhAa] *haloo? , maka sebuah string yang dapat dibentuk atau memiliki kesesuaian dengan pola *regex* tersebut di antaranya adalah “HaHahahaloo”, “hahahaloo”, “haHahaloo”, “haloo”, dan lain-lain.

2.4 Persentase Kemiripan Menggunakan Levenshtein Distance

Pencocokan dengan algoritma KMP dan BM adalah tipe pencocokan string yang eksak sehingga hanya dapat menentukan apakah suatu pola, keseluruhannya, terdapat pada suatu teks. Namun terdapat kemungkinan bahwa pencocokan tersebut gagal sehingga diperlukan penggunaan algoritma pencocokan string yang tidak eksak, tapi tingkat kemiripan dari pola ut pada teks.

Salah satu algoritma pencocokan tersebut adalah Levenshtein Distance. Levenshtein Distance atau jarak Levenshtein mengukur tingkat kemiripan antara dua buah string dengan menghitung jumlah operasi insertion(menambahkan huruf), deletion(menghapus huruf), dan substitution(mengganti huruf) yang diperlukan untuk mengubah satu string menjadi string yang lainnya. Jumlah operasi tersebut adalah jarak dari Levenshtein Distance. Menggunakan jarak ini kita dapat menghitung persentase kemiripan dua buah string.

$$similarity = (1 - \frac{\text{Levenshtein distance}}{\max_len}) \times 100$$



Gambar 2.4.1 Contoh transformasi string algoritma Levenshtein Distance

2.5 Desktop Application

Graphical user interface (GUI) merupakan sebuah program yang digunakan untuk menampilkan tampilan antar pengguna berbasis grafis, sehingga menciptakan interaksi antara pengguna dengan program lebih intuitif. Dalam pembuatan aplikasi desktop ini, kami menggunakan bahasa pemrograman C# dengan kerangka kerja .NET, dilengkapi dengan GUI berbasis WinForms.

Windows Forms (WinForms) adalah kelas GUI yang didesain khusus untuk menciptakan aplikasi *desktop* pada sistem operasi Windows. Kakas ini menyediakan UI yang komprehensif meliputi *button*, *text box*, dan *label*. Salah satu kelebihan Winforms yang menjadi alasan kami memilih menggunakan kakas ini adalah sistemnya yang simpel dan membantu pengembangan yang cepat, tetapi juga lengkap dan didukung penuh oleh .NET. Terlebih, integrasi dengan Visual Studio memungkinkan pengembang UI melakukan desain dengan *drag and drop*, yang sangat meringkas waktu pengembangan *desktop application* kami.

BAB III

ANALISIS PEMECAHAN MASALAH

3.1 Langkah-Langkah Pemecahan Masalah

Secara garis besar langkah-langkah pemecahan permasalahan pencocokan fingerprint adalah sebagai berikut:

1. Mengubah semua pixel pada gambar bitmap menjadi binary 0 atau 1.
2. Melakukan pengelompokan pixel-pixel menjadi kotak-kotak 8x2 dan menghasilkan string panjang hasil konversi gambar menjadi binary. Jika gambar tidak cukup untuk membentuk kotak 8x2 maka isi ruang kosong dengan 0.
3. Setiap 8 karakter pada string binary, diubah menjadi karakter 8-bit ascii sehingga menghasilkan konversi dari gambar menjadi ascii.
4. Membagi hasil konversi ascii dari gambar pola menjadi 16 bagian.
5. Ambil 8 karakter dari tengah 16 bagian ascii tersebut. Gunakan 16 string berisi 8 karakter untuk melakukan pencocokan string terhadap string ascii milik gambar patokan atau sumber.
6. Gunakan algoritma KMP atau BM untuk mendapatkan array yang berisi gambar yang memiliki bobot kecocokan dari KMP atau BM terbesar. Jika isi array hanya 1 gambar, maka pencocokan selesai.
7. Jika tidak ada gambar yang memenuhi algoritma KMP atau BM atau terdapat banyak gambar hasil pencocokan KMP dan BM yang memiliki bobot kecocokan yang sama, gunakan 16 pola tersebut dengan algoritma Levenshtein Distance.
8. Pilih gambar dengan nilai kemiripan yang paling baik. Jika nilai kemiripan terbaik ada lebih dari 73, maka solusi ditemukan dan pencocokan diberhentikan.
9. Jika nilai kemiripan dibawah 73, maka program gagal menemukan gambar yang dianggap cocok.

Untuk setiap pola kami menggunakan 8 karakter ascii atau setara dengan 64 pixel. Hal tersebut dikarenakan 8 karakter memberikan keunikan yang cukup tinggi sehingga lebih sedikit kemungkinan gambar yang sebenarnya tidak cocok dipilih sebagai solusi. Kami juga

menggunakan 16 pola untuk mengatasi fingerprint yang tidak dalam kondisi sempurna seperti terdapat alternifikasi pada bagian-bagian tertentu. Dengan 16 pola kami dapat mengatasi alternasi dari gambar yang cukup berat.

Alasan kami melakukan pengelompokan pixel menjadi kotak berukuran 8x2 pixel adalah untuk membawa elemen kolom pada pola dan pencocokan. Jika kami tidak melakukan hal tersebut kemungkinan pola dan pencocokan akan tidak akurat karena hanya menggunakan 1 baris pixel. Alasan lain kami adalah untuk mencegah perbedaan pola karena perbedaan dimensi dari gambar.

Untuk Batas kemiripan algoritma Levenshtein Distance, yaitu 73. Kami memilih angka tersebut karena itu merupakan batas yang terbaik dari hasil-hasil eksperimen program.

3.2 Proses Penyelesaian Solusi dengan Algoritma KMP dan BM

3.2.1 Pemetaan Permasalahan

- T: teks, yaitu string (sangat panjang) dengan panjang n karakter
Dalam permasalahan ini T adalah hasil konversi keseluruhan gambar menjadi string Ascii.
- P: pattern, yaitu string dengan panjang m ($m < n$) karakter yang digunakan sebagai pola yang dicari dari teks
Dalam permasalahan ini P adalah pola-pola yang diambil dari posisi tertentu dari hasil konversi gambar pola menjadi Ascii.

3.2.2 Proses Penyelesaian dengan algoritma KMP dan BM

Untuk menyelesaikan permasalahan ini dengan algoritma KMP atau BM, kami tidak hanya melakukan satu kali pencocokan. Namun kami menggunakan 16 pola yang masing-masing terdiri dari 8 karakter. Sehingga total kami melakukan 16 kali pencocokan dengan algoritma KMP atau BM.

Untuk setiap gambar yang akan dicocokan kami menyimpan bobot pencocokan sebagai dasar kecocokan dua buah gambar. Semakin besar bobot tersebut maka semakin cocok 2 buah gambar. Bobot pencocokan setiap pola ditentukan oleh posisinya. Semakin dekat dengan pusat

gambar maka semakin besar bobot dari pencocokan. Hal tersebut dikarenakan biasanya tengah dari gambar fingerprint, memiliki pola yang unik dibandingkan dengan bagian gambar lainnya.

Dengan mencari semua gambar yang memiliki bobot pencocokan tertinggi, kita dapat mengurangi gambar-gambar potensi pencocokan. Jika hasil jumlah gambar yang memiliki bobot kecocokan yang sama hanya 1 gambar, maka gambar tersebut menjadi solusi dari pencocokan. Namun jika lebih dari 1 gambar, maka perlu digunakan Levenshtein Distance untuk menentukan gambar mana yang paling mirip. Jika tidak ada gambar yang cocok sama sekali, maka seluruh gambar pada database akan dicocokan menggunakan algoritma Levenshtein Distance. Jika tidak ada gambar dengan nilai kemiripan lebih dari 73 maka dapat disimpulkan fingerprint tersebut tidak ditemukan pada data yang dimiliki.

3.3 Fitur Fungsional dan Arsitektur Aplikasi Desktop

3.3.1. Fitur Memilih Gambar

Aplikasi memungkinkan pengguna untuk memilih gambar fingerprint yang ingin diidentifikasi. Gambar ini tidak harus sudah berada di dalam database. Aplikasi membatasi file yang dapat di upload pengguna harus berekstensi .bmp (bitmap). Fitur ini dapat digunakan dengan menekan tombol *Select File*.

3.3.2. Fitur Memilih Algoritma

Aplikasi memungkinkan pengguna untuk menentukan algoritma pencarian. Algoritma pencarian yang diimplementasikan adalah Knuth-Morris-Pratt dan Boyer-Moore. Fitur ini diimplementasikan dalam bentuk tombol *toggle* dengan 2 pilihan yaitu KMP dan BM.

3.3.3. Fitur identifikasi sidik jari

Aplikasi dapat melakukan pencarian sidik jari dalam database yang paling serupa dengan gambar sidik jari yang sebelumnya diinputkan pengguna. Pencarian dilakukan dengan algoritma sesuai yang terpilih pada tombol *toggle*. Aplikasi menampilkan gambar fingerprint paling mirip di sebelah gambar yang di input pengguna. Selain itu, Aplikasi akan menampilkan waktu pencarian serta persentase kemiripan sidik jari. Fitur ini dijalankan dengan menekan tombol *Search*.

