

LAPORAN TUGAS BESAR III
IF2211 STRATEGI ALGORITMA

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



Dosen:

Ir. Rila Mandala, M. Eng, Ph. D.
Monterico Adrian, S. T., M. T
Dr. Ir. Rinaldi Munir, M.T..

Kelompok Thumbylicious:

13522121 Jonathan Emmanuel Saragih
13522139 Attara Majesta Ayub
13522156 Jason Fernando

PROGRAM STUDI TEKNIK INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG
SEMESTER II TAHUN 2023/2024

DAFTAR ISI

DAFTAR ISI	2
BAB 1	
DESKRIPSI MASALAH	3
BAB 2	
TEORI DASAR	4
2.1 Algoritma Knuth-Morris-Path	4
2.2 Algoritma Boyer Moore	4
2.3 Regular Expression	4
2.4 Pengukuran Persentase Kemiripan: Levenshtein Distance	4
2.5 Aplikasi dengan WPF	4
BAB 3	
ANALISIS PEMECAHAN MASALAH	5
3.1 Langkah - Langkah Pemecahan Masalah	5
3.2 Proses Penyelesaian Solusi Menggunakan Algoritma KMP	5
3.3 Proses Penyelesaian Solusi Menggunakan Algoritma BM	5
3.4 Proses Penyelesaian Solusi Menggunakan Algoritma Levenshtein Distance	5
3.5 Fitur Fungsional dan Arsitektur Aplikasi Desktop Yang Dibangun	5
3.6 Contoh Ilustrasi Kasus	5
BAB 4	
IMPLEMENTASI DAN PENGUJIAN	6
4.1 Spesifikasi Teknis Program	6
4.1.1 Struktur Data	6
4.2.2 Fungsi & Prosedur	6
4.2 Dokumentasi Pengguna	6
4.3 Hasil pengujian	6
4.4 Analisis Hasil Pengujian	6
BAB 5	
PENUTUP	7
5.1 Kesimpulan	7
5.2 Saran	7

5.3 Refleksi	7
LAMPIRAN	8
Daftar Pustaka	9

BAB 1

DESKRIPSI TUGAS



Gambar 1. Ilustrasi *fingerprint recognition* pada deteksi berbasis biometrik.

Sumber: <https://www.aratek.co/news/unlocking-the-secrets-of-fingerprint-recognition>

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

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

Dengan menggabungkan teknologi identifikasi sidik jari dan *pattern matching*, dimungkinkan untuk membangun sistem identifikasi biometrik yang aman, handal, dan mudah digunakan. Sistem ini dapat diaplikasikan di berbagai bidang, seperti kontrol akses, absensi karyawan, dan verifikasi identitas dalam transaksi keuangan.

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

BAB 2

TEORI DASAR

2.1 Algoritma Knuth-Morris-Path

Algoritma Knuth-Morris-Pratt (KMP) adalah algoritma pencocokan string yang ditemukan oleh Donald Knuth, Vaughan Pratt, dan James H. Morris pada tahun 1977. Algoritma ini dirancang untuk mencari kemunculan sebuah substring (pola) dalam string lain (teks) dengan cara yang efisien. KMP menghindari pemeriksaan ulang karakter yang sudah diketahui cocok, sehingga mengurangi jumlah perbandingan yang diperlukan. Algoritma KMP dinilai cukup efisien karena algoritma ini hanya membutuhkan waktu $O(m + n)$ untuk preprocessing dan pencocokan, membuatnya sangat efektif untuk pencocokan string panjang, dengan n adalah panjang teks dan m adalah panjang pola. KMP memiliki penggunaan waktu yang singkat, menghindari pemeriksaan secara berulang, konsisten, akurat, dan memiliki skalabilitas untuk dataset besar. Penjelasan Algoritma KMP (Knuth-Morris-Pratt)

Algoritma KMP bekerja berdasarkan prinsip "praproses" terhadap pola yang akan dicari. Ini berarti sebelum melakukan pencarian pola, KMP membangun sebuah tabel yang disebut sebagai tabel "lps" (longest proper prefix which is also suffix). Tabel lps ini menyimpan panjang dari proper prefix (prefix yang bukan seluruhnya) dari pola yang juga merupakan suffix.

