

## **LAPORAN TUGAS BESAR 3**

### **IF2211 STRATEGI ALGORITMA**

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



**Disusun oleh:**

**Kelompok - BismillahBersinar**

**Debrina Veisha Rashika W (13522025)**

**Melati Anggraini (13522035)**

**Shulha (13522087)**

**Miftahul Jannah (10023500)**

**SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA  
INSTITUT TEKNOLOGI BANDUNG**

**2024**

## DAFTAR ISI

<b>BAB I DESKRIPSI MASALAH</b>	<b>2</b>
<b>BAB II TEORI DASAR</b>	<b>5</b>
1. Algoritma Knuth-Morris-Pratt (KMP)	5
2. Algoritma Boyer Moore	6
3. Algoritma Similarity (LCS, Levenstein, Hamming Distance)	6
4. Algoritma Regular Expression (Regex) untuk Pattern Matching	7
5. Aplikasi Windows Forms Desktop App dengan C# dan .NET	9
<b>BAB III ANALISIS PEMECAHAN MASALAH</b>	<b>12</b>
1. Langkah-Langkah Pemecahan Masalah	12
2. Penyelesaian Masalah dengan Algoritma KMP dan BM	14
3. Fitur Fungsional dan Arsitektur Aplikasi Desktop	16
4. Contoh Ilustrasi Kasus	20
<b>BAB IV IMPLEMENTASI DAN PENGUJIAN</b>	<b>21</b>
1. Spesifikasi Teknis Program	21
2. Penjelasan Tata Cara Penggunaan Program	25
3. Hasil Pengujian	28
a. Hasil Uji 1	28
b. Hasil Uji 2	29
c. Hasil Uji 3	31
d. Hasil Uji 4	32
e. Hasil Uji 5	34
4. Analisis Hasil Pengujian	35
<b>BAB V PENUTUP</b>	<b>38</b>
1. Kesimpulan	38
2. Saran	38
3. Tanggapan dan Refleksi	38
<b>LAMPIRAN</b>	<b>40</b>
1. Tautan Repository Github	40
2. Tautan Video Bonus	40
<b>REFERENSI</b>	<b>41</b>

## BAB I

### DESKRIPSI MASALAH

Di era digital ini, keamanan data dan akses menjadi semakin penting. Perkembangan teknologi membuka peluang untuk berbagai metode identifikasi yang canggih dan praktis. Salah satunya adalah 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 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. Tugas ini 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.

Proses implementasi dilakukan dengan menggunakan algoritma Boyer-Moore dan Knuth-Morris-Pratt. 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. 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.

Jika tidak ada satupun sidik jari pada basis data yang *exact match* dengan sidik jari yang menjadi masukan melalui algoritma KMP ataupun BM, maka akan digunakan sidik jari paling mirip dengan kesamaan diatas nilai tertentu. Metode perhitungan tingkat kemiripan yang disarankan salah satu dari algoritma Hamming Distance, Levenshtein Distance, ataupun Longest Common Subsequence (LCS). Sebuah citra sidik jari akan dicocokkan dengan biodata seseorang. Biodata yang dicocokkan terdiri atas data-data yang terdapat pada KTP, antara lain: NIK, nama, tempat/tanggal lahir, jenis kelamin, golongan darah, alamat, agama, status perkawinan, pekerjaan, dan kewarganegaraan.

Seorang pribadi dapat memiliki lebih dari satu berkas citra sidik jari (relasi *one-to-many*). 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.

Jenis data korup adalah bahasa alay Indonesia, cara yang dapat digunakan untuk menangani ini adalah dengan menggunakan Regular Expression (Regex). Dengan regex, dapat dilakukan konversi pola karakter alay hingga dapat dikembalikan ke bentuk alfabetik yang bersesuaian. Setelah menggunakan Regex. untuk melakukan pattern matching antara nama yang bersesuaian dengan algoritma KMP dan BM dengan ketentuan yang sama seperti saat pencocokan sidik jari.

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. Komponen-komponen wajib aplikasi berikut:

1. Judul aplikasi
2. Tombol Insert citra sidik jari, beserta display citra sidik jari yang ingin dicari
3. Toggle Button untuk memilih algoritma yang ingin digunakan (KMP atau BM)
4. Tombol Search untuk memulai pencarian
5. Display sidik jari yang paling mirip dari basis data - Informasi mengenai waktu

eksekusi

6. Informasi mengenai tingkat kemiripan sidik jari dengan gambar yang ingin dicari, dalam persentase (%)
7. List biodata hasil pencarian dari basis data. Keluarkan semua nilai atribut dari individu yang dirasa paling mirip.

Berikut merupakan spesifikasi sistem yang akan digunakan:

1. Sistem dibangun dalam bahasa C# 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. Basis data yang digunakan 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
7. Program memiliki antarmuka yang user-friendly.

## BAB II

### TEORI DASAR

Pattern matching adalah proses untuk menemukan lokasi pertama dalam sebuah teks yang sesuai dengan pola tertentu. Dalam konteks ini, diberikan  $T$  atau teks (text), yaitu string panjang dengan panjang  $n$  karakter dan juga  $P$ : pola (pattern), yaitu string dengan panjang  $m$  karakter (asumsi  $m$  jauh lebih kecil dari  $n$ ) yang akan dicari dalam teks. Tujuan dari pattern matching adalah untuk menemukan (find atau locate) lokasi pertama dalam teks yang sesuai dengan pola yang diberikan.

#### 1. Algoritma Knuth-Morris-Pratt (KMP)

Algoritma Knuth-Morris-Pratt (KMP) mencari pola dalam teks dengan urutan dari kiri ke kanan, mirip dengan algoritma brute force. Namun, perbedaannya adalah algoritma KMP melakukan pergeseran pola dengan cara yang lebih cerdas dibandingkan algoritma *brute force*, sehingga menghindari banyak perbandingan yang tidak perlu.

Pendiri dari algoritma ini adalah Donald E. Knuth, seorang ilmuwan komputer dan Profesor Emeritus di Universitas Stanford. Knuth terkenal dengan karyanya yang monumental, "The Art of Computer Programming", dan dianggap sebagai bapak dari analisis algoritma. Karyanya sangat berpengaruh dalam pengembangan analisis rigor dari kompleksitas komputasional algoritma serta dalam penyusunan teknik-teknik matematis formal untuk algoritma.

Pada algoritma KMP, ketika terjadi ketidaksesuaian antara teks dan pola di posisi  $P$  pada indeks  $j$  (dengan  $T[i]$  tidak sama dengan  $P[j]$ ), algoritma ini akan menggeser pola dengan cara yang paling efisien untuk menghindari perbandingan yang sia-sia. Jawabannya adalah menggeser pola sejauh mungkin dengan memastikan pergeseran tersebut mencakup prefiks terbesar dari  $P[0..j-1]$  yang juga merupakan sufiks dari  $P[1..j-1]$ . Hal ini memungkinkan algoritma untuk menghindari perbandingan yang sudah pasti akan gagal, sehingga meningkatkan efisiensi pencarian pola dalam teks.

Kompleksitas waktu KMP dengan menghitung fungsi pinggiran :  $O(m)$  dan pencarian string :  $O(n)$ , sedangkan kompleksitas waktunya :  $O(m+n)$ . Keunggulan KMP adalah tidak perlu untuk mundur ke inputan. Sedangkan kekurangannya adalah KMP tidak berfungsi untuk ukuran alfabet meningkat.

## 2. Algoritma Boyer Moore

Algoritma Boyer Moore memiliki dua teknik yaitu *looking glass* dan teknik lompat karakter. Teknik looking glass dilakukan dengan menemukan  $P$  di  $T$  dengan bergerak mundur melalui  $P$  yang dimulai dari ujungnya. Sedangkan teknik lompat karakter dilakukan dengan apabila terjadi ketidaksesuaian pada  $T[i] == x$  dan karakter pada pola  $P[j]$  tidak sama dengan  $T[i]$ .

## 3. Algoritma Similarity (LCS, Levenstein, Hamming Distance)

### 3.1. Algoritma LCS

Longest Common Subsequence (LCS) adalah suatu metrik yang digunakan untuk menemukan panjang subsekuen terpanjang yang terdapat dalam dua atau lebih string. LCS dapat dihitung menggunakan algoritma dinamis yang memanfaatkan matriks dua dimensi. LCS adalah subsekuen terpanjang yang terdapat dalam dua atau lebih string. Algoritma LCS dapat dihitung menggunakan dua pendekatan: pendekatan rekursif dan pendekatan bottom-up (tabulation). Pendekatan rekursif LCS melibatkan generasi semua subsekuen dari dua string dan mencari subsekuen terpanjang yang terdapat dalam keduanya. Algoritma ini memiliki kompleksitas waktu  $O(2n)$ , di mana  $n$  adalah panjang string. Pendekatan bottom-up LCS melibatkan pembangunan matriks dua dimensi yang berisi panjang subsekuen terpanjang yang terdapat dalam string. Matriks ini dibangun secara bottom-up, dimulai dari matriks yang berisi nilai 0 dan 1, dan kemudian diisi dengan nilai yang sesuai berdasarkan karakter-karakter string. Dengan memoisasi, LCS memiliki kompleksitas waktu  $O(m*n)$  dimana  $m$  dan  $n$  adalah panjang string.