3.3.4. Fitur menampilkan biodata pemilik sidik jari

Untuk melengkapi sistem biometrik kami, aplikasi bisa menampilkan biodata pemilik sidik jari secara lengkap. Seluruh biodata pemilik disimpan dalam basis data aplikasi. Antarmuka akan menampilkan biodata meliputi nama, tempat lahir, tanggal lahir, jenis kelamin, golongan darah, alamat, agama, status perkawinan, pekerjaan, kewarganegaraan.

3.3.5. Fitur penanganan data korup

Basis data yang digunakan aplikasi mungkin saja memiliki data korup, sehingga Aplikasi kami dapat menangani hal tersebut. Pada kasus ini, korupsi data yang mungkin terjadi adalah nama pemilik sidik jari yang mungkin tersimpan dalam bahasa *alay*. Fitur ini diimplementasikan agar nama pemilik sidik jari tetap bisa teridentifikasi seperti seharusnya walaupun sudah korup di dalam basis data.

3.3.6. Arsitektur Aplikasi

Arsitektur dari aplikasi yang kami ciptakan adalah sebagai berikut

1. Tampilan Utama
 - 1.1. Judul & Logo Aplikasi
 - 1.2. Gambar sidik jari masukkan pengguna
 - 1.3. Gambar sidik jari hasil pencarian
 - 1.4. Area tombol
 - 1.4.1. Tombol upload file gambar
 - 1.4.2. Tombol *toggle* pemilihan algoritma
 - 1.4.3. Tombol mulai pencarian
2. Tampilan Hasil
 - 2.1. Waktu pencarian
 - 2.2. Persentase kemiripan kedua sidik jari
 - 2.3. Tampilan biodata identitas pemilik sidik jari

3.4 Ilustrasi Kasus

Tabel 3.4.1 Ilustrasi Pencocokan Fingerprint

Pencocokan Fingerprint	
Gambar Hasil Manipulasi	Gambar Asli



Sebagai ilustrasi kasus, kami akan melakukan pencocokan gambar fingerprint hasil manipulasi dengan gambar fingerprint yang bersih. Langkah pertama pencocokan adalah melakukan konversi gambar menjadi binary 1 atau 0. Karena gambar fingerprint sudah dalam tipe *grayscale*, maka semua pixel sudah pasti hanya memiliki satu value yang sama untuk ketiga merah(R), hijau(G), dan biru(B). Kami menggunakan batas intensitas dari *grayscale* yaitu 100 untuk menentukan apakah suatu pixel berubah menjadi nilai 1 atau 0.

Perlu diperhatikan bahwa kami tidak langsung merekonstruksi semua hasil konversi menuju binary menjadi satu string panjang. Namun kami melakukan pembagian gambar menjadi kotak-kotak berukuran 8x2 pixel, bila sisa pixel tidak memenuhi 8x2 maka ruang kosong akan diisi dengan 0. Semua kotak tersebut kemudian digabungkan menjadi satu string panjang berupa 1 dan 0.

Tabel 3.4.2 Ilustrasi Konversi Gambar menjadi String Binary

Gambar	Potongan Binary
	<pre data-bbox="832 1469 1421 1577"> 1111101111111001101100100100100100100 00000000000000000000000000001001001111111 111111111111111111110000 111111111111111111110000 0000000000000000000010010000001101111111 111111111111111111110000 1111111111111111111101000100100100110000100 00000000000000001000001001001001001111111 111111111111111111110000 111111111111111111110000 00000000000000001000001001001001001111111 </pre>

	11111111111111110000
--	----------------------

Hasil string tersebut kemudian diubah menjadi 8-bit Ascii.

Tabel 3.4.3 Ilustrasi Konversi String Binary menjadi String Ascii

Potongan Binary	Hasil Konversi menjadi Ascii
11111011111111001101100100100100100100 0000000000000000000000000000000010010011111111 11111111111111110000 1111111111111111001001001000100100000100 000000000000000010010000001101111111 11111111111111110000 1111111111111111010001001001100001001000 0000000000000010000010001001001111111 11111111111111110000 11111111111111110001001001000001001000 000000000000000010001001001111111 11111111111111110000	ûüÙ\$ÿðÿü\$►☻@ßÿÿðÿýÿâa ►DÿðÿýÿâA Dÿð

Dengan kedua gambar fingerprint sudah diubah menjadi bentuk string ascii, pertama kita perlu mengambil pola ascii dari gambar pola. Kami pertama membagi ascii menjadi 16 bagian kecil. Dari bagian-bagian tersebut, kami mengambil 8 karakter yang terletak pada tengah string tersebut sehingga kami mengambil 16 pola kecil dengan masing-masing pola berisi 8 karakter ascii. Kemudian 16 pola ascii tersebut digunakan untuk melakukan pencocokan string dengan algoritma KMP atau BM.

Dari 16 pola kami menghitung seberapa banyak pola kecil muncul pada gambar perbandingan (gambar-gambar fingerprint yang bersih). Semakin tengah pola, maka bobot kecocokan juga semakin tinggi. Kami menyimpan semua gambar yang memiliki total keberhasilan penemuan pola terbanyak. Jika hanya 1 maka secara langsung itu adalah gambar pencarian yang paling baik. Jika ternyata terdapat banyak gambar dengan jumlah pencocokan hasil algoritma KMP atau BM yang sama, maka cari gambar yang paling mirip dengan gambar asli dengan menggunakan algoritma Levenshtein Distance.

Tabel 3.4.4 Ilustrasi Bobot Kecocokan Gambar

Algoritma	Hasil Bobot Kecocokan	Gambar Pencocokan
-----------	-----------------------	-------------------

KMP	15	
BM	15	

Karena KMP dan BM menemukan hanya satu gambar maka hasil dari pencocokan fingerprint adalah gambar tersebut. Karena ditemukan dengan menggunakan KMP dan BM maka persentase kemiripan bernilai 100%.

BAB IV

IMPLEMENTASI DAN PENGUJIAN

4.1 Spesifikasi Teknis Program

4.1.1 Kelas

Kelas	Keterangan
FingerprintMatching	Berisi fungsi dan procedure yang melakukan pencocokan fingerprint.
ImageProcessing	Berisi fungsi dan procedure untuk memproses gambar seperti melakukan konversi dari gambar menjadi binary atau ascii.
Form1	Berisi fungsi dan procedure untuk implementasi GUI (antarmuka grafis).
KMP	Berisi fungsi dan procedure untuk melakukan pencocokan string dengan algoritma KMP.
BM	Berisi fungsi dan procedure untuk melakukan pencocokan string dengan algoritma BM.
LD	Berisi fungsi dan procedure untuk mencari kemiripan dua buah string dengan algoritma Levenshtein Distance.
Database	Berisi fungsi dan <i>procedure</i> untuk melakukan koneksi dan <i>query</i> dengan basis data.
AlayName	Berisi fungsi dan <i>procedure</i> untuk melakukan konversi dari nama alay (nama dengan penulisan yang salah dengan menggunakan bahasa alay) ke nama yang valid.
ImageDBProcessor	Menjadi <i>controller</i> antara kelas ImageProcessing, kelas Database, dan kelas GUI

4.1.2 Fungsi

- Kelas FingerprintMatching

Fungsi	Keterangan
FingerprintAnalysisBM(string targetPath, Dictionary asciiMap)	Melakukan pencocokan fingerprint dengan algoritma BM

FingerprintAnalysisKMP(string targetPath, Dictionary asciiMap)	Melakukan pencocokan fingerprint dengan algoritma KMP
--	---

- Kelas ImageProcessing

Fungsi	Keterangan
GetAsciiPattern(string imgPath)	Mendapatkan 16 pola ascii dari gambar
ConvertImageToBinary(string imgPath)	Mengkonversi gambar menjadi string binary 0 atau 1
ConvertBinaryToAscii(string binaryString)	Mengkonversi string binary 0 atau 1 menjadi string ascii

- Kelas Form1

Fungsi	Keterangan
Form1(Dictionary<string, string> _asciiMap)	Konstruktor kelas dengan attribute custom yaitu _asciiMap
Void InitializeComponent()	Inisialisasi komponen pada GUI
void Form_Load(object sender, EventArgs e)	Proses yang dilakukan ketika Form GUI di load
void exit_clicked(object sender, EventArgs e)	Prosedur untuk menghentikan aplikasi ketika tombol exit diklik
void minimize_clicked(object sender, EventArgs e)	Prosedur untuk minimize form aplikasi ketika tombol minimize diklik
void SelectFileButton_click(object sender, EventArgs e)	Prosedur pembuka dialog pemilihan file ketika event tombol select file diklik
void SearchButton_click(object sender, EventArgs e)	Prosedur mulai pencarian ketika event tombol search di klik
void toggleAlgorithmClick(object sender, EventArgs e)	Prosedur pergantian algoritma yang dipilih ketika event tombol toggle algoritma diklik
void setSearchStatus(bool enable)	Prosedur perubahan desain tombol Search bergantung pada status pencarian.