Terdapat beberapa langkah untuk mengimplementasikan Algoritma KMP. Pertama, membangun tabel lps untuk menyimpan panjang proper prefix dari pola yang juga merupakan suffix. Proper prefix adalah bagian awal dari string yang juga merupakan akhir dari string, tetapi tidak termasuk string itu sendiri secara keseluruhan. Setelah tabel lps terbentuk, kita dapat melakukan pencarian pola dalam teks. Ketika terjadi ketidakcocokan karakter antara teks dan pola, tabel lps digunakan untuk menggeser pola sehingga kita tidak perlu mencocokkan ulang karakter yang sudah kita ketahui cocok.

Misalnya, jika terdapat pola "AABAACAAABAA". Untuk pola tersebut, tabel lps akan terlihat seperti berikut:

Index	0	1	2	3	4	5	6	7	8	9	10
-------	---	---	---	---	---	---	---	---	---	---	----

Lps	0	1	0	1	2	0	1	2	3	4	5
-----	---	---	---	---	---	---	---	---	---	---	---

Penjelasan:

- Pada indeks 0, proper prefix dan suffix tidak ada, sehingga $\text{lps}[0] = 0$.
- Pada indeks 1, proper prefix "A" dan suffix "A" cocok, sehingga $\text{lps}[1] = 1$.
- Pada indeks 2, tidak ada proper prefix yang cocok dengan suffix, sehingga $\text{lps}[2] = 0$.
- Pada indeks 3, proper prefix "AB" dan suffix "AB" cocok, sehingga $\text{lps}[3] = 2$.
- Pada indeks 4, tidak ada proper prefix yang cocok dengan suffix, sehingga $\text{lps}[4] = 0$.
- Pada indeks 5, tidak ada proper prefix yang cocok dengan suffix, sehingga $\text{lps}[5] = 0$.
- Pada indeks 6, proper prefix "A" dan suffix "A" cocok, sehingga $\text{lps}[6] = 1$.
- Pada indeks 7, proper prefix "AB" dan suffix "AB" cocok, sehingga $\text{lps}[7] = 2$.
- Pada indeks 8, proper prefix "ABA" dan suffix "ABA" cocok, sehingga $\text{lps}[8] = 3$.

2.2 Algoritma Boyer Moore

Algoritma Boyer-Moore adalah salah satu algoritma pencocokan string yang paling efisien dalam praktik, ditemukan oleh Robert S. Boyer dan J Strother Moore pada tahun 1977. Algoritma ini bekerja dengan melakukan pencocokan dari kanan ke kiri dan menggunakan dua aturan utama untuk menggeser pola lebih jauh dari satu karakter setelah terjadi ketidakcocokan, yaitu "bad character rule" dan "good suffix rule".

Bad Character Rule berarti jika terjadi mismatch antara karakter dalam pola dan teks, BM menggunakan informasi tentang karakter yang mismatched untuk menggeser pola ke depan. BM menciptakan tabel dari pola yang mencatat posisi terakhir dari setiap karakter yang terjadi di pola (last occurrence table). Jika terjadi mismatch, pola bisa digeser ke depan ke posisi setelah kemunculan terakhir dari karakter yang mismatched. Sedangkan aturan Good Suffix Rule berarti jika terjadi mismatch, pola digeser ke posisi berikutnya di mana substring dari pola yang sudah cocok dengan bagian teks yang sama ditemukan. Jika tidak ada kesamaan yang ditemukan, pola digeser ke panjang keseluruhan dari substring yang cocok.

Langkah-langkah Boyer-Moore antara lain membuat tabel Bad Character dan tabel Good Suffix dari pola. Kemudian mulai pencocokan dari akhir pola ke awal. Jika ada mismatch, gunakan aturan Bad Character atau Good Suffix untuk menentukan berapa banyak pola harus digeser ke kanan.

Misalnya kita ingin mencari pola "EXAMPLE" dalam teks "HERE IS A SIMPLE EXAMPLE". Maka Tabel Bad Character untuk pola "EXAMPLE":

E	X	A	M	P	L	Lainnya
6	5	4	3	2	1	7

Saat mencocokkan dari kanan ke kiri:

- Mismatch pada karakter pertama dari teks setelah memeriksa dari kanan.
- Gunakan tabel Bad Character untuk memutuskan berapa banyak geser.
- Teruskan sampai pola ditemukan atau teks habis.

