

LAPORAN TUGAS BESAR 3 IF2211
STRATEGI ALGORITMA

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



Dosen Pengampu : Dr. Nur Ulfa Maulidevi, S.T, M.Sc.

Disusun oleh:

Amalia Putri	(13522042)
Angelica Kierra Ninta Gurning	(13522048)
Imanuel Sebastian Girsang	(13522058)

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG

2024

DAFTAR ISI

DAFTAR ISI.....	2
BAB I.....	3
DESKRIPSI TUGAS.....	3
BAB II.....	5
LANDASAN TEORI.....	5
2.1. Deskripsi Algoritma KMP (Knuth Morris Pratt).....	5
BAB III.....	13
ANALISIS PEMECAHAN MASALAH.....	13
3.1 Langkah-Langkah Pemecahan Masalah.....	13
3.2 Proses Pemetaan Masalah.....	18
3.3 Fitur Fungsional dan Arsitektur Aplikasi.....	19
3.4 Contoh Ilustrasi Kasus.....	22
BAB IV.....	23
IMPLEMENTASI DAN PENGUJIAN.....	23
4.1 Spesifikasi Teknis Program.....	23
4.2 Tata Cara Penggunaan Program.....	26
4.3 Hasil Pengujian.....	29
4.4 Analisis Hasil Pengujian.....	45
BAB V.....	47
KESIMPULAN, SARAN, DAN REFLEKSI.....	47
5.1. Kesimpulan.....	47
5.2. Saran.....	47
5.3. Tanggapan.....	48
5.4. Refleksi.....	48
DAFTAR PUSTAKA.....	49
LAMPIRAN.....	50
Repository.....	50
Youtube.....	50

BAB I

DESKRIPSI TUGAS



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

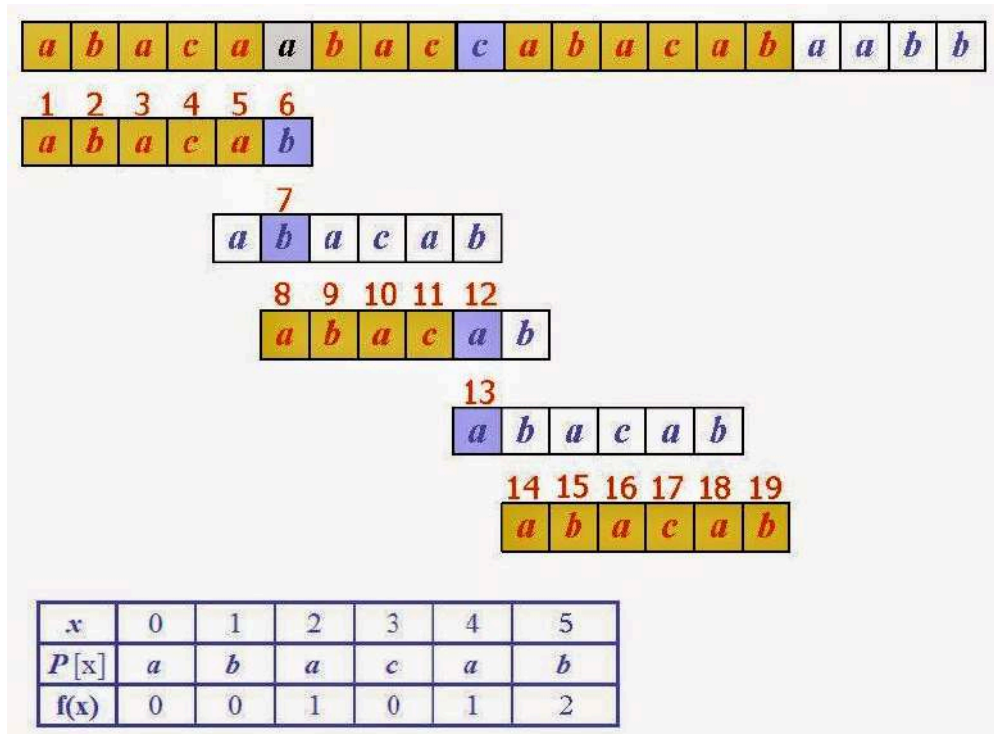
aman, handal, dan mudah digunakan. Sistem ini dapat diaplikasikan di berbagai bidang, seperti kontrol akses, absensi karyawan, dan verifikasi identitas dalam transaksi keuangan. Di dalam Tugas Besar 3 ini, terdapat implementasi sistem yang dapat melakukan identifikasi individu berbasis biometrik dengan 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.

BAB II

LANDASAN TEORI

2.1. Deskripsi Algoritma KMP (Knuth Morris Pratt)



Algoritma KMP merupakan salah satu algoritma dalam pencocokan *string* (*string matching*). Algoritma ini dikembangkan oleh Donald E. Knuth, James H. Morris dan Vaughan R. Pratt. Algoritma ini bekerja dengan cara melewati perbandingan-perbandingan *string* yang tidak diperlukan untuk menghindari besarnya kompleksitas perbandingan. Algoritma ini merupakan modifikasi terhadap algoritma pencarian *Brute Force*. Algoritma KMP memiliki kompleksitas $O(m+n)$, dengan $O(m)$ untuk menghitung fungsi pinggiran dan $O(n)$ untuk proses pencarian string. KMP menggunakan informasi dari pola yang dicari agar tidak terjadi pencarian ulang.

Algoritma ini terbagi menjadi dua tahapan, yaitu Preprocessing (proses membuat array LPS) dan Pencarian (proses pencarian pola menggunakan LPS Array). Berikut langkah-langkah melakukan pencocokan string dengan algoritma KMP:

1. Preprocessing (Membuat LPS Array)

- a. LPS merupakan array yang berisi sebuah prefix yang merupakan juga merupakan suffix untuk sub-pola tertentu.

Contoh : terdapat pola “ABCDABC”

- i. Prefix :
A, AB, ABC, ABCD, ABCDA, ABCDB, ABCDABC
- ii. Suffix : C, BC, ABC, DABC, CDABC, BCDABC
- iii. Prefix = Suffix : ABC

- b. Langkah-langkah membuat LPS adalah sebagai berikut :

- i. Inisialisasi $\text{lps}[0] = 0$, dan dua variabel, $i = 1$ dan $\text{length} = 0$.
- ii. Iterasi melalui pola dari posisi kedua hingga akhir.
- iii. Jika $\text{pattern}[i] == \text{pattern}[\text{length}]$, increment length dan set $\text{lps}[i] = \text{length}$, lalu increment i .
- iv. Jika $\text{pattern}[i] != \text{pattern}[\text{length}]$:
- v. Jika $\text{length} != 0$, set $\text{length} = \text{lps}[\text{length} - 1]$ (tidak mengubah i).
- vi. Jika $\text{length} == 0$, set $\text{lps}[i] = 0$ dan increment i .

- c. Contoh LPS Array:

- i. String : ABABD
 1. $\text{LPS}[0] = 0$, char: A
 2. $\text{LPS}[1] = 0$, A tidak sama dengan B
 3. $\text{LPS}[2] = 1$, A sama dengan A
 4. $\text{LPS}[3] = 1$, B sama dengan B
 5. $\text{LPS}[4] = 0$, D tidak sama dengan A

Hasil LPS = [0,0,1,2,0]

2. Pencarian menggunakan LPS Array yang telah dibuat

- a. Langkah:

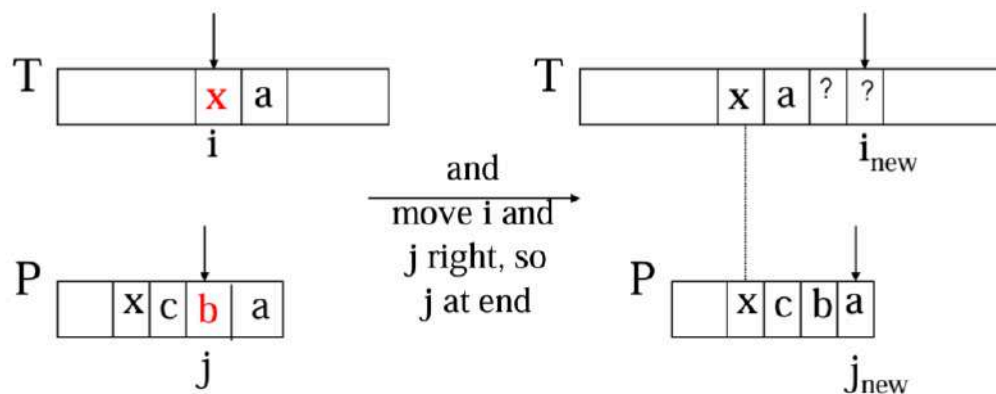
- i. Iterasi melalui teks dengan dua variabel, i (untuk teks) dan j (untuk pola).
- ii. Jika $\text{pattern}[j] == \text{text}[i]$, increment i dan j .