### 3.2. Algoritma Levenshtein

Algoritma ini dilakukan dengan menghitung jumlah operasi *string* paling sedikit yang diperlukan untuk mentransformasikan suatu *string* menjadi *string* yang lain. Algoritma ini menghitung jumlah minimum transformasi suatu *string* menjadi *string* lain

yang meliputi penggantian, penghapusan, dan penyisihan. Algoritma ini digunakan untuk mengoptimalkan pencarian tersebut karena sangat tidak efisien jika dilakukan pencarian setiap kombinasi operasi operasi string tersebut. Oleh karena itu, algoritma ini tergolong program dinamis dalam pencarian nilai minimal tersebut.

Jarak Levenshtein antara dua string  $a, b$  diberikan oleh  $\text{lev}_{a,b}(|a|, |b|)$  dimana

$$\text{lev}_{a,b}(i, j) = \begin{cases} \max(i, j) & \text{if } \min(i, j) = 0, \\ \min \begin{cases} \text{lev}_{a,b}(i - 1, j) + 1 \\ \text{lev}_{a,b}(i, j - 1) + 1 \\ \text{lev}_{a,b}(i - 1, j - 1) + 1_{(a_i \neq b_j)} \end{cases} & \text{otherwise.} \end{cases}$$

### 3.3. Algoritma Hamming Distance

Hamming distance adalah suatu kuantitas yang dimiliki saat membandingkan dua string dengan panjang yang sama. Kuantitas tersebut menyatakan berapa simbol yang berbeda pada kedua string.

## 4. Algoritma Regular Expression (Regex) untuk Pattern Matching

Regular expression (regex) adalah notasi standar yang mendeskripsikan suatu pola (*pattern*) berupa urutan karakter atau string. Regex digunakan untuk pencocokan string (*string matching*) dengan efisien. Regex telah menjadi standar yang tersebar di semua tools dan bahasa pemrograman sehingga penting untuk dipelajari. Komponen-komponen dari regex adalah sebagai berikut:

1. Konstruksi regex umum untuk membentuk karakter kelas.

Construct	Deskripsi
[abc]	a, b, atau c (simple class)
[^abc]	Semua karakter <b>selain</b> a,b,c (negasi)
[a-zA-Z]	a sampai z atau A sampai Z, inclusive (range)
[a-d[m-p]]	a sampai d atau m sampai p (gabungan)
[a-z&&[def]]	d, e atau f (irisan)
[a-z&&[^bc]]	a sampai z, kecuali b dan c (subtraksi)
[a-z&&[^m-p]]	a sampai z, dan bukan m sampai p (subtraksi)

2. Predefined class untuk membuat regex lebih mudah dibaca dan mengurangi kesalahan.

Construct	Deskripsi
.	Semua karakter
\d	Digit [0-9]
\D	Non digit [^0-9] (hati-hati dengan huruf besar)
\s	Whitespace character [ \t\n\x0B\f\r]
\S	Non whitespace character [^s]
\w	Word character [a-zA-Z_0-9]
\W	Non word character [^w]

3. Quantifier untuk mendefinisikan jumlah pengulangan pola dan perannya sangat penting.

Construct	Arti
X?	X muncul <b>satu</b> atau tidak sama sekali
X*	X muncul <b>nol</b> atau banyak
X+	X muncul <b>satu</b> atau banyak
x{ <i>n</i> }	X muncul tepat <i>n</i> kali
x{ <i>n</i> ,}	X muncul setidaknya <i>n</i> kali
x{ <i>n,m</i> }	X muncul antara <i>n</i> sampai <i>m</i> kali

4. Boundary matchers digunakan untuk mencari pola yang muncul di posisi tertentu, misalnya di awal atau di akhir.

Construct	Deskripsi
^	Awal baris
\$	Akhir baris
\b	Batas kata
\B	Batas bukan kata
\G	Akhir match sebelumnya
\z	Akhir dari input tapi untuk final terminator jika ada
\Z	Akhir dari input

## 5. Aplikasi Windows Forms Desktop App dengan C# dan .NET

C# adalah bahasa pemrograman berorientasi objek yang modern, inovatif, dan bersifat sumber terbuka, yang dikembangkan oleh Microsoft sebagai bagian dari inisiatif .NET. C# terus mendapatkan popularitas dan saat ini menjadi salah satu dari lima bahasa pemrograman teratas di GitHub. C# dirancang untuk memudahkan pengembangan aplikasi yang efisien dan aman, menjadikannya pilihan ideal untuk pengembangan aplikasi desktop, web, mobile, dan game. Bahasa ini mendukung pengembangan multi-platform, yang memungkinkan aplikasi yang dibuat dengan C# untuk berjalan di sistem operasi Windows, macOS, dan Linux. C# juga digunakan secara luas dalam

pengembangan aplikasi server-side dan aplikasi mobile berkat kerangka kerja .NET yang kuat.

.NET adalah platform pengembangan yang sumber terbuka dan lintas platform, yang menawarkan lingkungan yang kaya untuk pembuatan berbagai jenis aplikasi, mulai dari situs web, aplikasi mobile, aplikasi desktop, hingga permainan dan aplikasi untuk Internet of Things (IoT). Dengan .NET, pengembang dapat memilih dari berbagai bahasa pemrograman, editor, dan pustaka untuk menciptakan solusi yang inovatif dan efisien. Platform ini meliputi rangkaian alat, pustaka, dan dukungan bahasa yang luas, yang memungkinkan pengembangan perangkat lunak yang modern, dapat diskalakan, dan berperforma tinggi. Sebagai tambahan, .NET didukung oleh komunitas pengembang yang aktif dan berdedikasi, yang terus mengembangkan dan memperbarui platform untuk memenuhi tuntutan teknologi terkini.

Windows Forms (WinForms) adalah kerangka kerja untuk pengembangan aplikasi desktop pada platform Microsoft .NET. Kerangka kerja ini memungkinkan pengembang untuk menciptakan aplikasi yang kaya akan fitur dengan antarmuka pengguna grafis (GUI) yang menarik dan intuitif. WinForms menyediakan kumpulan kontrol dan komponen visual yang luas, yang dapat ditarik dan dilepas ke dalam desainer visual, memudahkan proses pengembangan UI. Pengembang dapat dengan mudah menyesuaikan komponen ini atau memperluas fungsionalitasnya melalui kode untuk menciptakan aplikasi yang responsif dan mudah digunakan. WinForms sangat cocok untuk aplikasi yang memerlukan interaksi intensif dengan pengguna, pengolahan data lokal, dan integrasi dengan layanan Windows.

Kerangka kerja yang sering digunakan lainnya adalah Windows Presentation Foundation (WPF). Dibanding WPF, keunggulan utama WinForms adalah pendekatan desain yang lebih sederhana dan *direct*, yang membuatnya ideal untuk proyek-proyek dengan kurva pembelajaran yang lebih pendek dan untuk pengembang yang memprioritaskan waktu pengembangan yang cepat dan kurang kompleksitas. Pada WinForms, *debugging* dan pemeliharaan aplikasi lebih mudah dilakukan karena strukturnya yang lebih mudah dipahami. Karena itu, WinForms sering dipilih untuk

pengembangan aplikasi yang tidak memerlukan antarmuka pengguna yang sangat kompleks, di mana efisiensi pengembangan lebih diutamakan.

## BAB III

### ANALISIS PEMECAHAN MASALAH

#### 1. Langkah-Langkah Pemecahan Masalah

Secara umum, berikut langkah penyelesaian masalah yang dilakukan dalam melakukan pencarian pola citra sidik jari yang sesuai:

1. Program melakukan enkripsi pada basis data yang dimiliki untuk menjaga keamanan dari basis data agar tidak diakses oleh pihak asing. Setelah, enkripsi program akan menyimpan data hasil dekripsi pada sebuah dictionary yang berisi biodata tiap orang.
2. Program menerima input gambar sidik jari dari pengguna, gambar yang diunggah dipastikan memiliki ekstensi jpg, jpeg, png, atau BMP. Setelah pengguna mengunggah gambar, gambar akan disimpan untuk nantinya diolah dalam melakukan pencarian sidik jari yang sesuai.
3. Setelah mengunggah gambar, pengguna memilih algoritma yang diinginkan dalam melakukan proses pencarian yakni algoritma Knuth-Morris-Pratt (KMP) atau Boyer Moore (BM). Jika pengguna tidak memilih algoritma apapun, maka secara *default* program akan melakukan pencarian menggunakan algoritma KMP.
4. Setelah pengguna menekan tombol *search* proses pencarian dimulai. Input gambar citra sidik jari akan diproses untuk diubah menjadi pola dalam bentuk ASCII. Proses perubahan gambar menjadi ASCII, dilakukan dengan *crop* gambar terlebih dahulu agar pola pencarian hanya berfokus pada citra sidik jari saja dengan menghapus latar belakang pada gambar. Setelah itu, dilakukan pencarian intensitas gelap terang pada setiap pixelnya yang selanjutnya akan diubah dalam representasi biner 8 bit. Setelah mendapat nilai biner untuk setiap pixel, biner 8 bit akan diubah menjadi karakter ASCII yang digunakan dalam proses *pattern matching*.
5. Untuk input gambar dari pengguna akan diambil 30 pixel awal dan akhir pada sidik jari. Hal ini disebabkan untuk menghindari kerusakan pola sidik jari yang

ada pada bagian awal atau akhir. Karena bisa saja yang gambar terpotong di bagian atas dan terdeteksi pola di bagian bawah ikut berbeda, padahal sebenarnya pola dibagian bawah sama dengan yang terdapat di basis data. Oleh karena itu, dengan menggunakan pola sidik jari pada awal dan akhir diharapkan bisa menemukan pola sidik jari yang sesuai dengan lebih akurat.