- Kelas KMP

Fungsi	Keterangan
KMPStringMatching(string source, string pattern)	Melakukan pencocokan string dengan algoritma KMP
ComputeBorderFunction(string pattern)	Membuat array sebagai border function

- Kelas BM

Fungsi	Keterangan
BMPStringMatching(string source, string pattern)	Melakukan pencocokan string dengan algoritma BM
LastOccurrenceBuilder(string pattern)	Membuat dictionary sebagai tabel <i>last occurrence</i>

- Kelas LD

Fungsi	Keterangan
LevenshteinDistance(string s1, string s2)	Mendapatkan jarak Levenshtein dari dua buah string
FindBestMatch(string imageAscii, string pattern)	Mencari kemiripan terbaik suatu pola terhadap teks

- Kelas Database

Fungsi	Keterangan
Void Connect()	Melakukan koneksi antara program C# dengan basis data MySQL
Void Disconnect()	Melepaskan/menghapus koneksi antara program C# dengan basis data MySQL
MySqlDataReader Select(String query)	Melakukan <i>query</i> “SELECT” pada basis data dengan masukan berupa sintaks “SELECT”
MySqlDataReader SelectAllFingerprint()	Melakukan <i>query</i> “SELECT” ke seluruh data pada tabel sidik_jari
MySqlDataReader SelectAllBiodata()	Melakukan <i>query</i> “SELECT” ke seluruh data di tabel biodata
MySqlDataReader SelectAllBiodata(String name)	Melakukan <i>query</i> “SELECT” ke tabel biodata untuk mendapatkan data dengan nama

	“name”
List<String> SelectAllFingerprintImages()	Mendapatkan seluruh <i>path</i> ke gambar sidik jari yang ada untuk dikonversi ke ASCII
(String, String) FindBiodata(String filename)	Mendapatkan nama alay di tabel biodata jika diberikan nama file gambar sidik jari ke tabel sidik_jari
Dictionary<String, String> ReturnBiodata(String biodataName)	Mendapatkan seluruh data dari tabel biodata dengan nama = “biodataName” berupa nama alay.
void ImportSQL(String filename)	Melakukan <i>import</i> file .sql dari eksternal yang berisi sintaks INSERT ke tabel basis data

- Kelas AlayName

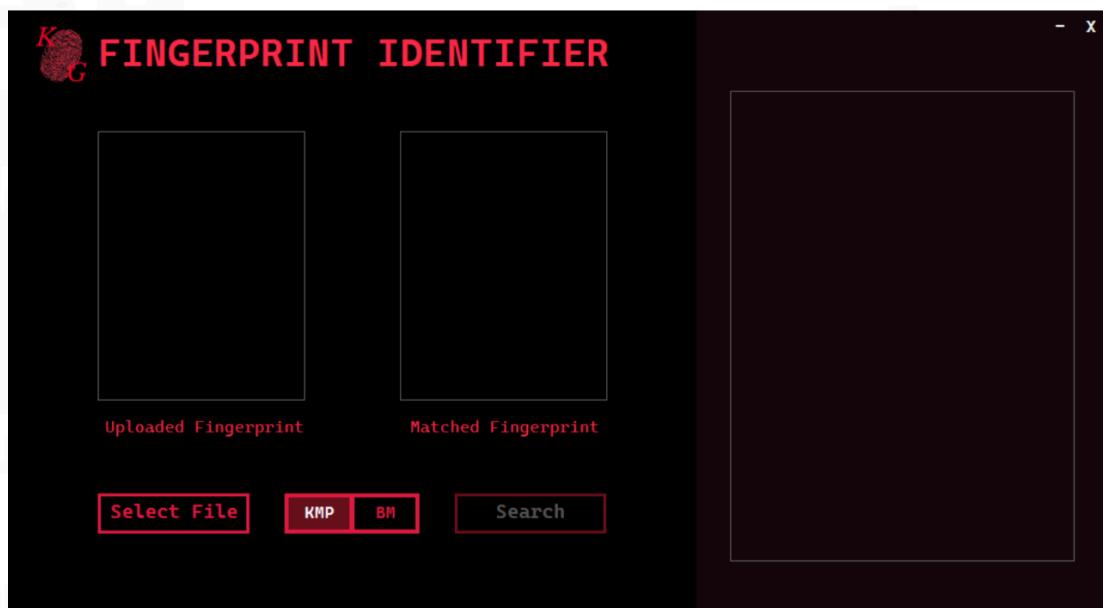
Fungsi	Keterangan
String NormalizeAlayName(String alayName)	Melakukan normalisasi pada string alay dengan mengubah semua angka yang ada menjadi huruf
string CreatePatternFromValidName(string validName)	Membuat sebuah pola regex yang berasal dari nama yang valid
bool IsAlayNameMatch(string alayName, string validName)	Menentukan apakah dua buah nama merupakan nama yang sama dengan membandingkan nama pertama dengan pola regex yang dibentuk dari nama kedua
int LevenshteinDistance(string source, string target)	Menghitung jarak Levenshtein antara dua buah string
String GenerateKey(String str, String key)	Generalisasi key untuk enkripsi menggunakan algoritma Vigenere
String Encrypt(String input, String key)	Melakukan enkripsi dengan mengubah kata input menggunakan key dengan algoritma Vigenere
String Decrypt(String hashed, String key)	Melakukan dekripsi dari sebuah kata yang sudah dienkripsi sebelumnya

- Kelas ImageDBProcessor

Fungsi	Keterangan
String ProcessImage(String filename)	Mengembalikan hasil biodata pencarian jika ditemukan sebuah file yang berisi sidik jari yang cocok dengan yang dimasukkan pengguna

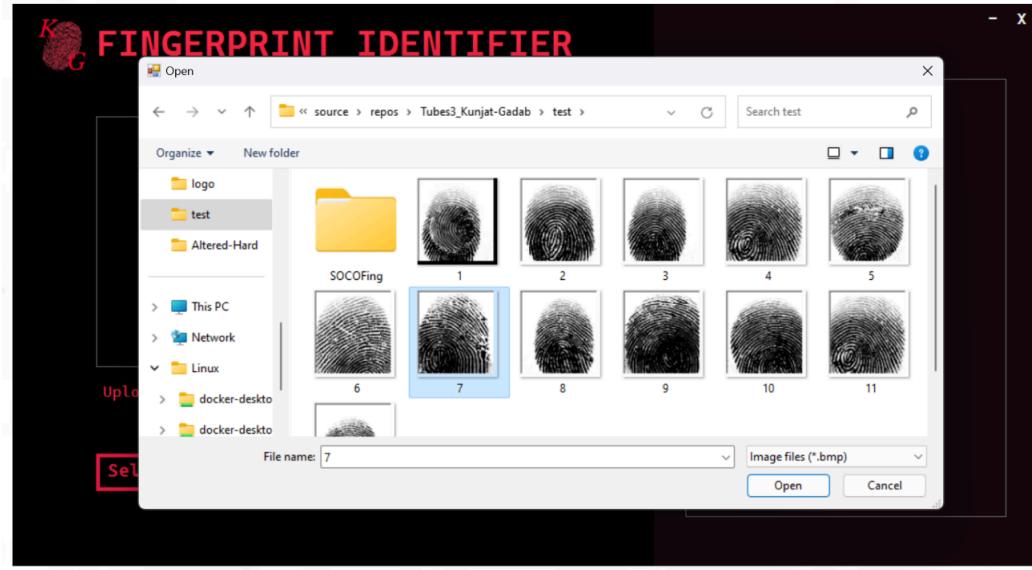
4.2 Tata Cara Penggunaan Program

Berikut adalah tampilan utama antarmuka Aplikasi

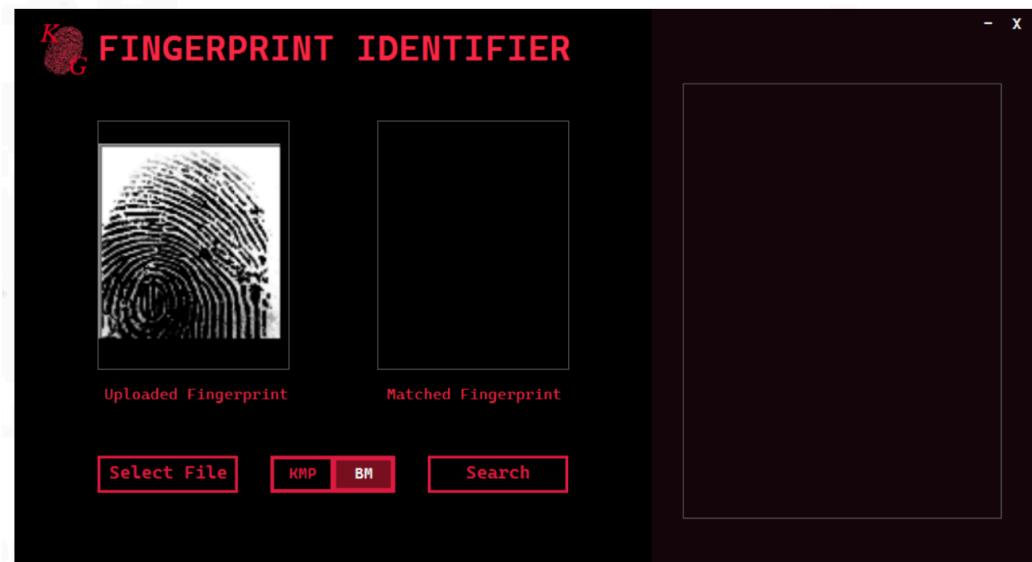


Tata cara penggunaan aplikasi adalah sebagai berikut:

1. Klik tombol Select File untuk memilih gambar sidik jari yang akan digunakan.

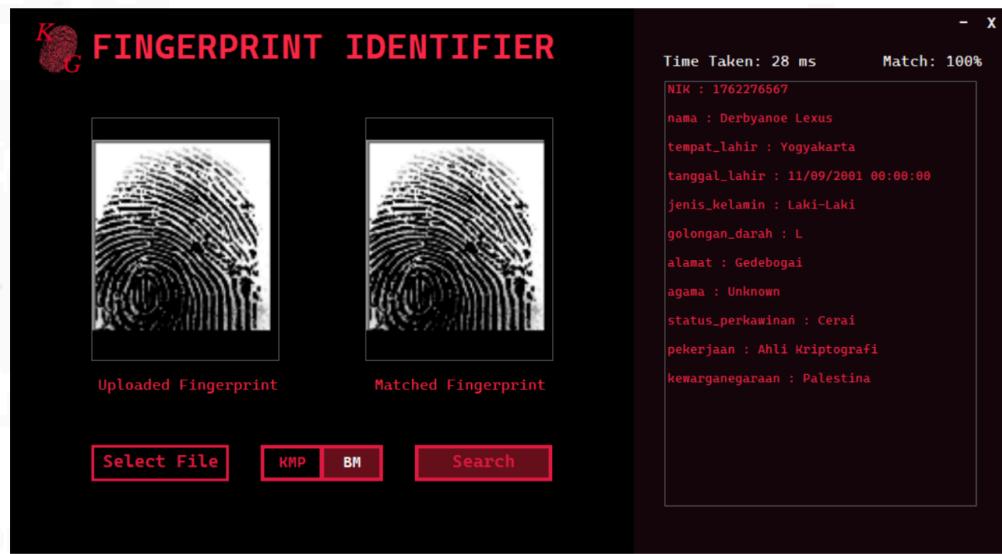


2. Pilih algoritma yang akan digunakan dengan cara klik tombol *toggle*, KMP melambangkan algoritma Knuth-Morris-Pratt, sedangkan BM melambangkan Boyer-Moore.



3. Tekan tombol search untuk memulai pencarian.
4. Aplikasi akan menampilkan hasil pencarian serta waktu lama pencarian. Apabila sidik jari dapat diidentifikasi, Aplikasi juga akan menunjukkan persentase

kemiripan serta Biodata lengkap pemilik sidik jari pada bagian kanan Aplikasi.



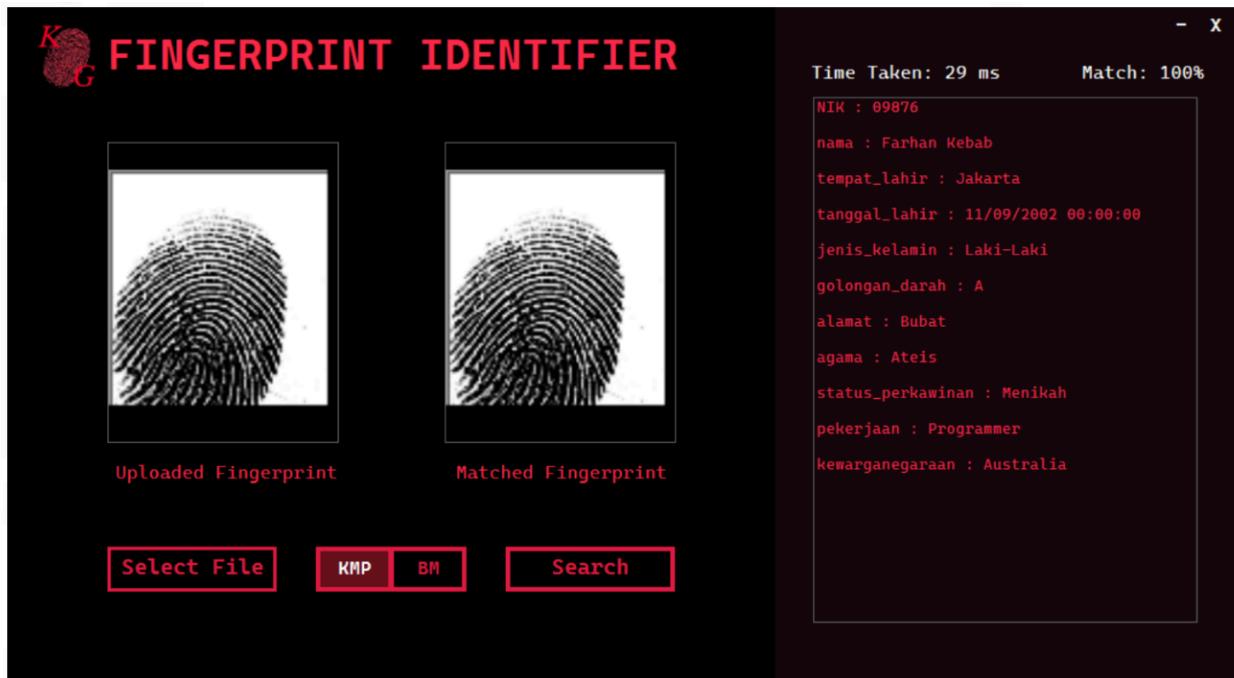
5. Ulangi langkah 1-4 jika ingin mengidentifikasi sidik jari lainnya.
6. Klik tombol X pada kanan atas untuk menutup Aplikasi.

4.3 Hasil Pengujian

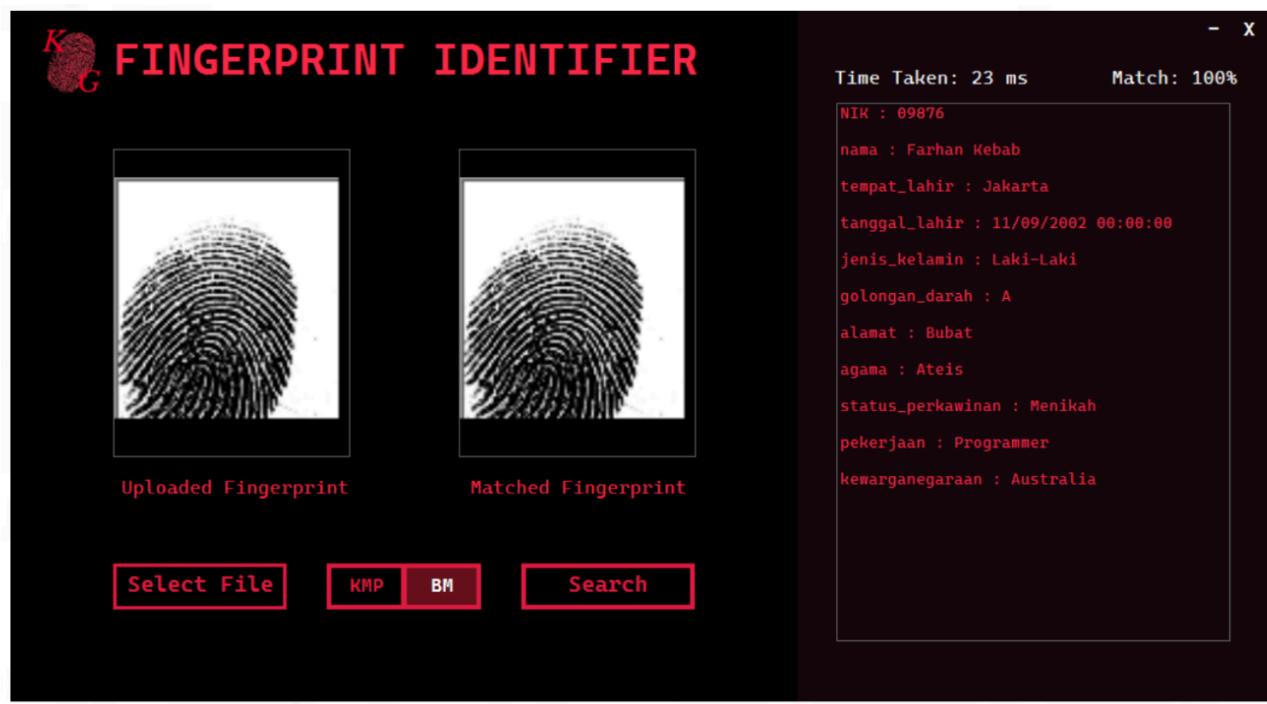
Untuk keperluan pengujian, dilakukan *seeding* data dengan menggunakan file pada folder test/ dengan *seeding* basis data yang digunakan pada dump.sql di folder src/.