- iii. Jika j sama dengan panjang pola, pola ditemukan. Catat indeks posisi mulai di $i - j$, lalu set $j = \text{lps}[j - 1]$.
- iv. Jika $\text{pattern}[j] \neq \text{text}[i]$ dan $j \neq 0$, set $j = \text{lps}[j - 1]$ tanpa mengubah i .
- v. Jika $\text{pattern}[j] \neq \text{text}[i]$ dan $j == 0$, increment i .

2.2. Deskripsi Algoritma BM (Boyer-Moore Algorithm)

Algoritma Boyer-Moore menggunakan dua teknik, yaitu *The looking-glass technique* dan *The character-jump technique*. Looking Glass Technique merupakan proses pencarian P (pattern) di dalam T (teks) dengan mencari dari bagian belakang P . Character Jump Technique merupakan teknik ketika terjadi ketidak samaan saat $T[i] == x$, maka $P[j]$ tidak sama dengan $T[i]$. Terdapat 3 kasus, untuk menentukan seberapa jauh P (pattern) digeser saat terjadi ketidaksesuaian dengan T . Kasus-kasus tersebut antara lain :

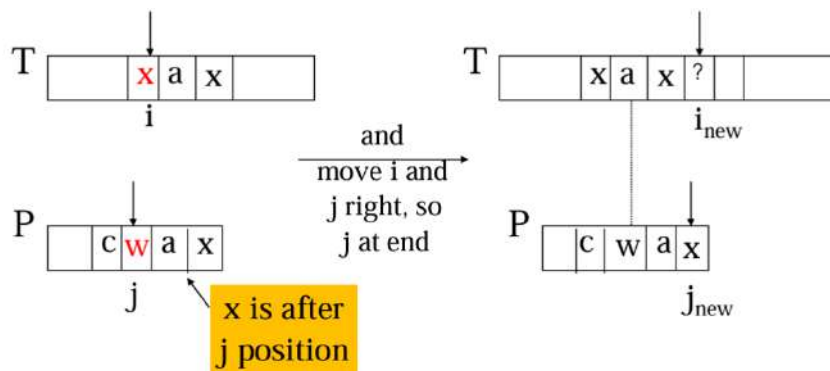
1. Jika P mengandung x di suatu tempat, maka coba geser P ke kanan untuk menyelaraskan kemunculan terakhir dari x dalam P dengan $T[i]$.
 - a. Ketika terjadi ketidaksesuaian pada karakter $T[i]$ dalam teks dan $P[j]$ dalam pola, kita mencari kemunculan terakhir karakter $T[i]$ dalam pola sebelum posisi j .
 - b. Tindakan: Geser pola ke kanan sehingga karakter $T[i]$ dalam teks sejajar dengan kemunculan terakhirnya dalam pola.



2. Jika P mengandung x di suatu tempat, tetapi pergeseran ke kanan ke kemunculan terakhir tidak mungkin, maka geser P ke kanan 1 karakter ke $T[i+1]$.

a. Jika pergeseran untuk menyelaraskan kemunculan terakhir karakter yang tidak cocok tidak mungkin (misalnya, karena posisi yang tidak sesuai dalam pola atau karakter tidak ditemukan), geser pola ke kanan satu karakter.

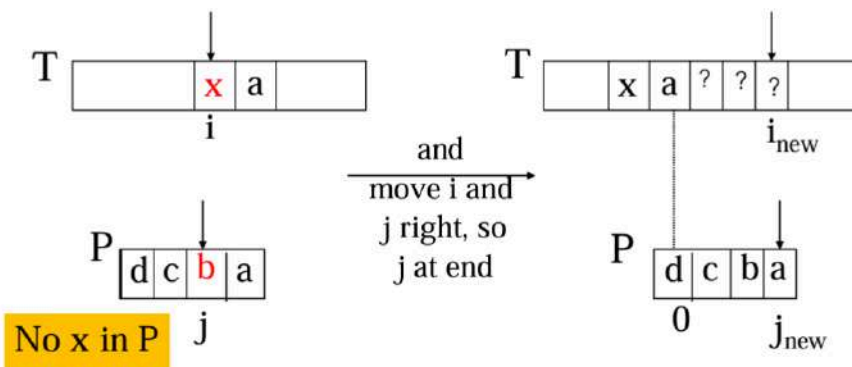
b. Tindakan: Geser pola ke kanan satu posisi.



3. Jika kasus 1 dan 2 tidak berlaku, maka geser P untuk menyelaraskan $P[0]$ dengan $T[i+1]$

a. Jika tidak ada kemunculan terakhir dari karakter yang tidak cocok dalam pola dan pergeseran satu karakter tidak mengatasi ketidaksesuaian, geser pola sehingga awal pola sejajar dengan karakter berikutnya dalam teks.

b. Tindakan: Geser pola ke kanan sehingga $P[0]$ sejajar dengan $T[i+1]$.



2.3. Deskripsi Regex

Regex merupakan singkatan dari Regular Expression. Regex merupakan sebuah teks (string) yang mendefinisikan sebuah pola pencarian sehingga dapat membantu kita untuk melakukan matching (pencocokan), locate (pencarian), dan manipulasi teks. Dengan kombinasi karakter khusus dan simbol, regex memungkinkan pengguna untuk mencari, memodifikasi, dan memanipulasi string berdasarkan pola tertentu. Pola-pola ini bisa berupa kata-kata spesifik, karakter numerik, huruf besar atau kecil, dan bahkan format kompleks seperti alamat email atau nomor telepon.

Dalam dunia pemrograman dan pemrosesan teks, regex memiliki banyak kegunaan. Contohnya, dalam pemrograman web, regex sering digunakan untuk validasi input pengguna, memastikan bahwa data yang dimasukkan sesuai dengan format yang diharapkan seperti alamat email atau kode pos. Dalam pemrosesan teks dan data, regex mempermudah menemukan dan mengganti teks dengan cepat dan efisien, yang sangat bermanfaat untuk mengedit dokumen besar atau menganalisis log file. Selain itu, regex juga digunakan dalam berbagai tools dan bahasa pemrograman seperti Perl, Python, JavaScript, dan lainnya, memberikan fleksibilitas dan kekuatan dalam pengolahan string dan teks.

Keunggulan regex terletak pada kemampuannya untuk mendefinisikan pola yang kompleks dengan cara yang ringkas dan mudah dibaca. Dengan fitur-fitur seperti pencocokan karakter individu, kelompok karakter, pengulangan, dan opsi pencocokan alternatif, regex menjadi alat penting bagi pengembang perangkat lunak, analis data, dan profesional lain yang rutin bekerja dengan teks dan string.

2.4. Teknik Pengukuran Persentase Kemiripan

Pada tugas besar kali ini, teknik pengukuran yang digunakan adalah Levenshtein Distance. Levenshtein Distance adalah teknik yang digunakan untuk mengukur perbedaan antara dua string. Teknik ini mengukur jumlah minimum operasi pengubahan yang diperlukan untuk mengubah satu string menjadi string lainnya. Semakin tinggi, nilai levenshtein, maka semakin berbedanya antar string. Pada Levenshtein Distance, ada tiga operasi yang berlangsung, yaitu:

1. Penggantian (Replace)
2. Penghapusan (Delete)
3. Pemasukkan (Insert)

Levenshtein Distance akan dihitung menggunakan sebuah matrix, yang bersesuaian dengan panjang *string* yang akan dibandingkan. Langkah-langkah untuk mengisi matriks Levenshtein sebagai berikut:

1. Inisialisasi

Baris pertama dan kolom pertama diisi dengan nilai dari 0 hingga masing-masing m dan n, menunjukkan jarak dari string kosong ke setiap karakter dalam string yang bersesuaian.