6. Setelah mendapat pola sidik jari dari pengguna, pola ini akan dicocokkan dengan gambar sidik jari yang ada di basis data. Gambar yang ada di basis data juga dilakukan perubahan menjadi pola ASCII terlebih dahulu dengan proses yang sama dengan yang terdapat pada langkah ke-3. Selanjutnya, akan dilakukan proses pencarian pola gambar masukan dengan gambar yang ada di basis data berdasarkan algoritma yang dipilih oleh pengguna.
7. Jika ditemukan pola pada salah satu gambar yang ada di basis data, maka program akan *me-return* lokasi gambar citra sidik jari yang ada pada basis data beserta tingkat kemiripan antara dua gambar tersebut. Selanjutnya, lokasi gambar yang ditemukan akan dicari pemiliknya melalui basis data yang dimiliki.
8. Jika tidak ditemukan pola yang sesuai akan dilakukan proses pencarian kemiripan melalui algoritma Hamming distance. Algoritma ini dilakukan dengan mencari kuantitas berapa simbol yang berbeda pada kedua string. Algoritma ini dipilih karena sederhana dan dapat menghasilkan hasil yang cukup akurat dalam mencari kemiripan dari dua pola. Selanjutnya, jika ditemukan gambar yang memiliki kemiripan di atas 50% maka lokasi gambar yang ditemukan akan dicari pemiliknya melalui basis data yang dimiliki. Batas ini, dipilih karena dalam situasi di mana kecepatan dan kenyamanan menjadi faktor utama, seperti pada ponsel atau aplikasi pengenalan sidik jari yang digunakan untuk mengakses perangkat atau aplikasi sehari-hari yang dimana mungkin saja sidik jari yang diberikan tidak penuh secara 100%, batas 50% mungkin diterima karena memberikan keseimbangan antara keamanan yang cukup dan kinerja yang baik. Kesalahan identifikasi kemungkinan akan lebih rendah, tetapi kemungkinan ada beberapa *false acceptances*.
9. Setelah ditemukan nama pemilik dari citra sidik jari tersebut. Nama yang ada pada basis data harus diubah dulu dari nama alay ke nama normal dengan

menggunakan regex. Selanjutnya akan dilakukan iterasi untuk setiap nama yang ada di basis data dan dicari nama yang ada pada basis data yang memiliki kemiripan paling tinggi dengan nama yang ada pada tabel sidik jari menggunakan algoritma Levenshtein Distance.

10. Setelah mendapat nama asli, akan dilakukan pencarian biodata lengkap berdasarkan nama yang dimiliki pada tabel biodata yang ada pada dictionary (tabel biodata yang telah didekripsi). Biodata akan ditampilkan pada layar beserta waktu kecepatan pencarian dan tingkat kemiripan sidik jari input dengan yang ada pada basis data.

## 2. Penyelesaian Masalah dengan Algoritma KMP dan BM

### 2.1. Penyelesaian Masalah dengan Algoritma KMP

Proses pencarian menggunakan algoritma KMP dalam melakukan perbandingan pola pada sidik jari sebagai berikut:

1. Pertama dilakukan adalah membuat tabel *border function* untuk pola yang akan dicari. Pembuatan tabel ini terdapat pada *function KMPPreprocess* yang bekerja dengan mencari nilai *pattern* pada sufiks yang sesuai dengan prefiks dari pattern tersebut.
2. Setelah memiliki tabel fungsi pinggiran, sebuah text yang merupakan pola sidik jari yang ada pada gambar asli dengan urutan dari kiri ke kanan akan dicocokkan setiap karakter satu per satu. Secara matematis, dalam pencariannya, jika ditemukan sebuah ketidakcocokan antara teks dan pattern P pada  $P[j]$  ( $T[i] \neq P[j]$ ), kita menggeser pattern-nya sejauh panjang pattern dikurangi panjang prefix terbesar dari  $P[0..j-1]$  yang merupakan suffix dari  $P[1..j-1]$ .
3. Jika proses pencarian berhasil mencari kemiripan hingga akhir karakter dari *pattern* yang dicari maka fungsi KMP akan mengembalikan nilai true, sebaliknya jika hingga akhir text tidak ditemukan pola yang sesuai akan dikembalikan nilai false.

4. Jika *false*, maka proses pencarian akan dilakukan dengan mencari kemiripan menggunakan algoritma Hamming Distance. Sidik jari dengan jarak Hamming terkecil dipilih sebagai yang paling mirip.
5. Program akan melakukan iterasi melalui setiap sidik jari dalam basis data. Jika sidik jari cocok dengan pola input menggunakan KMP, maka sidik jari tersebut dipilih langsung sebagai yang paling mirip. Jika tidak ada sidik jari yang cocok menggunakan KMP, dilakukan perhitungan jarak Hamming antara sidik jari input dan sidik jari dalam basis data.

Algoritma ini memiliki kompleksitas waktu untuk menghitung *border function*, yaitu  $O(m)$  dan pencarian string, yaitu  $O(n)$ . Dengan demikian, kompleksitas waktu untuk algoritma KMP adalah  $O(m + n)$ .

## 2.2. Penyelesaian Masalah dengan Algoritma BM

Proses pencarian menggunakan algoritma BM dalam melakukan perbandingan pola pada sidik jari sebagai berikut:

1. Pertama dilakukan adalah membuat tabel *last occurrence* untuk pola yang akan dicari. Pembuatan tabel ini terdapat pada *function BMPPreprocess* yang bekerja dengan menyimpan posisi terakhir kemunculan setiap karakter dalam pola.
2. Setelah memiliki tabel *last occurrence*, sebuah text yang merupakan pola sidik jari yang ada akan dicocokkan dengan Teknik looking-glass yang dimana pattern P dicari dalam teks T dengan bergerak mundur. Selanjutnya proses pencarian kemiripan dilakukan dengan teknik character-jump dengan tiga kemungkinan yang terjadi saat ada ketidakcocokan pada  $T[i] == x$ , sebagai berikut:
  - 1) Ketika pattern P mengandung x di suatu tempat, geser P ke kanan untuk menyejajarkan kemunculan terakhir x dalam P dengan  $T[i]$
  - 2) Ketika pattern P mengandung x di suatu tempat dan pergeseran ke kanan ke kemunculan terakhir tidak mungkin, geser P sejauh 1 karakter ke  $T[i+1]$ .
  - 3) Jika keadaannya selain kedua kemungkinan sebelumnya, geser pattern P untuk menyejajarkan  $P[0]$  dengan  $T[i+1]$ .

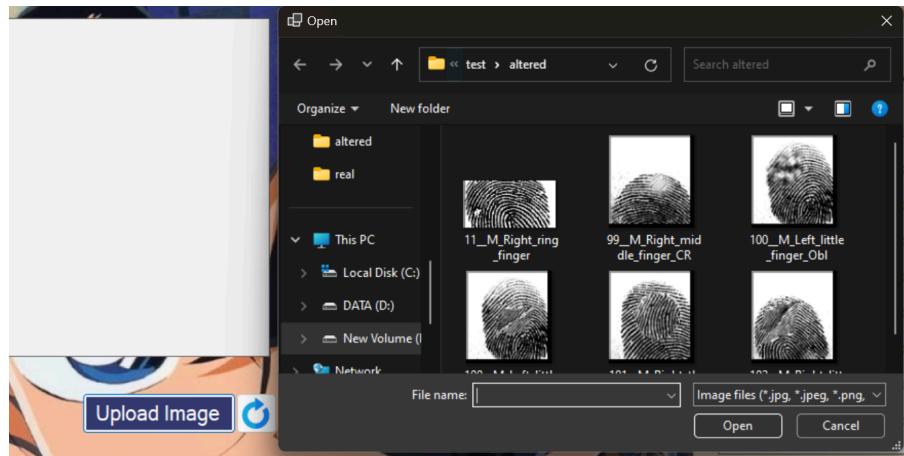
3. Jika proses pencarian berhasil mencari kemiripan hingga karakter pertama dari *pattern* yang dicari maka fungsi BM akan mengembalikan nilai true, sebaliknya jika hingga akhir text tidak ditemukan pola yang sesuai akan dikembalikan nilai false.
4. Jika *false*, maka proses pencarian akan dilakukan dengan mencari kemiripan menggunakan algoritma Hamming. Sidik jari dengan jarak Hamming terkecil dipilih sebagai yang paling mirip.
5. Program akan melakukan iterasi melalui setiap sidik jari dalam basis data. Jika sidik jari cocok dengan pola input menggunakan BM, maka sidik jari tersebut dipilih langsung sebagai yang paling mirip. Jika tidak ada sidik jari yang cocok menggunakan BM, dilakukan perhitungan jarak Hamming antara sidik jari input dan sidik jari dalam basis data.