4.3.1. Testcase 1

Knuth-Morris-Pratt

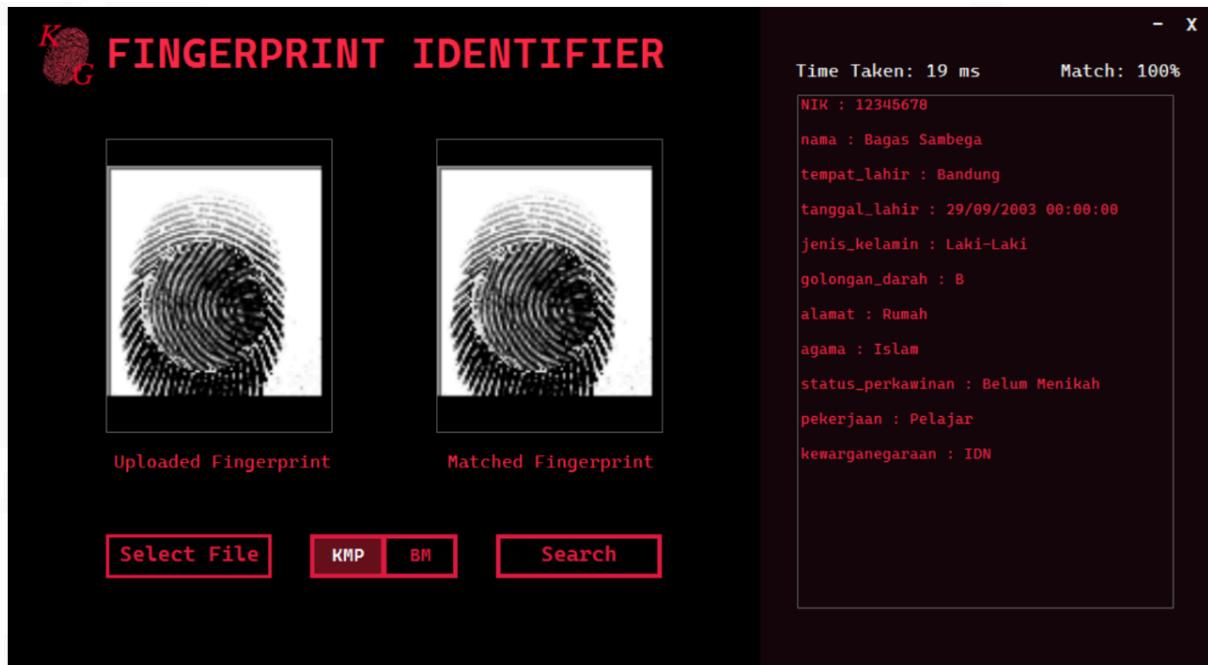


Boyer-Moore

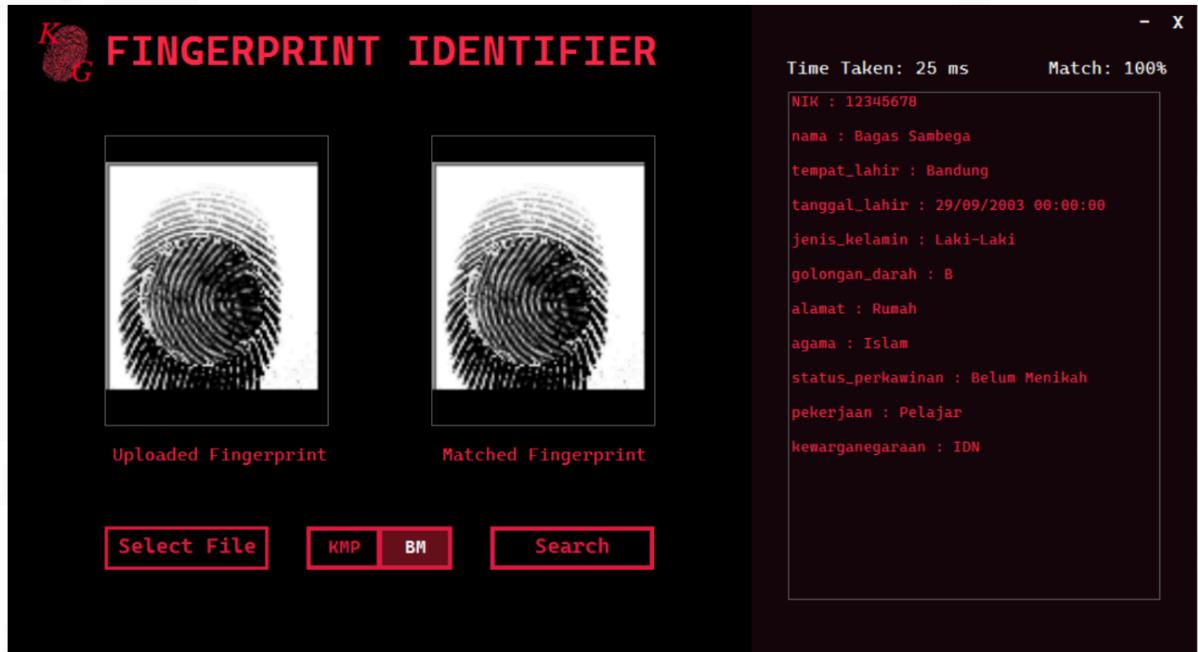


4.3.2. Testcase 2

Knuth-Morris-Pratt

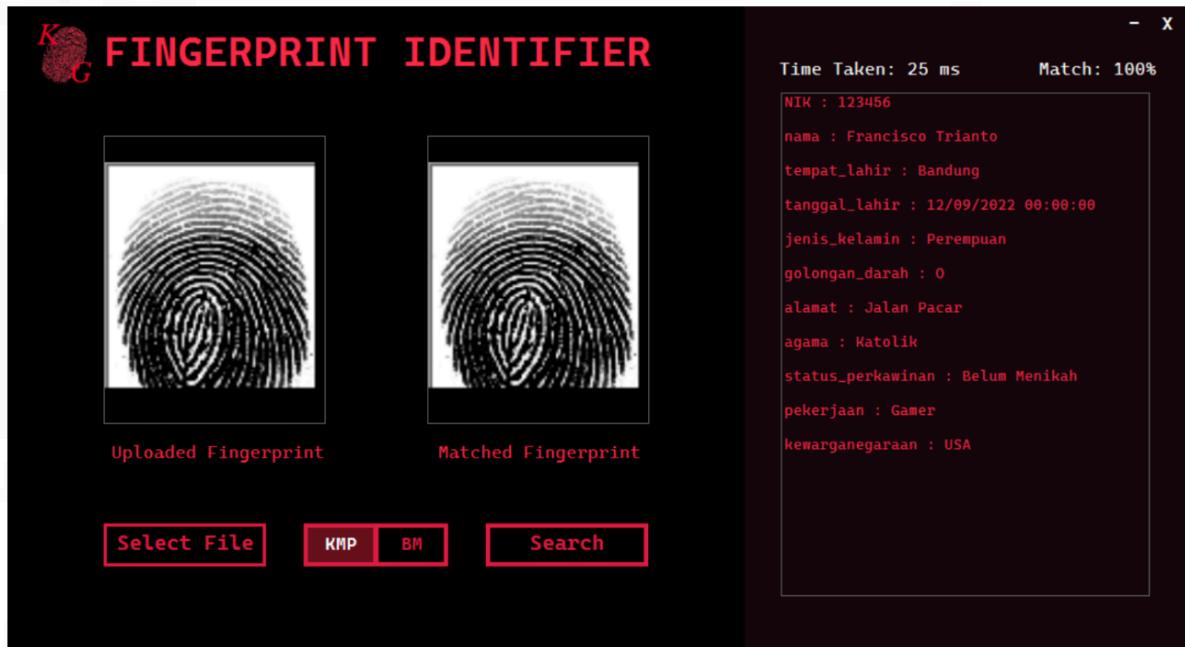


Boyer-Moore

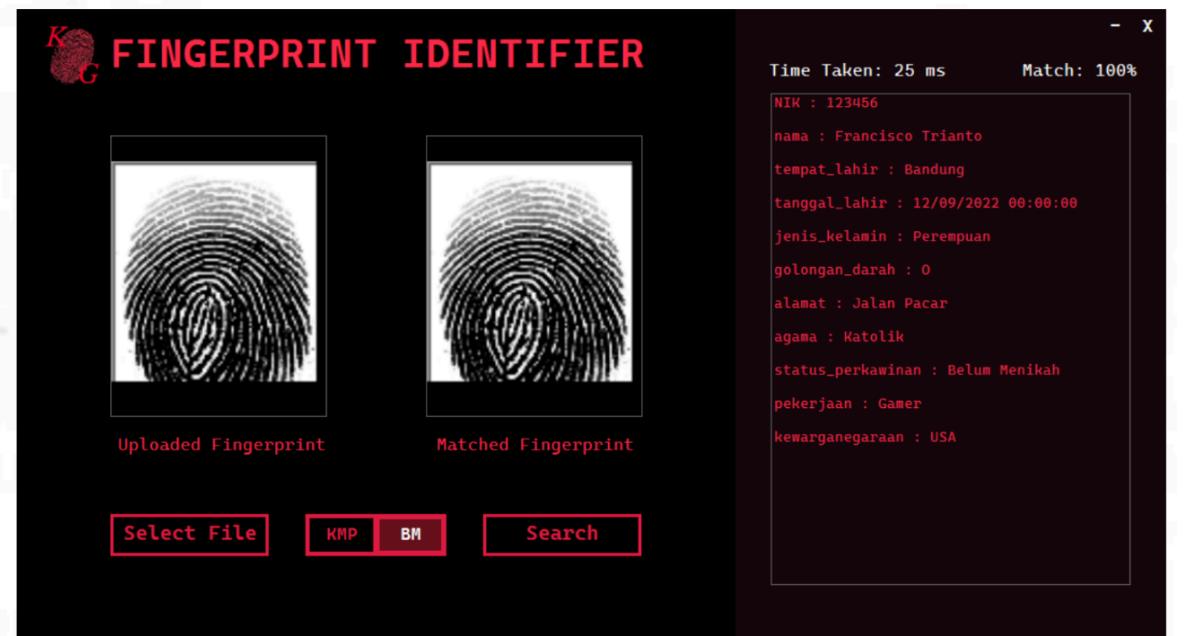


4.3.3. Testcase 3

Knuth-Morris-Pratt

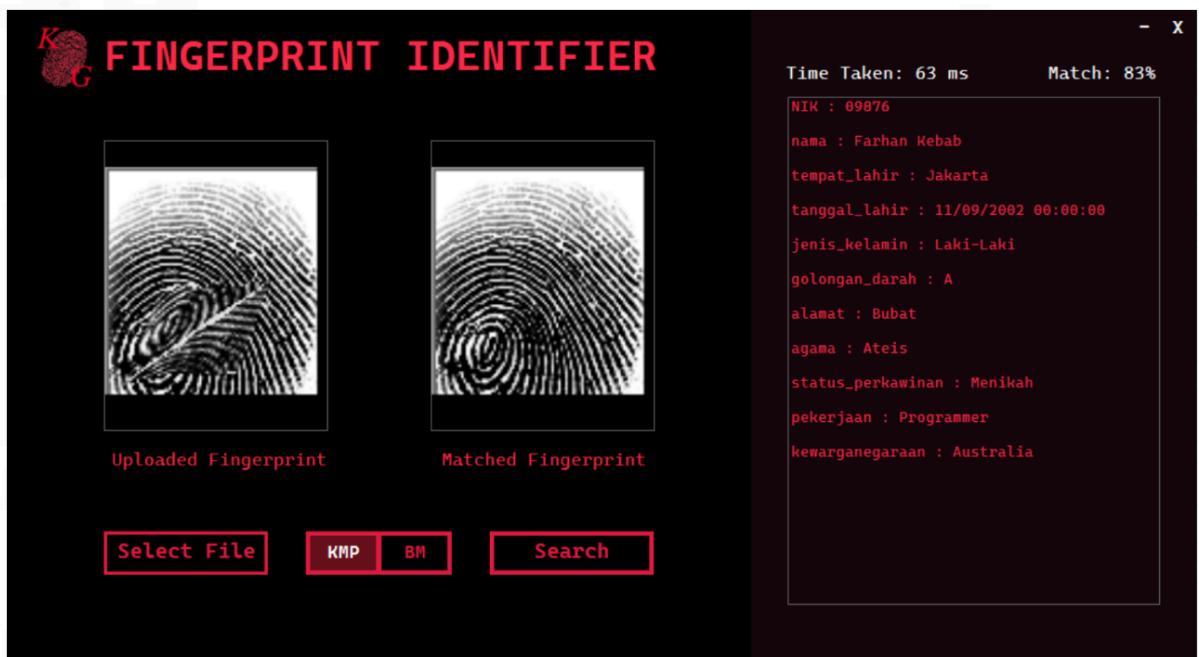


Boyer-Moore

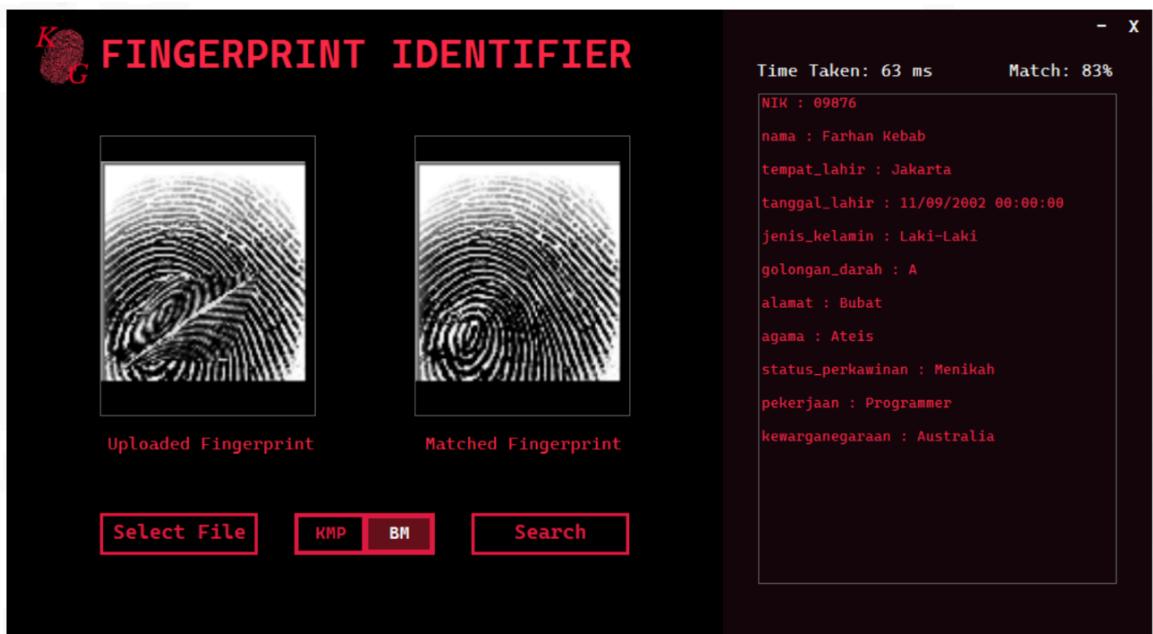


4.3.4. Testcase 4

Knuth-Morris-Pratt

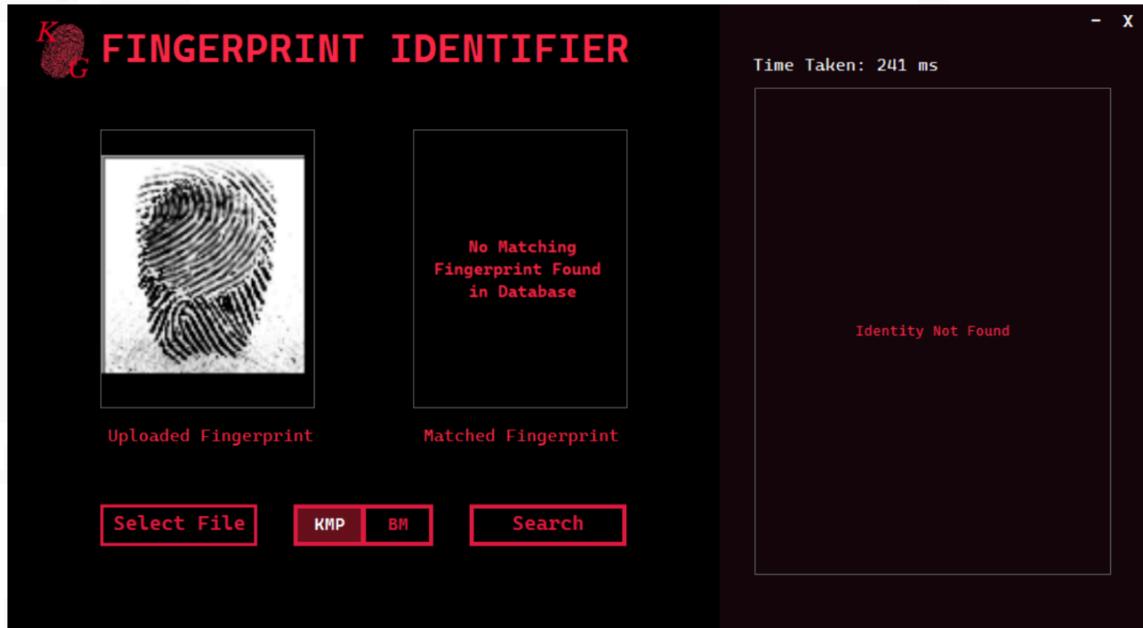


Boyer-Moore

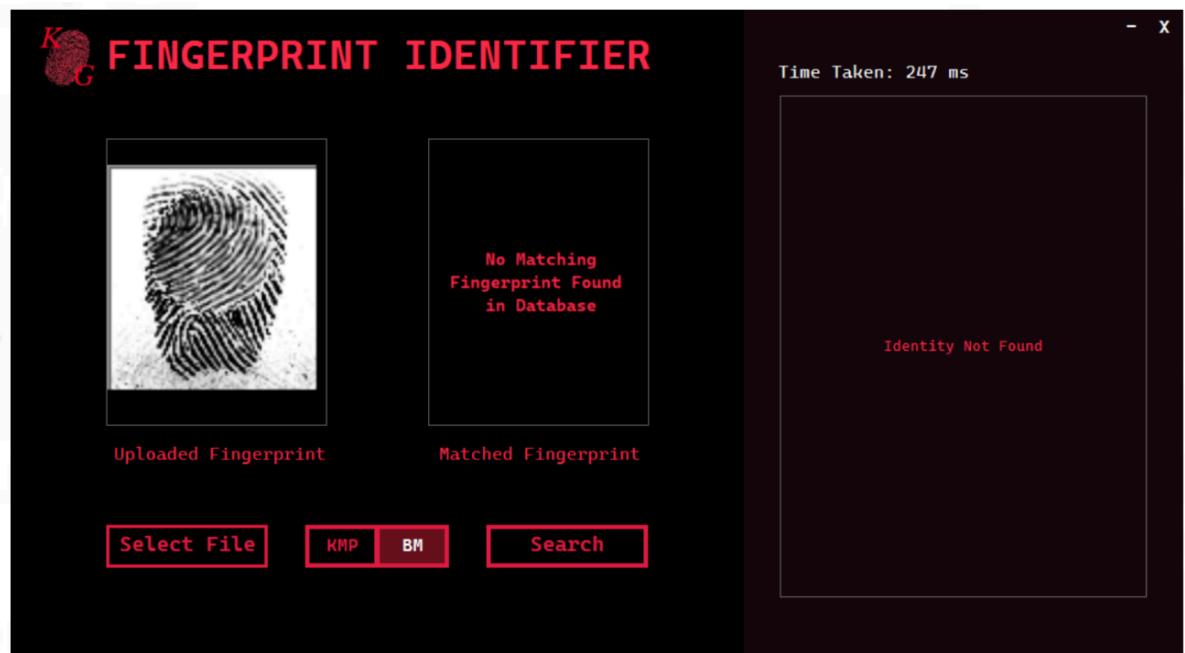