2. Iterasi

Untuk setiap sel (i, j) di dalam matriks, di mana i berada di antara 1 dan m, dan j berada di antara 1 dan n, kita lakukan langkah-langkah berikut:

- Jika karakter ke-i dari string pertama sama dengan karakter ke-j dari string kedua, maka nilai sel (i, j) sama dengan nilai sel (i-1, j-1). Tidak ada operasi yang diperlukan untuk memperbaiki karakter yang sama
- Jika karakter ke-i dari string pertama tidak sama dengan karakter ke-j dari string kedua, maka nilai sel (i, j) adalah minimum dari tiga operasi (Replace, Delete, Insert, Update)

3. Output

Nilai di sudut kanan bawah matriks (sel (m, n)) adalah Levenshtein Distance antara kedua string

2.5. Enkripsi AES

Metode AES adalah algoritma enkripsi simetris yang digunakan untuk mengenkripsi dan mendekripsi data dalam layanan pesan E2E. Metode ini pertama kali diperkenalkan pada tahun 1998. Metode ini meningkatkan ukuran blok sebelumnya dari 64 bit menjadi 128 bit. Metode ini mengambil plaintext dengan ukuran 128 bit atau 16 byte dan menghasilkan sebuah kunci. Panjang kunci bervariasi dari 128, 192, atau 256 bit. Khusus pada tugas besar ini akan menggunakan AES-256.

Metode AES merupakan enkripsi simetris, artinya akan ada satu kunci yang digunakan untuk melakukan enkripsi dan juga deskripsi. Secara umum, ada beberapa tahapan dalam melakukan enkripsi metode AES, antara lain sebagai berikut:

1. Byte Sub Transformation

Transformasi Byte Sub menggunakan S-Box untuk operasinya. S-Box adalah matriks 16×16 dari nilai byte. Matriks ini berisi permutasi yang telah dihitung sebelumnya dari semua kemungkinan 256 nilai 8-bit.

2. Shift Row Transformation

Dalam proses ini, baris-baris keadaan digeser secara melingkar ke kiri dengan beberapa offset. Baris 0 tidak digeser, penggeseran dimulai dari baris 1. Baris akan digeser sebanyak (n) byte, dengan n adalah indeks barisnya.

3. Mix Column Transformation

Proses ini diwakili sebagai mix-cols. Proses ini tidak dilakukan pada putaran terakhir. Transformasi shift row menghasilkan matriks keadaan. Dalam Transformasi Kolom Campuran, matriks keadaan dianggap sebagai matriks 4×4 . Kemudian operasi perkalian dilakukan. Perkalian ini melibatkan mengalikan setiap elemen dalam matriks keadaan dengan elemen yang sesuai dalam matriks kolom campuran yang sudah ditentukan, biasanya dari Galois Field.

4. Add Round Key

Pada setiap putaran, satu set kunci putaran akan diperoleh dari tiga operasi lainnya. Operasi XOR akan dilakukan antara kunci putaran sebelumnya dengan matriks keadaan baru. Hasilnya akan menjadi kunci akhir yang dihasilkan.

Untuk proses dekripsi merupakan, proses yang mirip dengan enkripsi hanya saja dilakukan inversenya.

2.6. Aplikasi Yang Dibangun

Aplikasi yang dibangun merupakan aplikasi berbasis GUI dan menggunakan bahasa C#. C# adalah sebuah bahasa pemrograman yang dirancang oleh Microsoft sebagai bagian dari platform .NET. Bahasa ini memiliki sintaks yang mirip dengan C++ dan Java, dengan penekanan pada pengelolaan memori yang aman dan dukungan yang kuat untuk pemrograman berorientasi objek. C# biasanya digunakan untuk pengembangan aplikasi berbasis Windows, aplikasi web dengan ASP.NET, pengembangan game dengan platform .NET seperti Unity, serta aplikasi perangkat lunak lainnya.

Pada sistem ini aplikasi dibangun menggunakan *framework* AvaloniaUI. *Framework* ini memfasilitasi pembuatan aplikasi menggunakan XAML. Dengan menggunakan AvaloniaUI, aplikasi memiliki kemampuan untuk beroperasi di berbagai sistem operasi, seperti Windows, macOS, dan Linux.

BAB III

ANALISIS PEMECAHAN MASALAH

3.1 Langkah-Langkah Pemecahan Masalah

Permasalahan sistem identifikasi sidik jari dapat diselesaikan dalam 4 tahap utama yaitu konversi data menjadi format yang lebih efektif, pencocokan data, pencarian detail data, serta penampilan data. Keempat langkah tersebut merupakan abstraksi umum yang dibuat sebagai tujuan akhir yang ingin dicapai. Secara detail, dilakukan hal-hal berikut untuk menyelesaikan masalah :

1. Mempelajari syntax C# secara umum terutama untuk membuat GUI serta memproses gambar
2. Mempelajari algoritma KMP, Boyer Moore, serta Levenshtein Distance
3. Membuat aplikasi GUI dalam bahasa C# untuk menerima masukan dari pengguna
4. Memetakan permasalahan menjadi elemen permasalahan *string matching*
5. Menentukan strategi *string matching* yang tepat
6. Menentukan strategi *handling* data-data korup berupa bahasa Alay
7. Menghubungkan algoritma yang ada dengan basis data untuk menjadi sistem identifikasi sidik jari yang terintegrasi

3.1.1. Konversi data

Perbandingan citra dengan citra menggunakan algoritma KMP maupun Boyer Moore jelas tidak dapat dilakukan secara langsung. Maka dari itu, diperlukan suatu cara untuk mengubah citra menjadi suatu format yang dapat dilakukan *pattern matching*. Maka dari itu, langkah pertama yang harus dilakukan adalah mengubah citra yang ada menjadi bentuk biner 0 dan 1 mengkonversinya lagi menjadi Karakter ASCII terkait agar kemudian bisa dilakukan perbandingannya. Berdasarkan riset yang telah kami lakukan dari gambar di dataset, kelompok kami memutuskan untuk

mengambil 88 pixel pertama dari 3 posisi berbeda untuk setiap gambar masukan dengan prioritas pengecekan sebagai berikut:

- a. Baris ke $0.75 * \text{tinggi gambar}$
- b. Baris ke $0.9 * \text{tinggi gambar}$
- c. Baris ke $0.25 * \text{tinggi gambar}$

Setelah didapatkan 88 representasi pixelnya, maka proses dilanjutkan dengan mengubah seluruh pixel terkait menjadi binary 1 0. Hal ini dilakukan dengan menentukan nilai gray dari masing-masing pixel. Bila dia lebih condong ke arah hitam maka akan dibuat ke angka 0, jika tidak maka dibuat ke angka 1.

Setelah itu, akan dilakukan konversi dari binary terkait menjadi karakter ASCII. Konversi ini dilakukan untuk mempermudah dan mempercepat komputasi. Konversi dilakukan dengan melakukan pengubahan setiap 8 pixel ke decimal dan kemudian dari decimal ke ASCII. Setelah mendapatkan representasinya, data siap untuk dibandingkan dengan setiap citra yang ada di basis data. Citra yang ada di basis data diproses dengan cara yang sama, tetapi tidak hanya diproses 88 pixel melainkan seluruh gambar.

Untuk menambah sekuritas, data pada database akan dienkripsi menggunakan enkripsi AES. Data yang sudah ada dalam database akan diubah menjadi hasil enkripsi. Berikut merupakan langkah-langkah pemecahan masalah dalam enkripsi AES:

1. Setiap data (kecuali Jenis Kelamin) akan dibaca sebagai string
2. String tersebut akan dikonversi menjadi byte array
3. Jika panjang string bukanlah 16 byte maka, program akan membuat beberapa blok-blok 16 byte untuk string tersebut
4. Jika panjang kurang dari 16 byte array maka akan dilakukan padding dengan cara menambahkan spasi
5. Selanjutnya byte array akan dipetakan menjadi matrix 4×4

6. Setelah itu akan diproses secara AES (seperti pada dasar teori)
7. Untuk melakukan dekripsi, diperlukan kunci yang sama seperti proses enkripsi

3.1.2. Pencocokan Data

Pada tahap pencocokan data, akan dilakukan iterasi ke seluruh gambar di basis data untuk dilakukan *string matching* dengan algoritma KMP atau BM sesuai dengan masukan pengguna. Apabila ditemukan *pattern* terkait di suatu gambar, maka iterasi akan dihentikan. Jika tidak ditemukan *exact match*, maka akan digunakan algoritma *levenshtein distance* ke seluruh gambar yang ada untuk mencari persentase kemiripan tertinggi diantara semuanya. Jika persentase kemiripan tertinggi diatas 60%, maka gambar bisa dibilang sama.