Algoritma Boyer-Moore memiliki banyak keunggulan. Secara teori, BM lebih efisien dibandingkan dengan KMP. Selain itu, Algoritma Boyer-Moore lebih efisien dibandingkan dengan algoritma naive dan sering digunakan dalam aplikasi seperti editor teks dan sistem deteksi pola. Algoritma ini memiliki jumlah perbandingan yang tidak banyak juga kecepatan pencocokan yang relatif cepat dan akurat.

2.3 Regular Expression

Regular Expression (Regex) adalah serangkaian karakter yang menentukan pola dalam teks (string) tertentu. Regex dapat digunakan untuk mencari, mengekstrak, memvalidasi, atau mengubah teks berdasarkan pola yang ditentukan. Pola ini bisa sesederhana mencari tanda koma (,) dalam teks atau serumit mencari alamat email yang valid dalam teks.

Regex sangat berguna dalam berbagai bidang seperti Pemrosesan Bahasa Alami (NLP), Web Scraping, Validasi Data, Pencarian dan Manipulasi Teks, dan lain-lain. Regex dibutuhkan oleh Pengembang Perangkat Lunak, Analis Data, atau Insinyur Pembelajaran Mesin pada satu waktu atau lainnya. Faktanya, Regex hampir tak terhindarkan dalam pekerjaan mereka. Berikut adalah komponen-komponen RegEx beserta penjelasannya.

Elemen	Simbol	Deskripsi	Contoh
Asterisk	*	Mencocokkan karakter sebelumnya 0 atau lebih kali.	<code>ab*c</code> matches "ac", "abc", "abbc", etc.
Plus	+	Mencocokkan karakter sebelumnya 1 atau lebih kali.	<code>ab+c</code> matches "abc", "abbc", "abbcc", etc.
Bracket	{}	Menyatakan jumlah pengulangan karakter sebelumnya sesuai dengan nilai di dalam kurung.	<code>a{2}</code> matches "aa"; <code>a{2, }</code> matches "aa", "aaa", etc.; <code>a{2,3}</code> matches "aa" or "aaa".
Wildcard	.	Mencocokkan sembarang karakter.	<code>a.c</code> matches "abc", "a2c", "a-c", etc
Karakter Opsional	?	Karakter sebelumnya mungkin ada atau tidak ada.	<code>docx?</code> matches "doc" or "docx".
Caret	^	Menyatakan bahwa pencocokan harus dimulai dari awal string.	<code>^\d{3}</code> matches "901" in "901-333-".
Dolar	\$	Menyatakan bahwa pencocokan harus terjadi di akhir string.	<code>-\d{3}\$</code> matches "-333" in "901-333".
Character Classes	\s, \d, \w	Mencocokkan jenis karakter tertentu seperti spasi, digit,	<code>[abc]</code> matches "a", "b", or "c".

		atau karakter kata.	
Negation	[^]	Mencocokkan karakter selain yang ada di dalam set.	[^abc] matches any character except "a", "b", or "c".
Character Range	[a-z]	Mencocokkan karakter dalam rentang tertentu.	[a-zA-Z] matches any letter from "a" to "z" or "A" to "Z".
Escape Symbol	\	Mencocokkan karakter spesial secara literal.	\d+[\+-**]\d+ matches "2+2" and "3*9".
Grouping	()	Mengelompokkan bagian dari ekspresi reguler.	([A-Z] \w+) matches "AWord" starting with an uppercase letter followed by word characters.
Alternation		Menyatakan bahwa minimal sesuai dengan salah satu alternatif	th(e is at) matches "the", "this", or "that".
Backreference	\number	Mengacu pada sub-ekspresi yang dicocokkan sebelumnya.	([a-z]) \1 matches "ee" in "Geek".
Komentar	(?x)	Menambahkan komentar dalam pola.	(?x) \bA\w+\b # Matches words starting with A.