Algoritma ini memiliki kompleksitas waktu untuk mencari *last occurrence*, yaitu  $O(m)$  dan pencarian string, yaitu  $O(n)$ . Dengan demikian, kompleksitas waktu untuk algoritma BM adalah  $O(m + n)$ .

### 3. Fitur Fungsional dan Arsitektur Aplikasi Desktop

Aplikasi desktop ini dikembangkan sepenuhnya menggunakan C# untuk antarmuka pengguna (dengan WinForms) dan backend. WinForms adalah framework pengembangan aplikasi desktop yang disediakan oleh platform Microsoft .NET, sementara C# adalah bahasa pemrograman yang digunakan untuk mengembangkan kedua bagian dari aplikasi.

#### 3.1. Unggah Gambar Sidik Jari



Gambar 3.1.1 Tampilan Button Upload Image dan File Dialog

Fitur upload gambar sidik jari memungkinkan pengguna untuk mengupload sidik jari yang akan dicari pemiliknya. Pengguna terlebih dahulu menekan button Upload Image, kemudian akan terbuka window sistem file untuk memilih file mana yang akan diupload. Pengguna dapat mengunggah gambar sidik jari dalam format .jpg, .jpeg, .png, atau .BMP. Kemudian sistem akan menyimpan gambar yang diupload di folder assets dan menampilkannya di antarmuka pengguna.

### 3.2. Reset Gambar



Gambar 3.2.1 Tampilan Button Refresh Image

Fitur reset pengguna memungkinkan pengguna untuk menghapus gambar yang telah diupload dan mereset antarmuka untuk mengunggah gambar yang baru. Sistem mengatur ulang dan mengosongkan tampilan di antarmuka.

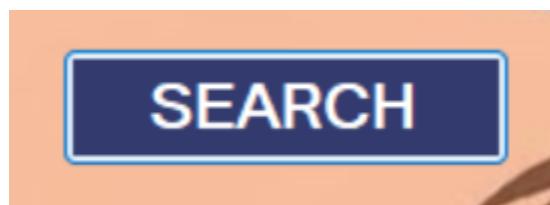
### 3.3. Memilih Algoritma Pencocokan



Gambar 3.3.1 Tampilan Dropdown untuk Memilih Algoritma Pencocokan

Pengguna dapat memilih algoritma pencocokan yaitu KMP atau BM dari dropdown yang disediakan. Algoritma yang dipilih pengguna digunakan dalam proses pencarian sidik jari.

### 3.4. Pencarian Sidik Jari



Gambar 3.4.1 Tampilan Button Search

Setelah berhasil mengupload gambar, pengguna lalu menekan tombol search untuk memulai pencarian sidik jari untuk menemukan sidik jari yang paling mirip yang ada di database. Sistem kemudian mengkonversi gambar kedalam representasi ASCII terlebih dahulu lalu melakukan pencarian kecocokan dengan algoritma yang telah dipilih.



Gambar 3.4.2 Tampilan Animasi Ketika Pencarian Dilakukan

Saat proses pencarian, program akan memunculkan animasi *loading searching* seperti pada Gambar 3.4.2 diatas.

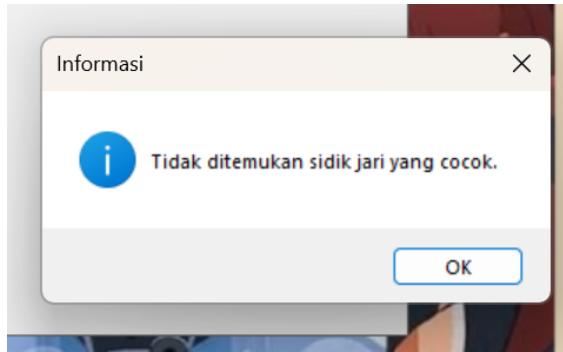
### 3.5 Tampilan Hasil Pencarian



Gambar 3.1.1 Tampilan List Biodata dan Waktu Pencarian

Setelah melakukan pencarian, antarmuka akan menampilkan biodata yang berisi NIK, Nama, Tempat Lahir, Jenis Kelamin, Golongan Darah, Alamat, Agama, Status Perkawinan, Pekerjaan dan Kewarganegaraan. Selain itu juga akan ditampilkan yang dibutuhkan untuk pencarian dalam milliseconds(ms). Informasi biodata ini diambil dari database stimaalay.db.

### 3.6 Popup Ketika Pencarian Tidak Ditemukan



Gambar 3.1.1 Tampilan Popup Tidak Ada Sidik Jari yang Cocok

Fitur popup ini akan muncul ketika pencarian tidak ditemukan, artinya persentase kecocokannya atau nilai similaritynya kurang dari 50%.

## 4. Contoh Ilustrasi Kasus

Berikut merupakan contoh kasus dari penggunaan Aplikasi FingerPrintDetector.

1. Pada saat pertama kali aplikasi ditampilkan, pengguna dapat mengupload gambar melalui *button upload Image*. Pengguna hanya bisa mengunggah gambar bertipe jpg, jpeg, png, atau BMP.
2. Jika ingin mengganti atau menghapus gambar yang diunggah, pengguna bisa menekan tombol *upload Image* kembali atau tombol *refresh* untuk menghapus gambar.
3. Kemudian pengguna memilih algoritma pencarian yaitu KMP atau BM. Jika pengguna tidak memilih algoritma maka secara default akan dilakukan pencarian dengan algoritma BMP.
4. Setelah upload image dan memilih algoritma pencarian, pengguna dapat langsung menekan *button Search* untuk memulai pencarian.
5. Hasil pencarian akan ditampilkan di antarmuka pengguna yang berisi detail biodata dan waktu pencarian.
6. Jika tidak ada sidik jari yang sesuai maka program akan menampilkan pop up yang memberi tahu pengguna bahwa “Tidak ditemukan sidik jari yang cocok”.

## BAB IV

### IMPLEMENTASI DAN PENGUJIAN

#### 1. Spesifikasi Teknis Program

##### 1.1. Struktur Data

###### 1. List

Struktur data list digunakan untuk menyimpan seluruh *image path* yang ada di database. Hal ini bertujuan untuk melakukan iterasi pada setiap gambar sidik jari yang dimiliki dalam proses pencocokan pola. Struktur data list juga digunakan untuk menyimpan seluruh nama pada biodata yang digunakan untuk melakukan iterasi pada setiap nama setelah dilakukan proses regex pada nama alay. Selain itu, list juga digunakan untuk menyimpan *border function* pada proses pencarian dengan algoritma KMP.

###### 2. Tuple

Struktur data tuple digunakan sebagai *return* objek pada proses pencarian dengan algoritma BM dan KMP. Dengan menggunakan tuple, bisa dilakukan return lebih dari satu objek sekaligus. Tuple yang dikembalikan akan berisi sumber lokasi gambar sidik jari yang sesuai beserta ukuran similaritynya.

###### 3. Dictionary

Struktur data *dictionary* digunakan dalam membuat tabel last occurrence pada algoritma BM. Struktur data ini digunakan untuk mempercepat mendapat nilai indeks terakhir karakter. Karakter akan berperan sebagai kunci dan lokasi indeks terakhir akan berperan sebagai *value* dari karakter tersebut.

###### 4. Struct

Salah satu struct yang digunakan adalah struktur data Biodata. Biodata digunakan untuk menyimpan detail identitas dari pemilik sidik jari. Struktur ini menyimpan NIK, Nama, Tempat Lahir, Tanggal Lahir, Jenis Kelamin, Golongan Darah, Alamat, Agama, Status Perkawinan, Pekerjaan dan Kewarganegaraan.

## 1.2. Fungsi Dan Prosedur

Pada folder src/FingerPrintDetector terdapat beberapa *class* yang digunakan dalam membangun program. Sebagai berikut:

### 1. BM

No.	Fungsi / Prosedur	Keterangan
1.	BmPrepocess (string pattern)	Fungsi untuk membuat tabel last occurrence pada pattern.
2.	BmSearch (string text, string pattern)	Fungsi utama untuk melakukan pencocokan string dengan algoritma BM.
3.	FindMostSimilarFingerprint (string inputFingerprint, List<string> database)	Fungsi untuk melakukan fingerprint paling mirip dengan melakukan iterasi pencarian pada tiap gambar di database.