Pada tugas besar ini, algoritma KMP akan melakukan hal-hal berikut untuk menentukan apakah suatu citra sama dengan citra lainnya. Misal masukan pengguna disebut sebagai Citra 1 dan gambar di basis data disebut sebagai Citra 2. Algoritma akan melakukan hal-hal berikut:

- a. Menerima 11 Karakter ASCII representasi Citra 1 serta Representasi ASCII penuh dari Citra 2.
- b. Membuat *border function* sesuai dengan algoritma yang telah dijelaskan di dasar teori menggunakan Citra 1 sebagai *pattern*.
- c. Melakukan pencarian *pattern* Citra 1 pada Citra 2 dengan menggunakan algoritma KMP. Algoritma KMP dimulai dengan menaruh dua pointer yaitu pada awal pattern (i) dan juga awal teks (j). Selanjutnya, dilakukan iterasi terhadap teks dan juga pattern hingga mencapai ujung *pattern* atau hingga ditemukan *pattern* yang sesuai dengan menggunakan 3 kasus berikut ini :
 - i. Jika nilai teks[i] == nilai pattern[j], maka periksa apakah i adalah ujung teks. Jika iya, maka fungsi mengembalikan nilai indeks terkait. Jika tidak, lanjutkan pencarian dengan menambah i dan juga j.

- ii. Jika nilai teks[i] tidak sama dengan pattern[j], tetapi nilai $j > 0$, artinya terjadi *missmatch* namun sebelumnya sudah sempat terjadi *match*. Pada kasus ini, maka digunakan tabel *border function* untuk menentukan nilai J yang baru yaitu di posisi prefiks terbesar dari *pattern* [0...n] yang juga merupakan sufiks dari *pattern* [1...n] dengan n adalah posisi *missmatch* - 1. Hal ini untuk menentukan seberapa jauh pengecekan boleh dilewati.
- iii. Jika bukan dua kondisi sebelumnya, maka tambah i dengan 1 dan lakukan hal ini lagi hingga i sudah mencapai ujung dari teks.
- d. Jika hingga ujung teks belum ditemukan *pattern*, maka dipastikan kedua gambar tidak sama.
- e. Apabila ditemukan *pattern* pada Citra 2, maka disimpulkan bahwa kedua citra adalah sama (Exact Match).

Untuk algoritma Booyer Moore, akan dilakukan hal-hal berikut untuk menentukan apakah suatu *pattern* terdapat di dalam teks:

- a. Menerima 11 Karakter ASCII representasi Citra 1 serta Representasi ASCII penuh dari Citra 2.
- b. Membuat *Last Occurence Table* sesuai dengan algoritma yang telah dijelaskan di dasar teori dengan menggunakan Citra 1 sebagai *pattern* dan juga 256 ASCII Sebagai seluruh alfabet.
- c. Melakukan pencarian pattern Citra 1 pada Citra 2 dengan menggunakan algoritma Boyer Moore. Algoritma ini dimulai dengan menggunakan pointer di posisi terjauh teks sepanjang pattern [i] dan juga di ujung pattern [j]. Algoritma ini melakukan iterasi mundur (pointer mundur) hingga ditemukan pattern atau hingga ujung dari teks dengan setiap iterasinya. Ketika terjadi *missmatch*, ada 3 kemungkinan yang bisa dilakukan :
 - i. Jika nilai teks[i] pada *Last Occurence Table* $\neq -1$ dan posisinya $< i$, maka geser i sejumlah panjang *pattern* - angka pada tabel dan pindah j ke ujung *pattern*.

- ii. Jika ditemukan teks[i] pada *Last Occurence Table* tetapi posisinya $> i$, maka geser i sejumlah panjang *pattern* - j dan pindah j ke ujung *pattern*.
- iii. Jika tidak ditemukan teks[i] pada *Last Occurence Table* (-1) maka langsung geser i sebanyak panjang *pattern*
- d. Apabila ditemukan *pattern* pada Citra 2, maka disimpulkan bahwa kedua citra adalah sama (Exact Match).

Untuk algoritma *Levenshtein Distance*, gambar masukan pengguna akan dicari representasi gambar penuhnya untuk kemudian dicari berapa seberapa mirip kedua gambar yang sedang dibandingkan. Cara kerjanya adalah sebagai berikut :

1. Inisialisasi Matriks

Dibuat sebuah matriks dua dimensi dengan ukuran (panjang string pertama + 1) x (panjang string kedua + 1). Baris pertama dari matriks ini diisi dengan nilai dari 0 hingga panjang string pertama. Hal ini menggambarkan jumlah operasi penghapusan yang diperlukan untuk mengubah string pertama menjadi string kosong. Kolom pertama dari matriks ini diisi dengan nilai dari 0 hingga panjang string kedua. Hal ini menggambarkan jumlah operasi penyisipan yang diperlukan untuk mengubah string kosong menjadi string kedua.

2. Pengisian Matriks

Matriks kemudian diisi dengan cara membandingkan karakter-karakter dari kedua string. Jika karakter pada posisi tertentu dari kedua string sama, maka biaya substitusi adalah 0. Jika berbeda, biaya substitusi adalah 1. Setiap elemen matriks diisi dengan nilai minimum dari tiga kemungkinan

- a. Nilai di atasnya ditambah 1 (penghapusan karakter dari string pertama).
- b. Nilai di sebelah kirinya ditambah 1 (penyisipan karakter ke string pertama).

c. Nilai di diagonal kiri atas ditambah biaya substitusi (penggantian karakter).

3. Menghitung Jarak

Setelah matriks terisi penuh, nilai di sudut kanan bawah matriks adalah Levenshtein Distance antara dua string tersebut. Nilai ini menggambarkan jumlah minimum operasi yang diperlukan untuk mengubah satu string menjadi string lainnya. Setelah mendapat jarak keduanya, hasilnya dibagi dengan panjang maksimum dari kedua teks dan dari situ didapat berapa persentase perbedaannya. Dengan mengurangi 100 dengan $100 * \text{angka perbedaan tersebut}$, didapatlah persentase kemiripannya.

Jika hingga dari semua metode yang digunakan tidak ditemukan satupun gambar yang memenuhi constraint terkait, maka aplikasi akan menunjukkan pesan kesalahan bahwa tidak ada gambar yang mirip.

3.1.3. Pencarian Detail Data

Setelah mendapatkan foto yang memenuhi kriteria pencocokan, maka selanjutnya adalah tahap mencari data detail dari orang terkait. Hal ini dilakukan dengan melakukan iterasi ke seluruh gambar data orang yang ada di basis data dan mencari nama yang cocok. Karena kami melakukan enkripsi serta pengubahan ke nama alay, maka hal yang dilakukan adalah melakukan dekripsi terhadap data nama di setiap *record* biodata di basis data, dan melakukan pencocokan *Regular Expression* dengan nama yang ada untuk mendapatkan seluruh data detail orang sesungguhnya.

3.1.4. Penampilan Data

Setelah ditemukan data orang terkait, informasi akan di *passing* ke GUI untuk kemudian ditampilkan kepada pengguna. Penampilan data yang ada berupa persentase kecocokan yang akan berupa 100 % apabila ditemukan dengan algoritma KMP maupun BM, dan dibawahnya bila menggunakan *Levenshtein Distance*, serta waktu eksekusi.

3.2 Proses Pemetaan Masalah

Pada tugas besar ini, *pattern* merupakan 11 Karakter ASCII hasil konversi dari 88 pixel biner yang diambil dari citra masukkan pengguna

dan Teks yang ada merupakan representasi ASCII dari masing-masing citra yang ada di basis data.

Pada setiap pencocokan, akan digunakan algoritma yang dipilih pengguna untuk membandingkan 11 karakter ASCII masukan (*pattern*) dengan hasil konversi gambar penuh yang ada di basis data ke karakter ASCII (Teks). Dari situ, dapat ditentukan apakah ada/tidak *pattern* terkait di setiap gambar. Jika ada, maka besar kemungkinan gambar terkait merupakan gambar yang sama persis dengan gambar yang ada di basis data, sebab kemungkinan untuk 2 gambar memiliki *pattern* yang sama adalah $\frac{1}{2^{88}}$.