2.4 Levenshtein Distance

Levenshtein Distance didefinisikan sebagai langkah operasi minimum untuk mentransformasi sebuah string menjadi string lainnya. Operasi termasuk insersi, delesi, dan substitusi. Levenshtein Distance juga dapat diinterpretasikan sebagai kemiripan dua buah string, karena semakin mirip dua string, maka operasinya akan semakin sedikit. Levenshtein Distance adalah cara pengukuran jarak paling populer diantara algoritma sejenis, walaupun perbedaannya hanya dalam jenis operasi yang diperbolehkan. Algoritma Hamming Distance hanya mengizinkan operasi substitusi. Sementara itu, Damerau-Levenshtein Distance mengizinkan transposisi karakter selain set yang didefinisikan oleh Levenshtein Distance. Ini sering digunakan sebagai pengganti jarak Levenshtein klasik dengan nama yang sama. Berikut adalah definisi formal dari Levenshtein Distance antara dua string a, b dalam $\text{lev}_{a,b}(|a|, |b|)$ dimana

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

Gambar 2. Formal Levenshtein Distance and Properties

(Sumber: <https://www.baeldung.com/cs/levenshtein-distance-computation>)

Misalnya, terdapat String A: "kitten" yang perlu diubah menjadi String B: "sitting" sehingga perlu ditentukan operasi minimum nya:

- kitten → sitten (substitusi "s" untuk "k")
- sitten → sittin (substitusi "i" untuk "e")
- sittin → sitting (penyisipan "g" di akhir).

Dalam kasus di atas, diperlukan 3 operasi untuk mengubah string A menjadi string B.

Implementasi Levenshtein Distance dapat menggunakan pendekatan rekursif, iteratif dengan matriks 2D, atau hanya dua baris matriks. Dua baris matriks digunakan untuk mengurangi *space complexity*. Kompleksitas waktu dari algoritma ini adalah $O(3^{(m+n)})$ dengan m adalah panjang string pertama dan n adalah panjang string kedua. Sedangkan kompleksitas ruang tambahan (*auxiliary space*) nya adalah $O(m+n)$.

2.5 Aplikasi Dengan WPF

Windows Presentation Foundation (WPF) adalah kerangka antarmuka pengguna (UI) yang free dan *open-source* yang dirancang untuk aplikasi desktop berbasis Windows. WPF menggunakan bahasa XAML (eXtensible Application Markup Language), yang berbasis XML, untuk mendefinisikan dan menghubungkan berbagai elemen antarmuka. Aplikasi WPF dikembangkan menggunakan bahasa pemrograman C# untuk logika programnya.

Dirilis pertama kali oleh Microsoft sebagai bagian dari .NET Framework 3.0 pada tahun 2006, WPF kemudian dibuka sebagai sumber terbuka di bawah Lisensi MIT pada tahun 2018. WPF mendukung berbagai fitur antarmuka pengguna seperti rendering 2D/3D, dokumen tetap dan adaptif, tipografi, grafik vektor, animasi waktu-eksekusi, dan media yang telah dirender sebelumnya. Aplikasi WPF dapat didistribusikan sebagai program desktop mandiri dan library runtime WPF sudah termasuk dalam semua versi Windows sejak Windows Vista dan Windows Server 2008.

Meskipun tersedia secara luas untuk proyek yang ditargetkan ke kerangka kerja perangkat lunak .NET, WPF tetap terbatas pada platform Windows dan tidak tersedia lintas platform. Meskipun demikian, desain dan bahasa XAML WPF telah mempengaruhi pengembangan berbagai kerangka antarmuka pengguna lainnya dalam ekosistem .NET seperti UWP, .NET MAUI, dan Avalonia.

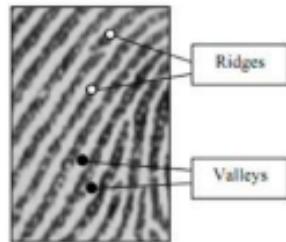
BAB 3

ANALISIS PEMECAHAN MASALAH

3.1 Langkah - Langkah Pemecahan Masalah

Dalam implementasi sistem identifikasi sidik jari berbasis biometrik melibatkan serangkaian proses dari pengambilan gambar sidik jari hingga pencocokan dan penyimpanan data dalam basis data. Berikut adalah langkah-langkah tersebut:

1. Pengambilan Gambar Sidik Jari: Pengguna mengunggah gambar sidik jari dengan ekstensi yang telah ditentukan (*.jpg, *.png, atau *.BMP) melalui antarmuka aplikasi yang disediakan.
2. Pengambilan 30 pixel: Diambil 30 pixel dengan cara me-*resize* gambar. Hal ini bertujuan untuk mempercepat proses perbandingan sebanyak 30x.
3. Konversi Gambar ke ASCII: Sidik jari diproses dengan mengubah gambar menjadi *grayscale*, lalu mengonversinya menjadi bentuk biner berdasarkan gelap (*ridge*) dan terang (*valley*).