### 2. KMP

No.	Fungsi / Prosedur	Keterangan
1.	KmpPrepocess (string pattern)	Fungsi untuk membuat tabel border function pada pattern.
2.	KmpSearch (string text, string pattern)	Fungsi utama untuk melakukan pencocokan string dengan algoritma KMP.
3.	FindMostSimilarFingerprint (string inputFingerprint, List<string> database)	Fungsi untuk melakukan fingerprint paling mirip dengan melakukan iterasi pencarian pada tiap gambar di database.

### 3. ImageManager

No.	Fungsi / Prosedur	Keterangan
1.	ConvertImageToAscii8Bit (string imagePath, int pattern)	Fungsi utama untuk mengubah gambar menjadi pattern yang terdiri dari karakter ASCII 8 bit.
2.	ImagetoAscii(string imagePath, int pattern)	Fungsi untuk mengubah gambar menjadi pattern.
3.	GetAllImagePaths (string directoryPath)	Fungsi untuk mendapat semua <i>path</i> gambar yang ada di direktori.

#### 4. Database

No.	Fungsi / Prosedur	Keterangan
1.	GetPersonNameByImagePath (string imagePath)	Fungsi untuk mendapat nama pemilik sidik jari dari tabel basis data berdasarkan <i>path</i> .
2.	GetBiodataByName (string name)	Fungsi untuk mendapat biodata penuh pemilik sidik jari dari tabel biodata data berdasarkan nama.

#### 5. RegexHelper

No.	Fungsi / Prosedur	Keterangan
1.	ConvertAlayToNormal(string alayText)	Fungsi utama untuk mengubah nama alay menjadi nama sebenarnya menggunakan regular expression.
2.	IsMatch(string input, string pattern)	Fungsi untuk memeriksa apakah nama input dan <i>pattern</i> sudah sesuai.

3.	LevenshteinDistance(string s1, string s2)	Fungsi untuk mendapat nilai Levenshtein distance yang digunakan dalam mencari nama yang mirip dengan yang ada di basis data.
4.	FindClosestMatch(string input, List<string> candidates)	Fungsi untuk mencari nama pada biodata yang yang paling mendekati dengan nama input.
5.	GetAllNamesFromBiodata()	Fungsi untuk mendapat semua nama yang ada pada tabel biodata di basis data.
6.	GetAlayName(string sidikJariName)	Fungsi untuk membantu mendapat nama alay dari basis data.

## 6. Similarity

No.	Fungsi / Prosedur	Keterangan
1.	CalculateHammingDistance (string text1, string text2)	Fungsi untuk nilai kemiripan antara dua text menggunakan algoritma Hamming.
2.	CalculateLevenshteinDistance (string text1, string text2)	Fungsi untuk nilai kemiripan antara dua text menggunakan algoritma Levenshtein.

## 7. MainForm

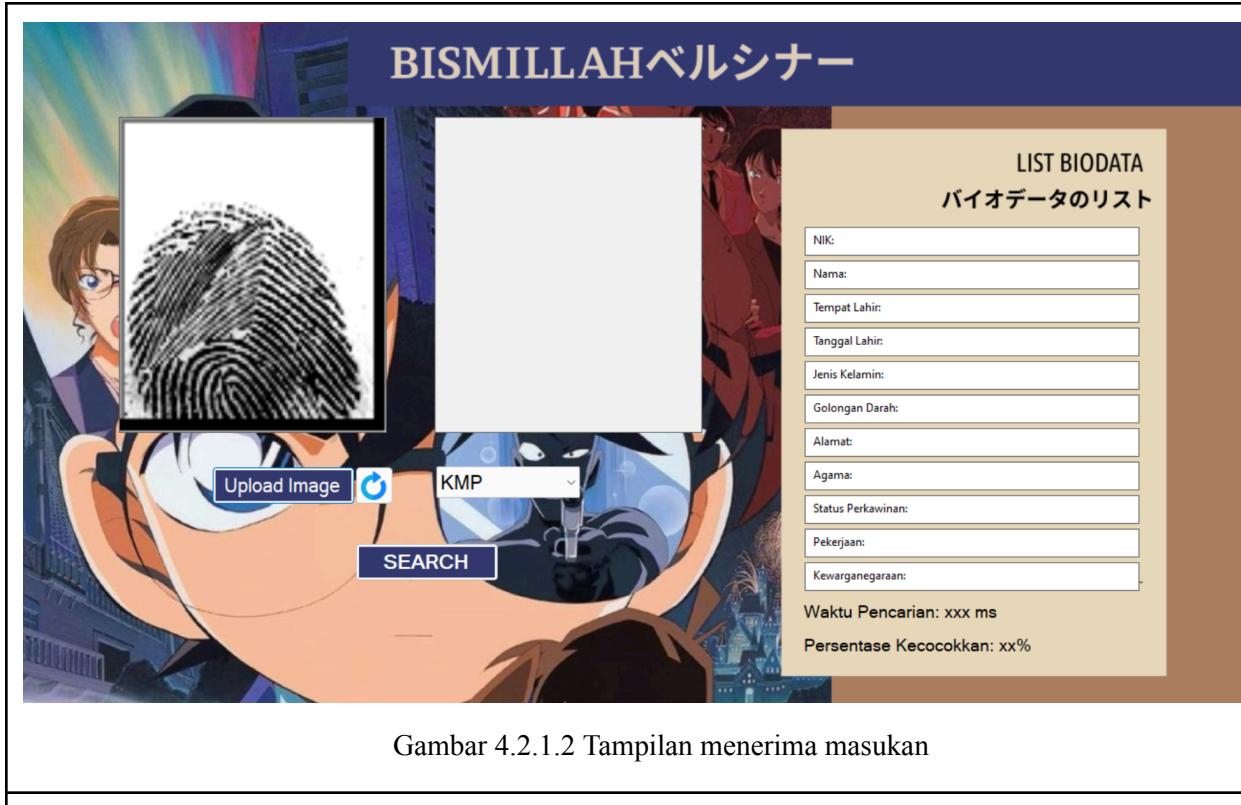
No.	Fungsi / Prosedur	Keterangan
1.	UploadImageButton_Click(object sender, EventArgs e)	Event handler untuk mengunggah gambar sidik jari.

2.	RefreshButton_Click(object sender, EventArgs e)	Event handler untuk mereset gambar yang sebelumnya telah diunggah.
3.	SearchButton_Click(object sender, EventArgs e)	Event handler untuk memulai pencarian gambar sidik jari.