Regular Expression yang digunakan untuk tugas besar ini adalah sebagai berikut :

1. Untuk melakukan penanganan terhadap huruf vokal yang hilang, maka setiap huruf vokal di sebuah *string* dibuat opsional.
2. Untuk melakukan penanganan terhadap huruf besar dan kecil, maka seluruh huruf besar dibuat menjadi huruf kecil agar sekarang perbandingan setara
3. Untuk melakukan penanganan terhadap angka yang berubah menjadi huruf, digunakan format seperti berikut untuk setiap huruf yang mungkin berubah menjadi angka

3.3 Fitur Fungsional dan Arsitektur Aplikasi

3.3.1. Splash Screen



Gambar 3.3.1 Splash Screen

Aplikasi memiliki Splash Screen awal yang berisi cover dan juga judul aplikasi.

3.3.2. Main Screen



Gambar 3.3.2 Main Screen

Halaman utama memiliki sistem untuk penampilan informasi, serta tombol-tombol untuk mengoperasikan aplikasi.

3.3.3. Tombol



Gambar 3.3.3 Tombol-tombol

Terdapat 2 jenis tombol yang digunakan, yaitu tombol biasa dan juga tombol radio. Tombol biasa digunakan untuk menampilkan Main Screen, mengunggah citra dan juga untuk melakukan pencarian. Tombol radio digunakan untuk memilih algoritma yang akan digunakan

3.3.4. Penampilan Informasi

Gambar 3.3.4 Tampilan Informasi

Penampilan informasi digunakan sebagai, penampilan gambar hasil dan juga gambar masukan, waktu eksekusi, persentase kemiripan, dan juga biodata yang telah ditemukan

3.3.5. Musik

Pada saat aplikasi dijalankan, aplikasi akan memutar musik sebagai musik background.

3.3.6. Enkripsi Basis Data

```

MariaDB [tubesetina]> select * from biodata limit 3;

```

NIK	nama	agama	tempat_lahir	tanggal_lahir	jenis_kelamin	golongan_darah	ala
raan			status_pernikahan	pekerjaan			keuargaanega
wcncUwqBIMia+IsxjU7um==	k8zIGB2K9fy80fpaowX0n6qxk4QJtg2q69YxMIGo=	aaTvvt1lByThigaM1Ma7w==	TFuc8m6N8kt/GbPQp0Cqwe=	Laki-Laki	cINFONsRlqTzd1r2Fu/xrA==	TRI	
vJF2V+uufAMVVLBNPcVd6ojcegB0y0mrD6Z9IBqs=	hYhzkZRFU7ftvJ5M69hD/g==	iaaBhcnL7T624J9Hwsew==	g5ZM6/7HP4nb0g+VH11V6+U69gyJOK7AJxrisdYXL+e=			YuFwHadb/c	
VXxbgcT0ypA=							
L/1U8a0Y2bL886wH+XQO=	gxvneEp+5s8t95z16Gw==	Us/evCTUjPTereR2M17um==	X3aDdX09T8vEaQ5vM1Hq=	Laki-Laki	cINFONsRlqTzd1r2Fu/xrA==	qH1	
dEzzotFBLsYkIa5t0B7n2A6r7M1hB1SYOhp4Nies=	hYhzkZRFU7ftvJ5M69hD/g==	73/jQPhLN3VDehp4nhp+Xg==	71Jpht73REFhMcFT8GLINF81/UV5t8FsIitbTas=			YuFwHadb/c	
VXxbgcT0ypA=							
yf8d12zeWQ/f3N9j+qBe=	4Aq40EpZjcod8FGG0kFTA=	shw5ngJcV455Rg79I8e==	Uak3Lnfe7zLAzhXEY08g==	Laki-Laki	bz2M27qptXmyALPj4dIhRA==	P21	
PUNIG9QksbyScv811Vd6ojcegB0y0mrD6Z9IBqs=	hhNe3vgFahCY2vFL6z4zIQ==	iaaBhcnL7T624J9Hwsew==	cMdf2Q+V5P4eJ29s1CMyaVT+Ev1NR8V+H/dYy52Ftr/RbXWQcYvdwRH/p6Ta8ex			YuFwHadb/c	
VXxbgcT0ypA=							

Gambar 3.3.5 Basis Data Terenkripsi

Enkripsi akan dilakukan pada database, tujuannya adalah agar menjaga keamanan biodata. Setelah enkripsi dilakukan, maka biodata tidak dapat diakses dengan *query* biasa.

3.4 Contoh Ilustrasi Kasus

Angie merupakan seorang mahasiswa Institut Teknologi Bandung yang ingin mencari data seseorang dengan menggunakan foto sidik jari yang ia dapatkan. Ia mengetahui suatu perusahaan yang menyediakan kumpulan basis data sidik jari dan nama orang yang memilikinya. Angie kemudian memasukkan gambar tersebut ke aplikasi dan memilih algoritma yang ingin digunakannya yaitu antara KMP atau BM. Dari situ, aplikasi akan melakukan pencocokan dengan data-data yang ada di basis datanya. Terdapat 3 kemungkinan luaran:

1. Apabila ditemukan *exact match* dengan menggunakan algoritma BM maupun KMP, maka akan dikeluarkan persentase kecocokan 100 % dan data detail dari orang terkait akan ditampilkan di GUI
2. Jika tidak ditemukan *exact match* namun persentase kecocokan dengan menggunakan algoritma *Levenshtein Distance* masih lebih besar dari 60 %, maka data kecocokan dan
3. Jika sama sekali tidak ditemukan data terkait di basis data, maka akan ditampilkan pesan kesalahan bahwa data orang terkait tidak ada di basis data.

BAB IV

IMPLEMENTASI DAN PENGUJIAN


4.1 Spesifikasi Teknis Program

4.1.1. Struktur Data

1. Matriks

Matriks digunakan sebagai struktur data bagi beberapa algoritma seperti *Levenshtein Distance* yang menyimpan nilai terkecil untuk mengubah string ke string lainnya, sebagai *nature* dari program dinamis, serta sebagai pembantu pembuatan sbbox, enkripsi byte, invers sbbox, serta berbagai hal lainnya pada algoritma AES.

2. Image Record



```
1 public class ImageRecord
2 {
3     public required string Name { get; set; }
4     public required string BerkasCitra { get; set; }
5 }
6
```

Gambar x.

Image record merupakan struktur data yang digunakan untuk merepresentasikan data sidik jari yang didapatkan dari tabel `sidik_jari` pada basis data. Image Record digunakan untuk melakukan pencocokan antara *pattern* dan juga *full image* dari basis data.

3. Person Record



```

1 public class PersonRecord
2 {
3     public required string NIK { get; set; }
4     public required string Nama { get; set; }
5     public required string Tempat_lahir { get; set; }
6     public required string Tanggal_lahir { get; set; }
7     public required string Jenis_kelamin { get; set; }
8     public required string Golongan_darah { get; set; }
9     public required string Alamat { get; set; }
10    public required string Agama { get; set; }
11    public required string Status_perkawinan { get; set; }
12    public required string Pekerjaan { get; set; }
13    public required string Kewarganegaraan { get; set; }
14 }

```

Gambar 4.1.1 Struktur data Person Record

Person Record merupakan struktur data yang digunakan untuk merepresentasikan data biodata yang didapatkan dari basis data. Data - data ini akan ditampilkan kepada pengguna melalui GUI yang ada ketika nantinya

4.1.2. Fungsi dan Prosedur

1. BM

Nama Fungsi/Prosedur	Keterangan
public static int BmMatch(string text, string pattern)	Untuk melakukan pencarian menggunakan algoritma BM
public static int[] BuildLast(string pattern)	Untuk mencari last function algoritma BM

2. KMP

Nama Fungsi/Prosedur	Keterangan
public static int KmpMatch(string text, string pattern)	Untuk melakukan pencarian menggunakan algoritma KMP
public static int[] ComputeBorder(string pattern)	Untuk mencari <i>border function</i> KMP

3. Levenshtein

Nama Fungsi/Prosedur	Keterangan
public static int Compute(string s1, string s2)	Untuk melakukan komputasi dengan menggunakan Levenshtein Distance
public static double ComputeSimilarity(string s1, string s2)	Untuk mencari similaritas antar dua string
public static double ComputeMaxSimilarityParallel(List<Tuple<string, string>> pairs)	Untuk mencari similaritas antar dua string secara paralel

4. Converter

Nama Fungsi/Prosedur	Keterangan
public static bool[] ConvertFullImageToBinary(Bitmap bitmap)	Untuk mengubah suatu citra menjadi binary
public static (bool[] abc, int count) GetSelectedBinary(Bitmap bitmap, String type)	Untuk memiliki piksel yang akan digunakan pada pencocokan
public static string ConvertBinaryToAscii(bool[] binary)	Untuk mengubah semua binary menjadi suatu ASCII