4.3.5. Testcase 5

Knuth-Morris-Pratt

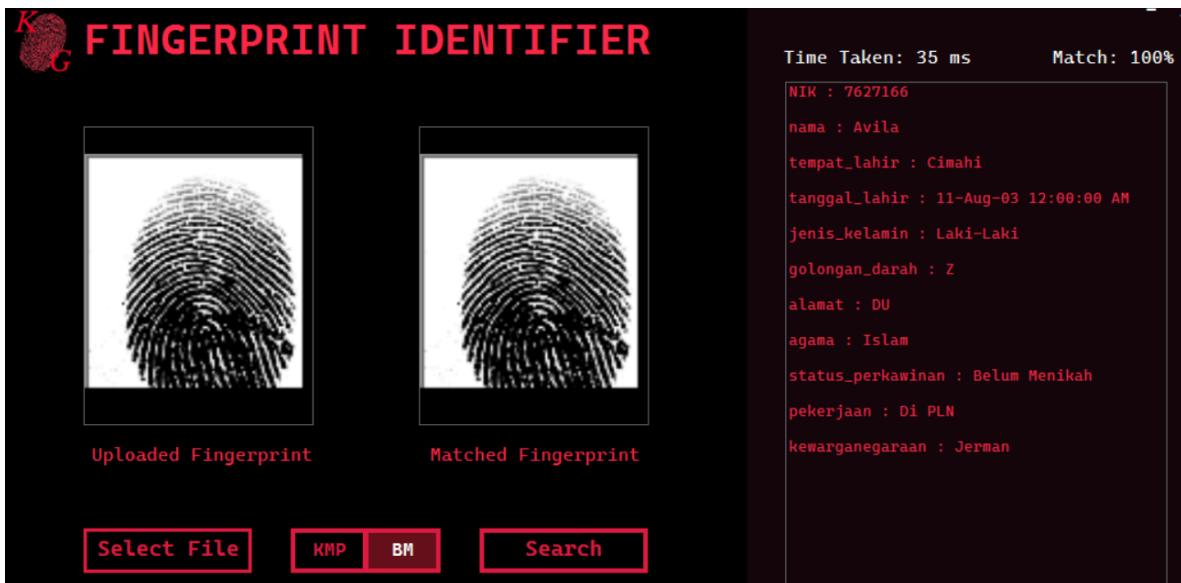


Boyer-Moore

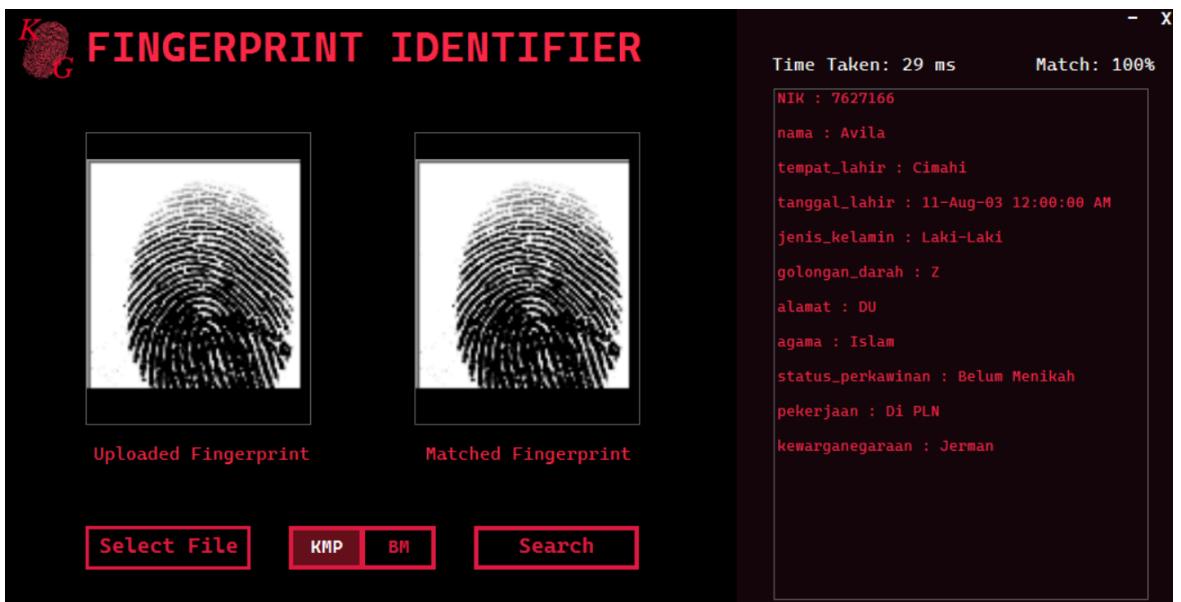


4.3.6. Testcase 6

Boyer Moore



Knuth-Morris-Pratt



4.4 Analisis Hasil Pengujian

Percobaan di atas dilaksanakan sebanyak 6 kali dengan 5 file *fingerprint* terdaftar di database dan sebuah *fingerprint* yang tidak terdaftar di database. Berdasarkan 6 pengujian tersebut, didapat hasil sebagai berikut:

Test	Boyer-Moore (ms)	Knuth-Morris-Pratt (ms)	Kecocokan (%)
------	------------------	-------------------------	---------------

1	29	23	100
2	25	19	100
3	25	25	100
4	63	63	83
5	241	207	-
6	35	29	100

Berdasarkan tabel di atas, dapat dilihat bahwa rata-rata waktu yang dibutuhkan pada algoritma Boyer-Moore lebih tinggi dibandingkan pencocokan dengan menggunakan algoritma Knuth-Morris-Pratt. Sementara itu persentase kecocokan antara kedua algoritma selalu sama. Persentase tersebut pasti selalu benar karena keduanya pasti mencari sebuah *string* yang eksak sama baik pada algoritma Boyer-Moore maupun KMP.