## 2. Penjelasan Tata Cara Penggunaan Program

### 2.1. Interface Program

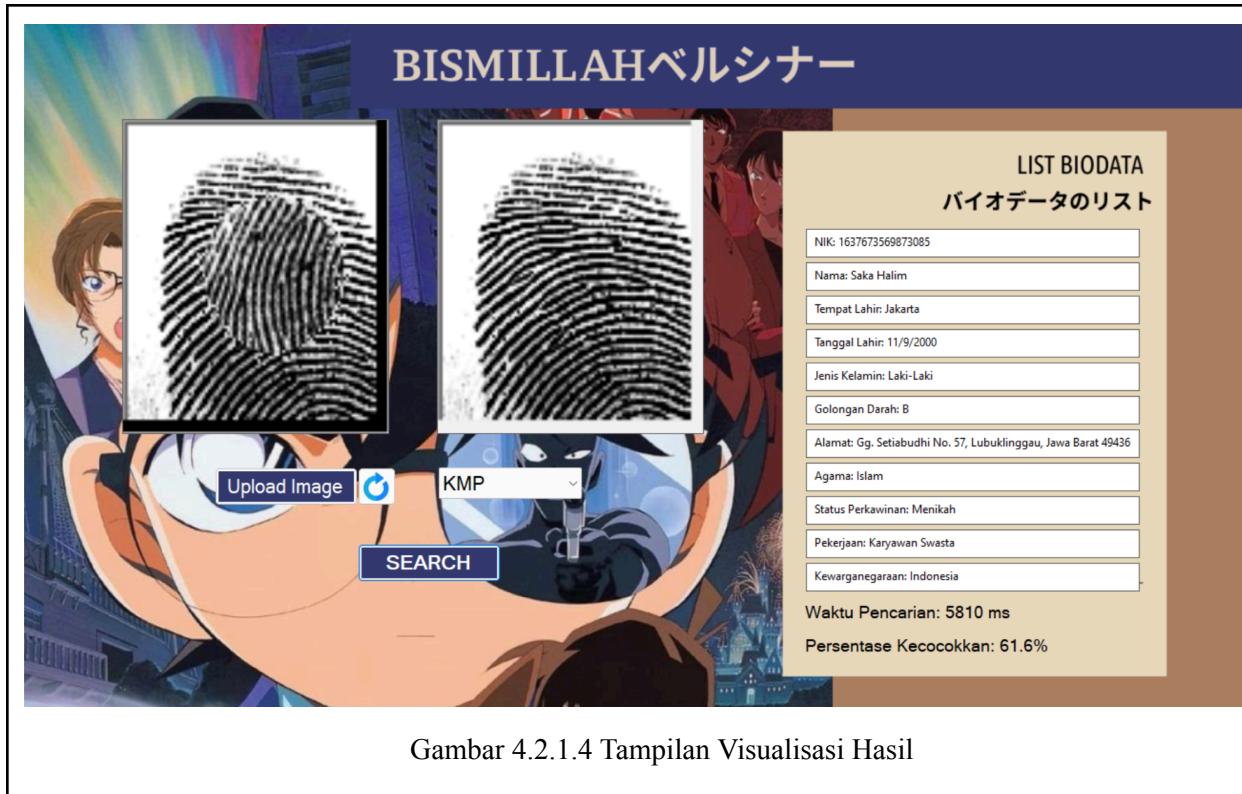




Gambar 4.2.1.2 Tampilan menerima masukan



Gambar 4.2.1.3 Tampilan Belum Memasukkan Input



Gambar 4.2.1.4 Tampilan Visualisasi Hasil

## 2.2. Tata Cara Penggunaan Program

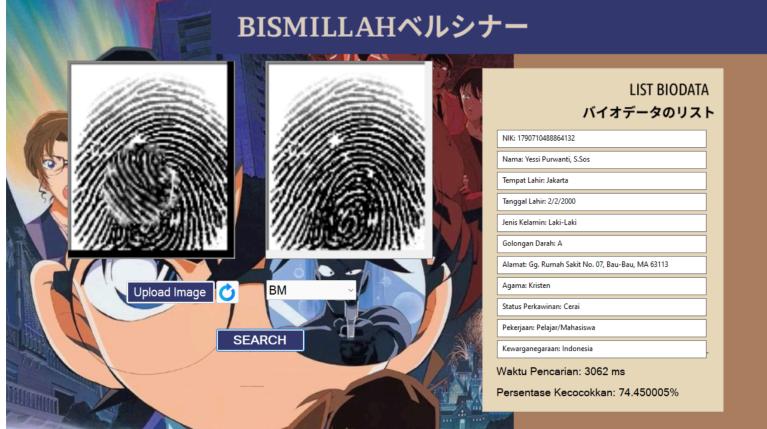
Sebelum menjalankan program, terlebih dahulu lakukan clone pada repository github tugas besar ini : [https://github.com/shulhajws/Tubes3\\_BismillahBersinar](https://github.com/shulhajws/Tubes3_BismillahBersinar). Berikut tata cara menjalankan program melalui terminal :

1. Install .NET 8.0 pada komputer Anda.
2. Buka folder tugas besar ini.
3. Pindah ke direktori “src/FingerPrintDetector”.
4. Pada terminal, masukkan perintah ‘dotnet build’ untuk mengkompilasi proyek dan semua dependensinya.
5. Jika program sudah selesai melakukan proses build, masukkan perintah ‘dotnet run’ pada terminal untuk menjalankan aplikasi pada window baru.
6. Setelah aplikasi telah terbuka, Anda bisa langsung memulai pencarian biodata dari sidik jari yang Anda miliki.

7. Untuk mematikan Aplikasi, tekan Control+C atau tekan tombol silang di pojok kanan atas.

### 3. Hasil Pengujian

#### a. Hasil Uji 1

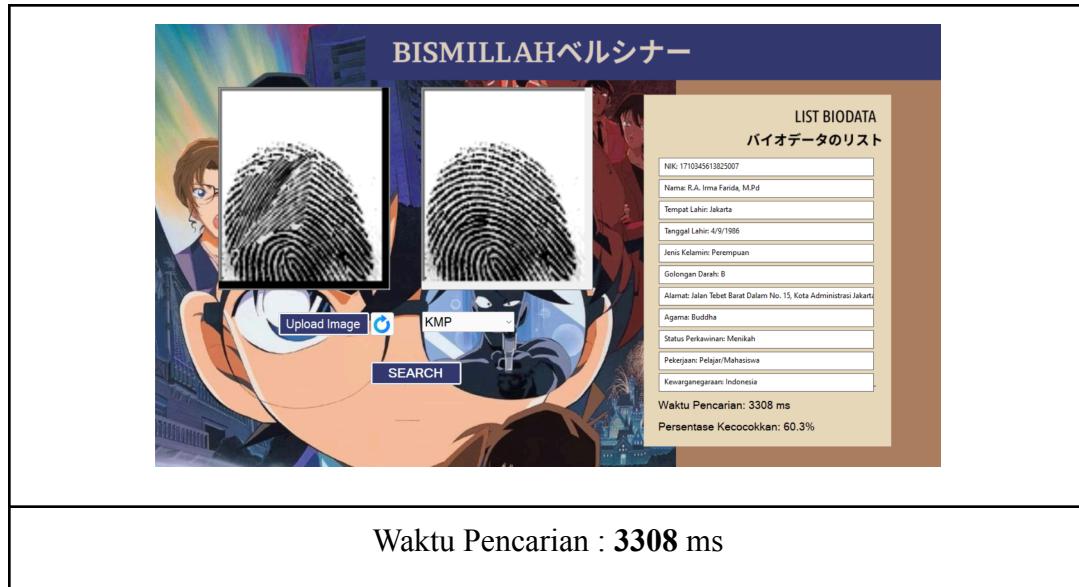
<b>Pattern</b>	<u>ü</u> <u>ù</u> <u>ü</u> <u>÷</u> <u>ÿ</u> <u>ü</u> <u>ô</u> <u>û</u> <u>y</u> <u>ù</u> <u>_</u> <u>û</u> <u>ñ</u> <u>ò</u> <u>o</u> <u>ò</u> <u>y</u> <u>÷</u> <u>í</u> <u>é</u>
<b>Similarity</b>	<b>74.450005%</b>
<b>Algoritma BM</b>	
	
Waktu Pencarian : <b>3062 ms</b>	
<b>Algoritma KMP</b>	
	

Waktu Pencarian : 4745 ms

Berdasarkan hasil pengujian 1 didapat waktu pencarian dengan menggunakan algoritma BM (3062 ms) lebih cepat dibandingkan dengan menggunakan algoritma KMP (4745 ms) karena algoritma BM menggunakan heuristik yang memungkinkan lompatan lebih besar selama pencocokan terjadi, sehingga mengurangi jumlah perbandingan karakter yang diperlukan. Sedangkan untuk patternnya yang dihasilkan kedua algoritma tersebut sama serta nilai similaritynya juga sama yaitu 74.450005%.

b. Hasil Uji 2

<b>Pattern</b>	Y_Dæo_yAñôc_Ñyo◆↑?áàíü?¼?ù?^ô
<b>Similarity</b>	<b>60.3%</b>
<b>Algoritma BM</b>	
	
Waktu Pencarian : 3026 ms	
<b>Algoritma KMP</b>	



Berdasarkan hasil pengujian 2 didapat waktu pencarian dengan menggunakan algoritma BM (3026 ms) lebih cepat dibandingkan dengan menggunakan algoritma KMP (3308 ms) karena algoritma BM menggunakan heuristic yang memungkinkan lompatan lebih besar selama pencocokan terjadi, sehingga mengurangi jumlah perbandingan karakter yang diperlukan. Sedangkan untuk patternnya yang dihasilkan kedua algoritma tersebut sama serta nilai similaritynya juga sama yaitu 60.3%.