5. Enkripsi AES

Nama Fungsi/Prosedur	Keterangan
public AES(byte[] key)	Konstruktor AES dan untuk melakukan key expansion untuk pertama kali
public static List<byte[]> SplitAndPadInput(string input)	Membagi sebuah <i>input</i> menjadi beberapa block 16 byte
public static byte[] PadStringToMultipleOf16Bytes(string input)	Melakukan padding pada block sehingga menjadi 16 byte

private void KeyExpansion(byte[] key)	Untuk melakukan tahapan Key Expansion pada AES
private byte[] SubWord(byte[] word)	
private byte[] RotWord(byte[] word)	
private byte[,] ParseBlock(byte[] block)	Untuk melakukan parsing terhadap block-block yang ada
private void AddRoundKey(byte[,] state, int round)	Untuk melakukan round key pada tiap ronde AES
private void SubBytes(byte[,] state)	Proses substitusi byte dengan SBOX
private void ShiftRows(byte[,] state)	Proses untuk mengganti baris
private byte GFMul(byte a, byte b)	Untuk melakukan Galois Field Multiplication
private void MixColumns(byte[,] state)	Melakukan tahap Mix Column pada Enkripsi AES
public byte[] Encrypt(string input)	Untuk melakukan enkripsi
private void InverseShiftRows(byte[,] state)	Untuk melakukan kebalikan dari Shift Rows pada Enkripsi
private void InverseSubBytes(byte[,] state)	Untuk melakukan kebalikan dari Substitusi Byte pada Enkripsi
private void InverseMixColumns(byte[,] state)	Untuk melakukan kebalikan dari Mix Column pada Enkripsi
public List<byte[]> SplitInputIntoBlocks(byte[] data)	Untuk membagi <i>input</i> menjadi block-block
public string Decrypt(string encryptedString)	Untuk melakukan Dekripsi

4.2 Tata Cara Penggunaan Program

4.2.1. Interface Program



Gambar 4.2.1.1 Splash Screen



Gambar 4.2.1.2 Main Screen



Gambar 4.2.1.3 Loading Screen



Gambar 4.2.1.4 Search Result

4.2.2. Tata Cara Penggunaan Program

Untuk menjalankan program ini, dapat dilakukan dengan melakukan *clone* repository dari repository github tugas besar ini : https://github.com/angiekierra/Tubes3_jajarija.

Sebelum program dijalankan, pastikan sudah terdapat Database yang aktif, berikut adalah tata cara untuk melakukan. Database akan di

enkripsi untuk sekuritas yang lebih baik, berikut adalah tata cara untuk melakukan enkripsi dan juga melakukan konektivitas kepada database:

1. Buat file .env sesuai dengan ketentuan yang ada di README
2. Masukkan dump dari sql ke basis data
3. Masuk ke direktori converter lalu jalankan **dotnet run**
4. File di basis data seharusnya sudah terenkripsi dengan baik


Untuk Menggunakan aplikasi, lakukan langkah-langkah berikut

1. Masuk ke direktori src
2. Jalankan **dotnet run**
3. Tekan tombol untuk memulai aplikasi
4. Unggah gambar yang ingin dicari
5. Pilih algoritma menggunakan BM atau KMP untuk pencarian
6. Tekan tombol ‘Search’ untuk mencari
7. Tunggu sampai hasil ditampilkan

4.3 Hasil Pengujian



4.3.1. Test Case 1 (Easy)

Input	
	
Algoritma	Hasil

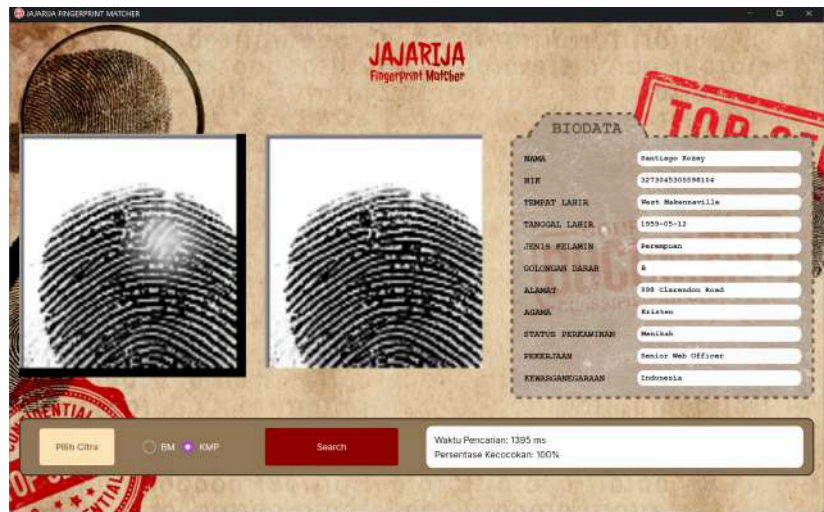
<p>BM</p>	 <p>The screenshot shows the JAJARIJA Fingerprint Matcher application. The 'BIODATA' form contains the following information:</p> <table border="1"> <thead> <tr> <th colspan="2">BIODATA</th> </tr> </thead> <tbody> <tr> <td>NAMA</td> <td>Kellie Smithan</td> </tr> <tr> <td>NIK</td> <td>3273041304051949</td> </tr> <tr> <td>TEMPAT LAHIR</td> <td>Malitanevisee</td> </tr> <tr> <td>TANGGAL LAHIR</td> <td>2000-04-12</td> </tr> <tr> <td>JENIS KELAKS</td> <td>Laki-Laki</td> </tr> <tr> <td>GOLONGAN DARAH</td> <td>O</td> </tr> <tr> <td>ALAMAT</td> <td>745 The Ridings</td> </tr> <tr> <td>AGAMA</td> <td>Kong Ho Cu</td> </tr> <tr> <td>STATUS PERKAWINAN</td> <td>Menikah</td> </tr> <tr> <td>PEKERJAAN</td> <td>Chief Identity Facilitator</td> </tr> <tr> <td>KEMANGKARAGANAAN</td> <td>Indonesia</td> </tr> </tbody> </table> <p>At the bottom, the search results show: Waktu Pencarian: 785 ms, Persentase Kecocokan: 100%.</p>	BIODATA		NAMA	Kellie Smithan	NIK	3273041304051949	TEMPAT LAHIR	Malitanevisee	TANGGAL LAHIR	2000-04-12	JENIS KELAKS	Laki-Laki	GOLONGAN DARAH	O	ALAMAT	745 The Ridings	AGAMA	Kong Ho Cu	STATUS PERKAWINAN	Menikah	PEKERJAAN	Chief Identity Facilitator	KEMANGKARAGANAAN	Indonesia
BIODATA																									
NAMA	Kellie Smithan																								
NIK	3273041304051949																								
TEMPAT LAHIR	Malitanevisee																								
TANGGAL LAHIR	2000-04-12																								
JENIS KELAKS	Laki-Laki																								
GOLONGAN DARAH	O																								
ALAMAT	745 The Ridings																								
AGAMA	Kong Ho Cu																								
STATUS PERKAWINAN	Menikah																								
PEKERJAAN	Chief Identity Facilitator																								
KEMANGKARAGANAAN	Indonesia																								
<p>KMP</p>	 <p>The screenshot shows the JAJARIJA Fingerprint Matcher application. The 'BIODATA' form contains the following information:</p> <table border="1"> <thead> <tr> <th colspan="2">BIODATA</th> </tr> </thead> <tbody> <tr> <td>NAMA</td> <td>Kellie Smithan</td> </tr> <tr> <td>NIK</td> <td>3273041304051949</td> </tr> <tr> <td>TEMPAT LAHIR</td> <td>Malitanevisee</td> </tr> <tr> <td>TANGGAL LAHIR</td> <td>2000-04-12</td> </tr> <tr> <td>JENIS KELAKS</td> <td>Laki-Laki</td> </tr> <tr> <td>GOLONGAN DARAH</td> <td>O</td> </tr> <tr> <td>ALAMAT</td> <td>745 The Ridings</td> </tr> <tr> <td>AGAMA</td> <td>Kong Ho Cu</td> </tr> <tr> <td>STATUS PERKAWINAN</td> <td>Menikah</td> </tr> <tr> <td>PEKERJAAN</td> <td>Chief Identity Facilitator</td> </tr> <tr> <td>KEMANGKARAGANAAN</td> <td>Indonesia</td> </tr> </tbody> </table> <p>At the bottom, the search results show: Waktu Pencarian: 812 ms, Persentase Kecocokan: 100%.</p>	BIODATA		NAMA	Kellie Smithan	NIK	3273041304051949	TEMPAT LAHIR	Malitanevisee	TANGGAL LAHIR	2000-04-12	JENIS KELAKS	Laki-Laki	GOLONGAN DARAH	O	ALAMAT	745 The Ridings	AGAMA	Kong Ho Cu	STATUS PERKAWINAN	Menikah	PEKERJAAN	Chief Identity Facilitator	KEMANGKARAGANAAN	Indonesia
BIODATA																									
NAMA	Kellie Smithan																								
NIK	3273041304051949																								
TEMPAT LAHIR	Malitanevisee																								
TANGGAL LAHIR	2000-04-12																								
JENIS KELAKS	Laki-Laki																								
GOLONGAN DARAH	O																								
ALAMAT	745 The Ridings																								
AGAMA	Kong Ho Cu																								
STATUS PERKAWINAN	Menikah																								
PEKERJAAN	Chief Identity Facilitator																								
KEMANGKARAGANAAN	Indonesia																								