Persentase minimum kecocokan agar sebuah gambar *fingerprint* diidentifikasi sama dengan gambar pada basis data sebesar . Nilai tersebut didasarkan pada

Algoritma Boyer-Moore memiliki waktu eksekusi yang lebih lama dibandingkan algoritma Knuth-Morris-Pratt pada *test case* 1, *test case* 4, *testcase* 5, dan *testcase* 6. Hal ini menunjukkan adanya konsistensi yang cukup tinggi pada waktu eksekusi KMP yang lebih cepat. Pada kasus pencocokan *fingerprint*, sebuah gambar akan dikonversi menjadi *binary* dan kemudian dikonversi lagi menjadi sebuah string ASCII panjang yang merepresentasikan gambar tersebut, sehingga memungkinkan pencocokan string secara linear.

Secara teoritis, kompleksitas terburuk (*worst complexity*) untuk algoritma KMP adalah $O(m + n)$ dan untuk algoritma Boyer-Moore adalah $O(m \cdot n)$. Pada pencocokan kedua ASCII, secara teoritis seharusnya algoritma BM akan lebih cepat secara signifikan dibandingkan KMP. ASCII memiliki 256 karakter yang berbeda, dimana pada pencocokan dengan karakter banyak KMP akan dirugikan oleh besarnya peluang terjadi mismatch, sedangkan BM akan semakin cepat karena lompatan yang bisa ia lakukan lebih jauh. Namun, kasus ini tidak sepenuhnya sesuai dengan teori. Hal ini mungkin diakibatkan oleh waktu *preprocessing* BM yang lebih lama karena banyaknya karakter yang perlu diperiksa. Selain itu, perlu disadari bahwa kompleksitas

KMP lebih stabil pada rata-ratanya yaitu $O(m+n)$, tidak seperti BM yang kurang konsisten. Hal ini mengakibatkan hasil pada kelima test case cenderung menunjukkan bahwa KMP lebih cepat.

Pada basis data, nilai atribut nama pada tabel biodata adalah nilai yang korup dan perlu dikonversi menjadi sebuah nama yang valid dengan menggunakan regex untuk dapat digunakan pada tabel sidik_jari juga. Algoritma yang digunakan pada pencocokan ini adalah pencocokan dengan menggunakan regex, dimulai dengan menggenerasi pola regex dari nama yang valid pada tabel sidik_jari, lalu melakukan pencarian pada tabel biodata jika sebuah nama di tabel biodata memiliki kesesuaian dengan pola regex dari tabel sidik_jari.

Pola regex yang digenerasi adalah pada penggunaan huruf besar dan huruf kecil dan juga penyingkatan huruf vokal. Setiap angka yang terdapat pada nama alay akan dikonversi terlebih dahulu menjadi karakter sesuai aturan berikut: “1” dengan “I”, “3” dengan “E”, “4” dengan “A”, “5” dengan “S”, “6” dengan “G”, “7” dengan “T”, “8” dengan “B”, “9” dengan “G”, dan “0” dengan “O”. Setelah seluruh angka pada string diubah menjadi karakter yang bersesuaian, akan dilakukan pembuatan pola regex untuk nama yang valid. Pola regex yang dibuat akan dikhususkan untuk menemukan penyingkatan karakter vokal pada nama dengan menggunakan pola regex `[[aeiouAEIOU]?]*` pada setiap karakter vokal yang ditemukan di nama yang valid yang menandakan bahwa mungkin saja ada karakter vokal atau tidak, dengan jumlah huruf vokal bisa saja lebih dari 1, dan juga huruf kapital atau kecil yang salah pada nama dengan memakai fungsi IgnoreCase.

Pada basis data di test case 2 sebagai contoh, nama yang tercantum pada tabel biodata adalah “b4645 smbg” dan pada tabel sidik_jari adalah “Bagas Sambega”. Setelah dilakukan normalisasi pada nama alay, nama alay menjadi “bagas smbg”, dan pola regex yang dibentuk adalah

```
[Bb] [[aeiouAEIOU]?]*[Gg] [[aeiouAEIOU]?]*[Ss]\s[Ss] [[aeiouAEIOU]?]*[Mm] [[aeiouAEIOU]?]*[Bb] [[aeiouAEIOU]?]*[Gg] [[aeiouAEIOU]?]*
```

dari nama yang valid. Dapat dilihat bahwa setiap karakter konsonan dan vokal diubah sehingga dapat menjadi kapital maupun kecil, dan huruf vokal diganti polanya menjadi `[[aeiouAEIOU]?]*` untuk memastikan bahwa nama yang mendapat penyingkatan dapat diperiksa. Dengan menggunakan pola yang digenerasi tersebut, maka nama “bagas smbg” dan juga “Bagas Sambega” dapat memiliki kecocokan.

Expression JavaScript ▾ Flags ▾

```
/[Bb][aeiouAEIOU]>*[Gg][aeiouAEIOU]>*[Ss]\s[Ss][aeiouAEIOU]>*[Mm][aeiouAEIOU]>*[Bb][aeiouAEIOU]>*[Gg][aeiouAEIOU]*/g
```

Text Tests NEW 2 matches (0.0ms)

Bagas_Sambega
bagas_smbg

BAB V

KESIMPULAN DAN SARAN

5.1 Kesimpulan

Pada tugas besar 3 IF2211 Strategi Algoritma Semester 2 Tahun Ajaran 2023/2024 ini, kami diminta untuk menyelesaikan permasalahan pencocokan fingerprint. Melalui tugas ini, kami berhasil membuat program yang mampu menyelesaikan permasalahan tersebut dengan menggunakan algoritma Knuth-Morris-Pratt (KMP) dan Boyer-Moore (BM). Kedua algoritma ini digunakan untuk pencocokan string yang efektif, yang dalam konteks ini diaplikasikan pada pencocokan pola fingerprint.

Selain itu, kami juga menerapkan Levenshtein Distance untuk mengukur kemiripan antara dua fingerprint. Levenshtein Distance merupakan algoritma yang menghitung jumlah operasi yang diperlukan untuk mengubah satu string menjadi string lainnya, sehingga sangat berguna dalam mengidentifikasi tingkat kesamaan antara dua fingerprint.

Melalui penerapan ketiga algoritma tersebut, program kami mampu melakukan pencocokan fingerprint dengan tingkat akurasi yang tinggi dan efisiensi yang baik. Penggunaan KMP dan BM memungkinkan pencocokan string dilakukan dengan cepat, sementara Levenshtein Distance memberikan ukuran kemiripan yang lebih fleksibel. Hasil dari implementasi ini menunjukkan bahwa algoritma-algoritma yang kami pilih mampu menangani permasalahan pencocokan fingerprint dengan efektif. Pada Aplikasi ini, kami menyimpulkan bahwa implementasi algoritma Knuth-Morris-Pratt kami cenderung lebih cepat dibandingkan Boyer-Moore.

5.2 Saran dan Tanggapan

Saran untuk kelompok ini diantaranya :

1. Memulai penggerjaan dari lebih awal
2. Meningkatkan komunikasi progress antar anggota

3. Mendalami pengetahuan mengenai bahasa pemrograman C# dengan lebih cepat dan lebih dalam
4. Mencari tahu kelebihan dan kekurangan suatu kakas sehingga tidak kesulitan saat menggunakanya
5. Melakukan desain basis data dengan enkripsi lebih awal agar dapat diselesaikan

5.3 Refleksi

Refleksi yang kami dapatkan dari tugas ini antara lain adalah pentingnya manajemen waktu yang baik untuk menyelesaikan tugas tepat waktu. Selain itu, komunikasi yang efektif antar anggota tim sangat diperlukan untuk memastikan semua anggota memiliki pemahaman yang sama dan dapat berkontribusi secara maksimal. Kami juga menyadari perlunya pemahaman yang lebih mendalam tentang alat dan teknologi yang digunakan, serta pentingnya mempersiapkan desain yang matang sejak awal, termasuk aspek keamanan seperti enkripsi basis data. Pengalaman ini mengajarkan kami untuk lebih proaktif dan kolaboratif dalam menyelesaikan proyek-proyek di masa depan.

LAMPIRAN

Link Repository: https://github.com/NoHaitch/Tubes3_Kunjat-Gadab

Link Video: <https://youtu.be/qCkFmItR0Ww>

DAFTAR PUSTAKA

Rinaldi Munir. Penentuan Rute (Bagian 1). Diakses pada 25 Mei 2024 dari
<https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Route-Planning-Bagian1-2021.pdf>

Rinaldi Munir. Penentuan Rute (Bagian 2). Diakses pada 25 Mei 2024 dari
<https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Route-Planning-Bagian2-2021.pdf>

Geeksforgeeks. KMP algorithm for Pattern Searching. Diakses 7 Juni 2024 dari
<https://www.geeksforgeeks.org/kmp-algorithm-for-pattern-searching/>

Geeksforgeeks. Introduction to Levenshtein distance. Diakses 9 Juni 2024 dari
<https://www.geeksforgeeks.org/introduction-to-levenshtein-distance/>

MDN Docs. Regular Expression. Diakses 8 Juni 2024 dari
https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Regular_expressions

Microsoft. Windows Forms Documentation. Diakses 8 Juni 2024 dari
<https://learn.microsoft.com/en-us/dotnet/desktop/winforms/?view=netdesktop-8.0>