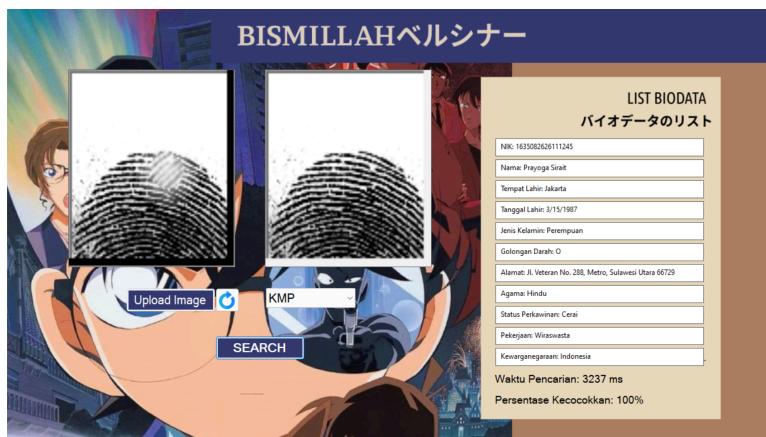
### c. Hasil Uji 3

Algoritma BM



Waktu Pencarian : **2996 ms**

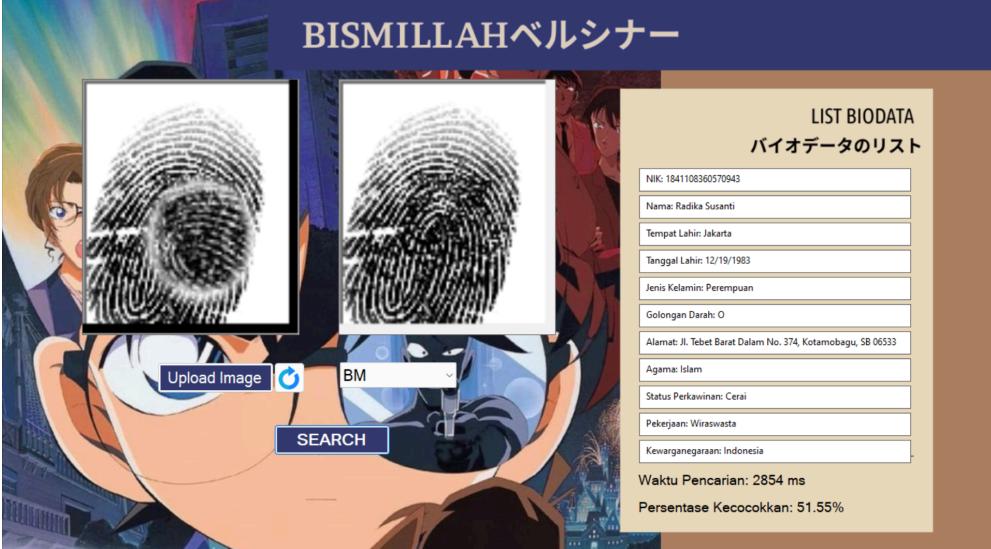
## Algoritma KMP

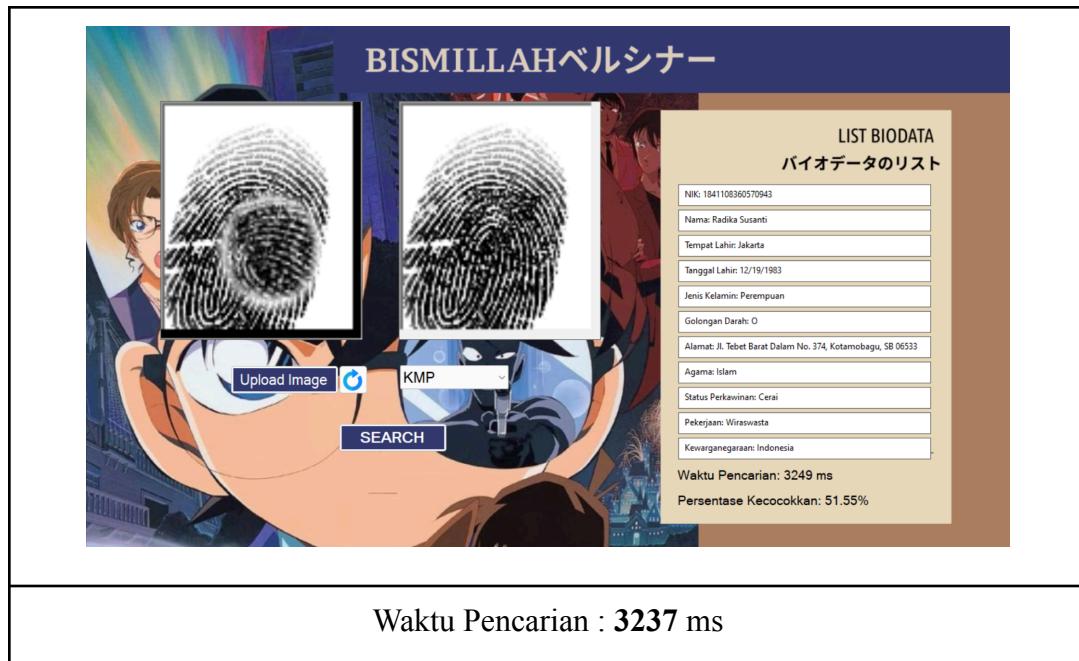


Waktu Pencarian : **3237** ms

Berdasarkan hasil pengujian 3 didapat waktu pencarian dengan menggunakan algoritma BM (2996 ms) lebih cepat dibandingkan dengan menggunakan algoritma KMP (3237 ms) karena algoritma BM menggunakan heuristik yang memungkinkan lompatan lebih besar selama pencocokan terjadi, sehingga mengurangi jumlah perbandingan karakter yang diperlukan. Sedangkan untuk patternnya yang dihasilkan kedua algoritma tersebut sama serta nilai similaritynya juga sama yaitu 100%.

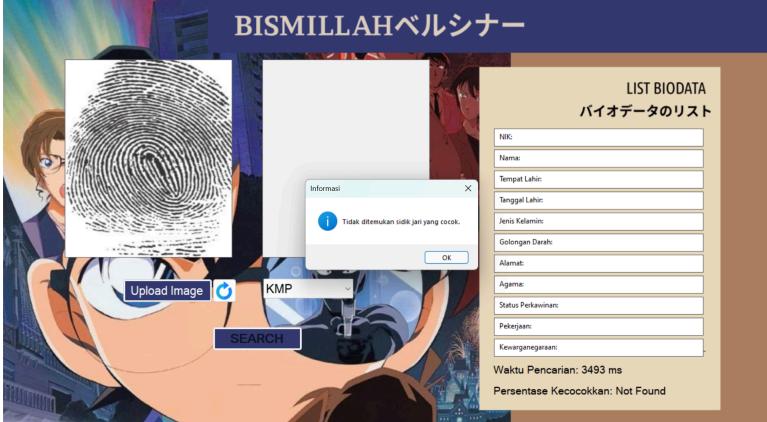
d. Hasil Uji 4

<b>Pattern</b>	y_úöüûú_üòßyéyy_yùé_UUúEOáûoö÷
<b>Similarity</b>	<b>51.55%</b>
<b>Algoritma BM</b>	
	
Waktu Pencarian : <b>2854 ms</b>	
<b>Algoritma KMP</b>	



Berdasarkan hasil pengujian 4 didapat waktu pencarian dengan menggunakan algoritma BM (2854 ms) lebih cepat dibandingkan dengan menggunakan algoritma KMP (3237 ms) karena algoritma BM menggunakan heuristik yang memungkinkan lompatan lebih besar selama pencocokan terjadi, sehingga mengurangi jumlah perbandingan karakter yang diperlukan. Sedangkan untuk patternnya yang dihasilkan kedua algoritma tersebut sama serta nilai similaritynya juga sama yaitu 51.55%.

e. Hasil Uji 5

Algoritma BM		Waktu Pencarian : 3353 ms
		
Algoritma KMP		Waktu Pencarian : 3493 ms
		

Berdasarkan hasil pengujian 5 didapat waktu pencarian dengan menggunakan algoritma BM (3353 ms) lebih cepat dibandingkan dengan menggunakan algoritma KMP (3493 ms) karena algoritma BM menggunakan heuristik yang memungkinkan lompatan lebih besar selama pencocokan terjadi, sehingga mengurangi jumlah perbandingan karakter yang diperlukan. Sedangkan untuk patternnya yang dihasilkan kedua algoritma tersebut sama. Program memunculkan popup tidak ditemukan sidik jari yang cocok karena nilai kemiripan

nya kurang dari 50% (sesuai batas bawah kecocokan program yang kami tetapkan pada keterangan sebelumnya).

#### 4. Analisis Hasil Pengujian

Waktu Uji (ms)					
Hasil Uji	1	2	3	4	5
KMP	4745	3308	3308	3237	3496
BM	3062	3026	2996	2854	3353

Tabel 4.1 Hasil Pengujian

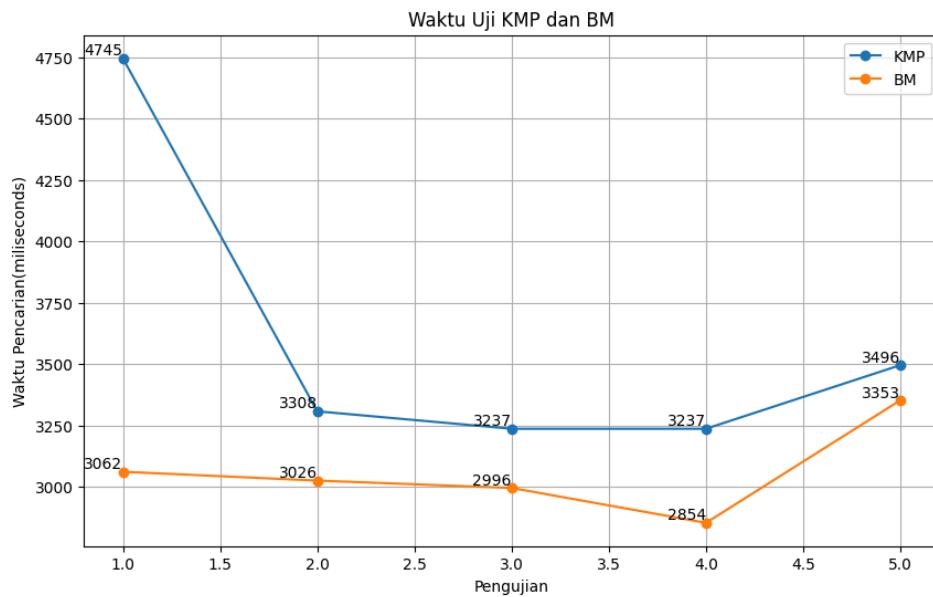
Dalam pengembangan sistem deteksi individu berbasis biometrik melalui citra sidik jari, pemanfaatan algoritma pattern matching seperti Boyer-Moore dan Knuth-Morris-Pratt (KMP) sangat penting. Kedua algoritma ini memiliki cara kerja dan keunggulan yang berbeda, yang dapat mempengaruhi efisiensi dan kecepatan proses pencocokan pola pada citra sidik jari.