4.3.2. Test Case 2 (Easy)

<p>Input</p>

	
Algoritma	Hasil
BM	

KMP





4.3.3. Test Case 3 (Easy)

Input




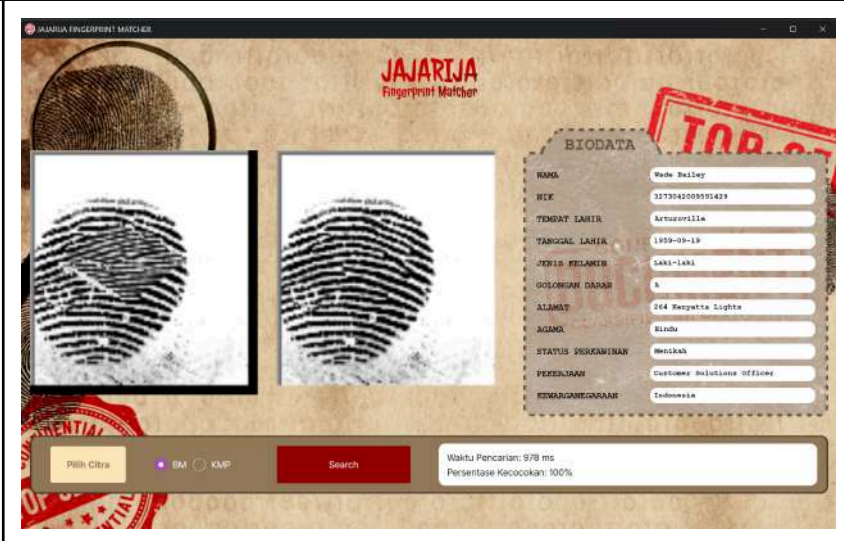
Algoritma

Hasil

BM	 <p>The screenshot shows the JAJARIJA Fingerprint Matcher application. The 'BM' (Biometric Matching) option is selected in the bottom navigation bar. The interface displays two fingerprint images, a biometric data form, and search results. The search results show a match with 100% similarity and a search time of 793 ms.</p>
KMP	 <p>The screenshot shows the JAJARIJA Fingerprint Matcher application. The 'KMP' (K-Nearest Neighbors) option is selected in the bottom navigation bar. The interface displays two fingerprint images, a biometric data form, and search results. The search results show a match with 100% similarity and a search time of 950 ms.</p>

4.3.4. Test Case 4 (Medium)

Input

	
Algoritma	Hasil
BM	

KMP



4.3.5. Test Case 5 (Medium)

Input




Algoritma

Hasil

<p>BM</p>	 <p>The screenshot shows the JAJARIJA Fingerprint Matcher application. The 'BM' (Biometric Matching) option is selected in the bottom navigation bar. The interface displays two fingerprint images, a biometric data form, and search results. The search results show a match with 100% similarity and a search time of 801 ms.</p>
<p>KMP</p>	 <p>The screenshot shows the JAJARIJA Fingerprint Matcher application. The 'KMP' (Key Point Matching) option is selected in the bottom navigation bar. The interface displays two fingerprint images, a biometric data form, and search results. The search results show a match with 100% similarity and a search time of 937 ms.</p>

4.3.6. Test Case 6 (Medium)

<p>Input</p>

	
Algoritma	Hasil
BM	

KMP



4.3.7. Test Case 7 (Hard)

Input





Algoritma

Hasil

<p>BM</p>	
<p>KMP</p>	

4.3.8. Test Case 8 (Hard)

<p>Input</p>

	
Algoritma	Hasil
BM	

KMP





4.3.9. Test Case 9 (Hard)

Input





Algoritma

Hasil

<p>BM</p>	 <p>The screenshot shows the JAJARIJA Fingerprint Matcher application. The 'BIODATA' section contains the following information:</p> <table border="1"> <thead> <tr> <th colspan="2">BIODATA</th> </tr> </thead> <tbody> <tr> <td>NAMA</td> <td>Ralph Hartmann</td> </tr> <tr> <td>NIK</td> <td>3273042510463271</td> </tr> <tr> <td>TEMPAT LAHIR</td> <td>Kamasefort</td> </tr> <tr> <td>TANGGAL LAHIR</td> <td>1946-10-25</td> </tr> <tr> <td>JENIS KELAMIN</td> <td>Laki-laki</td> </tr> <tr> <td>GOLONGAN DARAH</td> <td>A</td> </tr> <tr> <td>ALAMAT</td> <td>257 Rongel Ferry</td> </tr> <tr> <td>AGAMA</td> <td>Kristen</td> </tr> <tr> <td>STATUS PERAWINAN</td> <td>Belum Menikah</td> </tr> <tr> <td>PEKERJAAN</td> <td>Statistic Response Associate</td> </tr> <tr> <td>KEWARGANEGARAAN</td> <td>Indonesia</td> </tr> </tbody> </table> <p>At the bottom, the search results show: Waktu Pencarian: 926 ms, Persentase Keocokan: 100%.</p>	BIODATA		NAMA	Ralph Hartmann	NIK	3273042510463271	TEMPAT LAHIR	Kamasefort	TANGGAL LAHIR	1946-10-25	JENIS KELAMIN	Laki-laki	GOLONGAN DARAH	A	ALAMAT	257 Rongel Ferry	AGAMA	Kristen	STATUS PERAWINAN	Belum Menikah	PEKERJAAN	Statistic Response Associate	KEWARGANEGARAAN	Indonesia
BIODATA																									
NAMA	Ralph Hartmann																								
NIK	3273042510463271																								
TEMPAT LAHIR	Kamasefort																								
TANGGAL LAHIR	1946-10-25																								
JENIS KELAMIN	Laki-laki																								
GOLONGAN DARAH	A																								
ALAMAT	257 Rongel Ferry																								
AGAMA	Kristen																								
STATUS PERAWINAN	Belum Menikah																								
PEKERJAAN	Statistic Response Associate																								
KEWARGANEGARAAN	Indonesia																								
<p>KMP</p>	 <p>The screenshot shows the JAJARIJA Fingerprint Matcher application with the KMP algorithm selected. The 'BIODATA' section contains the same information as the previous screenshot.</p> <p>At the bottom, the search results show: Waktu Pencarian: 914 ms, Persentase Keocokan: 100%.</p>																								


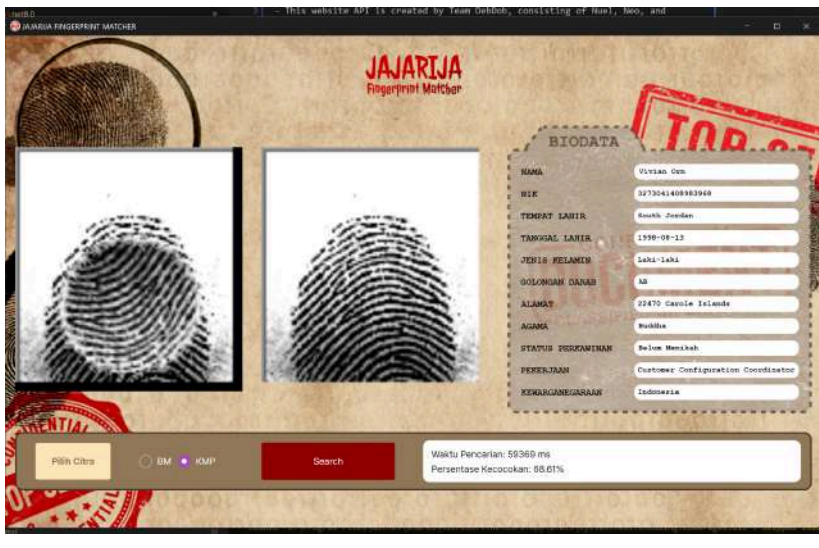
4.3.10. Test Case 10 (Levenstein)