Algoritma Boyer-Moore dikenal karena efisiensinya dalam pencarian teks panjang. Algoritma ini dapat melakukan character jump yang memungkinkan algoritma untuk melompati sejumlah karakter yang tidak mungkin mencocokkan pola saat terjadi ketidakcocokan. Oleh karena itu, Boyer-Moore untuk melompat lebih jauh dalam teks, mengurangi jumlah perbandingan yang perlu dilakukan secara signifikan. Ini membuat Boyer-Moore sangat efisien dalam menangani teks panjang atau pola yang panjang dan jarang. Dalam konteks citra sidik jari, jika pola sidik jari yang dicari memiliki karakteristik unik dan jarang muncul, Boyer-Moore bisa sangat efektif. Kecepatan pencarian yang lebih tinggi dapat mengurangi waktu komputasi.

Knuth-Morris-Pratt (KMP) menggunakan pendekatan yang berbeda dengan memanfaatkan tabel prefix function. Tabel ini menyimpan informasi tentang panjang prefiks terpanjang dari pola yang juga merupakan sufiks. Selama proses pencocokan, ketika terjadi ketidakcocokan, KMP menggunakan tabel untuk menghindari

perbandingan ulang dari karakter yang sudah dicocokkan sebelumnya. KMP sangat efisien dalam menangani pola dengan banyak kesamaan internal (prefiks dan sufiks yang berulang). Ketika terjadi ketidakcocokan, algoritma hanya perlu melihat tabel dan melompat ke posisi yang sesuai, sehingga mengurangi jumlah perbandingan yang perlu dilakukan. Dalam aplikasi citra sidik jari, jika pola sidik jari yang dicari memiliki banyak kesamaan internal atau struktur berulang, KMP bisa lebih efektif daripada Boyer-Moore. Penggunaan tabel pi membantu mengurangi redundansi perbandingan dan memastikan pencarian yang lebih cepat dan efisien.

Dalam sistem deteksi individu berbasis biometrik melalui citra sidik jari, pemilihan antara algoritma Boyer-Moore dan KMP sangat bergantung pada karakteristik spesifik dari pola sidik jari yang dianalisis dan konteks penggunaannya. Jika pola sidik jari panjang dan memiliki karakteristik yang jarang, Boyer-Moore biasanya lebih efisien karena dapat melompati lebih banyak karakter. Jika pola memiliki banyak kesamaan internal, KMP dapat mengurangi jumlah perbandingan yang diperlukan, sehingga lebih efisien dalam skenario ini.



Gambar 4.1 Grafik Waktu Pencarian

Berdasarkan 5 pengujian yang dilakukan (bisa lihat grafik diatas), algoritma Boyer-Moore(BM) memberikan performa yang lebih baik dibandingkan dengan algoritma Knuth-Morris-Pratt (KMP) dalam pencarian pola pada sidik jari. Dapat ditinjau dari tabel waktu uji yang dihasilkan BM lebih cepat dibandingkan KMP. Hal ini disebabkan oleh teknik character-jump yang mengurangi jumlah perbandingan karakter yang diperlukan dengan menggunakan tabel last occurrence. Hasil pattern yang didapat dalam pengujian rata-rata memiliki pola yang sangat variatif dan hanya sedikit pola berulang yang ditemukan. Hal ini membuat algoritma KMP memerlukan waktu yang lebih lama dalam melakukan proses pencarian karena harus mencocokkan karakter satu per satu dari awal.

Meski begitu terdapat beberapa kasus di mana hasil *pattern* yang didapat memiliki pola yang sama walau tidak banyak. Hal ini terdapat pada hasil uji 4 terdapat pengulang untuk pola “y\_” sehingga waktu yang dibutuhkan untuk perbandingan lebih sedikit dibandingkan dengan *pattern* yang memiliki pola yang sangat bervariatif seperti pada kasus uji coba 2. Karena sangat bervariasi, waktu yang dibutuhkan untuk melakukan perbandingan juga menjadi lebih lama. Meskipun terdapat perulangan pola, perulangan pola yang terjadi sangatlah minim sehingga waktu yang dibutuhkan secara keseluruhan tetap lebih lama ditambah lagi text yang dibanding memiliki panjang yang jauh lebih besar dari *pattern* yang dibandingkan.

Karena ukuran text yang panjang dibandingkan ukuran *pattern* serta pola pattern yang sangat variatif membuat algoritma BM merupakan algoritma yang paling efisien dalam menyelesaikan persoalan ini. Walaupun mungkin di beberapa kasus ditemukan pola yang berulang, namun secara keseluruhan, frekuensi perulangan ini sangat rendah. Hal ini menyebabkan keuntungan penggunaan tabel prefix pada algoritma KMP tidak dapat dimanfaatkan secara optimal. Sebaliknya, algoritma Boyer-Moore dengan teknik character-jump mampu melompati sejumlah karakter dalam sekali langkah, mengurangi jumlah perbandingan yang harus dilakukan dan mempercepat proses pencarian secara signifikan.

## BAB V

### PENUTUP

#### 1. Kesimpulan

Terdapat kesimpulan dari hasil pembuatan dan analisis dari aplikasi yang dibuat sebagai berikut:

1. Algoritma Boyer-Moore(BM) lebih cepat dalam pencarian pola pada sidik jari dibandingkan dengan KMP, terutama karena heuristik character-jump yang memungkinkan lompatan lebih besar dan mengurangi jumlah perbandingan karakter yang diperlukan. Akan tetapi, efektivitas BM juga bergantung dengan distribusi karakter teksnya.
2. Kedua Algoritma menunjukkan akurasi pencocokan yang sama, terbukti dengan nilai similarity yang identik dalam setiap pengujian.

#### 2. Saran

Terdapat beberapa saran yang dapat dilakukan untuk pengembangan lebih lanjut, antara lain:

1. Aplikasi dapat dibuat lebih responsive sehingga meningkatkan pengalaman pengguna
2. Membuat algoritma menjadi lebih efisien agar bisa mempercepat waktu pencarian program.

#### 3. Tanggapan dan Refleksi

Dari hasil tugas besar 3IF2211 Strategi Algoritma ini, penulis memiliki tanggapan dan refleksi sebagai berikut.

1. Penulis berhasil mengimplementasikan algoritma Knuth-Morris-Pratt (KMP), algoritma Boyer-Moore (BM), dan algoritma Levenshtein Distance beserta Hamming dalam pembuatan aplikasi.

2. Penulis berhasil menggunakan Regular Expression (regex) untuk mengidentifikasi mengubah data *corrupt* berupa nama dengan bahasa alay ke bahasa sebenarnya
3. Penulis mendapatkan pengalaman bekerja secara berkelompok dalam membangun aplikasi menggunakan bahasa pemrograman c#.
4. Penulis berhasil membuat database untuk mendukung aplikasi yang kami buat.

## **LAMPIRAN**

### **1. Tautan Repository Github**

[https://github.com/shulhajws/Tubes3\\_BismillahBersinar](https://github.com/shulhajws/Tubes3_BismillahBersinar)

### **2. Tautan Video Bonus**

<https://youtu.be/JgPFh2PeHUG>

## REFERENSI

Munir, R. (2021). *Pencocokan string*. Institut Teknologi Bandung. Diakses dari <https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Pencocokan-string-2021.pdf>

Munir, R. (2019). *String Matching dengan Regex*. Institut Teknologi Bandung. Diakses dari <https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2022-2023/String-Matching-dengan-Regex-2019.pdf>

Serjey, G. (2023). *Levenshtein Distance Computation*. Baeldung CS. Diakses dari <https://www.baeldung.com/cs/levenshtein-distance-computation#:~:text=Levenshtein%20distance%20is%20the%20smallest,insertions%2C%20deletions%2C%20and%20substitutions.>

Raut, N. (2023). *What is Hamming Distance?*. tutorials point. Diakses dari <https://www.tutorialspoint.com/what-is-hamming-distance>

GeeksforGeeks. (2024). *Longest Common Subsequence (LCS)*. Diakses dari <https://www.geeksforgeeks.org/longest-common-subsequence-dp-4/>