<p>Input</p>

	
Algoritma	Hasil
Levensthein	


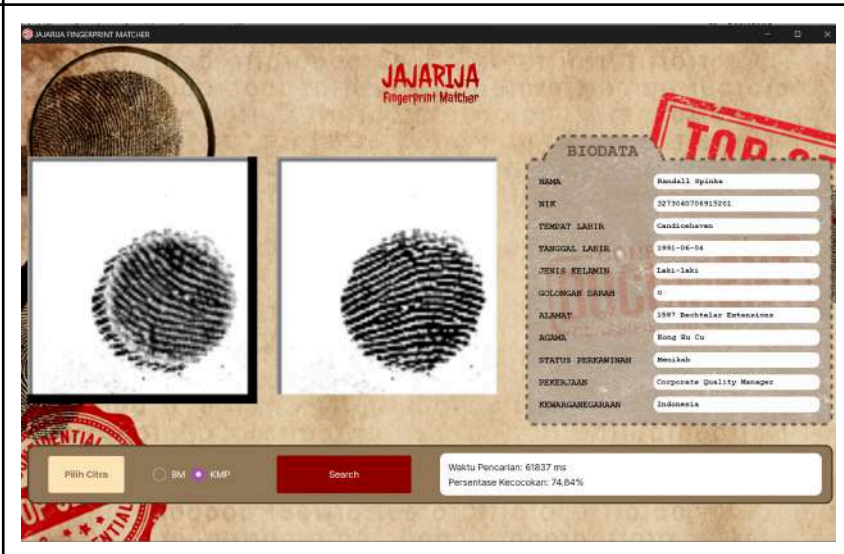
4.3.11. Test Case 11 (Levensthein)

Input

	
Algoritma	Hasil
Levenstein	

4.3.12. Test Case 12 (Levenstein)

Input

	
Algoritma	Hasil
Levensthein	

4.4 Analisis Hasil Pengujian

Berdasarkan hasil pengujian, algoritma Boyer-Moore (BM) dan Knuth-Morris-Pratt (KMP) tidak menunjukkan perbedaan waktu eksekusi yang signifikan. Rata-rata, kedua algoritma dapat dieksekusi dalam rentang waktu 700 - 1400 ms. Namun, secara umum, algoritma BM memiliki waktu eksekusi yang lebih cepat. Keunggulan utama dari algoritma

Boyer-Moore adalah efisiensinya dalam menangani ukuran alfabet yang besar. Karena kita membandingkan karakter ASCII yang terdiri dari 255 karakter, algoritma Boyer-Moore memiliki keunggulan di situ. Tidak hanya itu, Algoritma KMP juga kurang berguna di kebanyakan perbandingan karena kecil kemungkinan ada perulangan sufiks. Hal ini disebabkan konversi setiap 8 bit menjadi ASCII membuat kemungkinan bahwa setiap 8 bit terjadi perulangan yang sama persis menjadi sangat kecil terjadi.

Kompleksitas waktu algoritma BM dalam kasus terbaik adalah $O(N/M)$, di mana N adalah panjang teks dan M adalah panjang pola. Dalam kasus terburuk, kompleksitas waktu BM adalah $O(N * M)$. Di sisi lain, kompleksitas waktu KMP adalah $O(N + M)$ dalam semua kasus, menjadikannya lebih konsisten namun sedikit kurang efisien dibandingkan BM dalam banyak kasus praktis.

Namun, terdapat beberapa kasus di mana algoritma BM dan KMP tidak dapat menemukan kecocokan yang tepat untuk pola yang diberikan. Untuk situasi seperti itu, algoritma Levenshtein Distance digunakan untuk mengukur kesamaan pola. Algoritma ini memiliki kompleksitas waktu $O(M * N)$, di mana M dan N adalah panjang kedua string. Meskipun lebih pasti dalam menentukan gambar yang mirip, Levenshtein Distance jauh lebih lambat dibandingkan algoritma BM dan KMP. Berdasarkan hasil percobaan, waktu eksekusi dari algoritma Levenshtein Distance 70 kali lebih lama dibandingkan dengan algoritma BM dan KMP.

Oleh karena itu, pemilihan algoritma tergantung pada trade-off yang ada. Jika membutuhkan algoritma yang cepat namun mungkin kurang akurat, KMP atau BM dapat digunakan. Namun, jika memerlukan hasil yang selalu akurat meskipun lebih lambat, Levenshtein Distance adalah pilihan yang lebih baik.

BAB V

KESIMPULAN, SARAN, DAN REFLEKSI

5.1. Kesimpulan

Algoritma Boyer Moore dan juga Knuth–Morris–Pratt dapat digunakan sebagai sarana untuk menyelesaikan permasalahan deteksi individu berbasis biometrik melalui citra sidik jari. Walaupun tidak selalu menemukan *exact match*, namun kedua algoritma tersebut berhasil memangkas waktu komputasi yang jauh lebih lama dari pencocokan secara lengkap menggunakan algoritma *levensthein distance*. Kedua algoritma tersebut 60 kali lebih cepat dari *levensthein distance* dan menjadi algoritma yang cukup dapat diandalkan untuk melakukan identifikasi individu berdasarkan citra sidik jari, selama pemilihan pixel yang akan dipilih sebagai *pattern* tepat. Tidak hanya itu, *regular expressioin* juga menjadi alat yang sangat bisa digunakan untuk melakukan konversi dari data-data yang korup karena berubah dalam bahasa Alay menjadi data yang sesungguhnya.

5.2. Saran

Terdapat beberapa saran yang dapat dilakukan untuk pengembangan lebih lanjut aplikasi pencocokan sidik jari ini sehingga menjadi semakin efektif:

1. Menerapkan *preprocessing* pada gambar-gambar yang akan dibandingkan seperti *cropping* pada bagian yang tidak esensial
2. Menggunakan pemilihan pixel yang lebih heuristik untuk mendapatkan pixel-pixel terbaik dari setiap gambar
3. Melakukan enkripsi dengan algoritma enkripsi yang asimetris dan mengubah tabel basis data yang ada untuk meningkatkan keamanan.
4. Membuat tampilan GUI menjadi lebih responsif agar nyaman digunakan di berbagai ukuran perangkat

5.3. Tanggapan

Tugas besar ini sangat membantu memahami pemanfaatan *string matching* dalam kehidupan sehari-hari. Tidak hanya itu, Kami juga turut merasakan guna dari Regular Expression untuk bisa melakukan hal-hal berguna di kehidupan sehari-hari. Tidak lupa pula tugas besar ini membantu kami dalam memahami proses pembuatan GUI dalam bahasa C#.

5.4. Refleksi

Selama pengerjaan tugas besar ini kami sangat banyak belajar mengenai bahasa C#, *pattern matching*, dan juga *Regular Expression*. Hal yang paling menjadi tantangan ada pada bagaimana memahami algoritma-algoritma yang ada, serta bagaimana strategi untuk bisa memilih pixel yang paling bisa digunakan sebagai *pattern* untuk dibandingkan dengan gambar-gambar yang ada. Secara keseluruhan, pengerjaan tugas besar kali ini berjalan dengan maksimal dan sangat menyenangkan.

DAFTAR PUSTAKA

<https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2022-2023/String-Matching-dengan-Regex-2019.pdf> diakses pada 7 Juni 2024.

<https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Pencocokan-string-2021.pdf> diakses pada 7 Juni 2024

LAMPIRAN

Repository

Link Repository dari Tugas Besar 03 IF2211 Strategi Algoritma kelompok 31 “Jajarija” adalah sebagai berikut.

https://github.com/angiekierra/Tubes3_jajarija

Youtube

Link video Youtube dari Tugas Besar 03 IF2211 Strategi Algoritma kelompok 31 “Jajarija” adalah sebagai berikut.

<https://youtu.be/wFlsGxI79ic?si=ooemxct9ZjMl2w2t>