Gambar 3. Ridge and Valley

(Sumber: [OnA Stima 2024](#))

Setelah itu, setiap 8-bit bilangan biner akan dikonversi menjadi ASCII untuk mempersingkat pola yang akan dicocokkan dengan sidik jari dalam basis data.

4. Pengambilan data dari MySQL: Membuat fungsi dengan query MySQL untuk mendapatkan berkas_citra dan nama dari tabel sidik_jari. Setelah itu, data akan disimpan dalam *dictionary*.
5. Pencocokan Sidik Jari: Pola ASCII dari sidik jari input dicocokkan dengan pola yang telah disimpan dalam *dictionary*. Pencocokan dilakukan dengan algoritma Knuth-Morris-Pratt (KMP) dan Boyer-Moore (BM) untuk tingkat similaritas 100%.

- Sementara itu, digunakan algoritma Levenshtein Distance untuk similaritas tidak sempurna.
6. Pengambilan Data Pengguna: Ditampilkan 5 data pertama dengan tingkat similaritas tertinggi dengan query yang mengakses MySQL dalam tabel biodata..
 7. Integrasi dan Antarmuka Pengguna:
Antarmuka aplikasi harus dirancang agar mudah digunakan dan memberikan instruksi yang jelas kepada pengguna.

3.2 Proses Penyelesaian Masalah Dengan Algoritma Knuth-Morris-Path

Dalam program ini, Algoritma Knuth-Morris-Path ini terdiri dari dua tahap utama, yaitu preprocessing pola untuk membuat array Longest Prefix Suffix (LPS) dengan fungsi [computeLPSArray\(\)](#) dan pencarian pola dalam teks menggunakan array LPS tersebut dengan fungsi [KMPSearch\(\)](#).

Variabel ‘pat’ mewakili pola yang akan dicari, sedangkan ‘txt’ mewakili teks di mana pola akan dicari. M adalah panjang dari pola, dan N adalah panjang dari teks. Selanjutnya, dibuat array lps yang akan menyimpan nilai-nilai LPS untuk setiap indeks dalam pola. Variabel j digunakan sebagai indeks untuk pola, sedangkan i sebagai indeks untuk teks.

Fungsi [computeLPSArray\(\)](#) menghitung nilai LPS yang menunjukkan panjang prefix yang sama dengan suffix pada pola hingga indeks tertentu. Jika karakter pada pola cocok dengan karakter pada indeks sebelumnya yang dicocokkan, nilai len diinkrement dan disimpan di lps[i], kemudian i juga diinkrement. Jika tidak cocok dan len tidak nol, len diperbarui ke nilai lps[len-1] tanpa meningkatkan i. Jika len nol, nilai lps[i] diset ke nol dan i ditingkatkan.

Setelah array LPS dihitung, langkah selanjutnya adalah melakukan pencarian pola dalam teks. Proses ini dimulai dengan inisialisasi i dan j ke nol. Karakter pada pola dan teks dibandingkan satu per satu. Jika cocok, baik i maupun j diinkrement. Jika j mencapai panjang pola, berarti pola ditemukan dan fungsi mengembalikan nilai true. Jika tidak cocok dan j tidak nol, j diset ke nilai lps[j-1]. Jika j nol, i ditingkatkan. Hasil akhir dari pencarian adalah jika pola tidak ditemukan setelah seluruh iterasi selesai, fungsi mengembalikan nilai false.

3.3 Proses Penyelesaian Masalah Dengan Algoritma Boyer-Moore

Implementasi algoritma Boyer-Moore dalam kode ini berfokus pada penggunaan heuristik karakter buruk untuk pencarian pola dalam teks. Kode ini terdiri dari dua fungsi utama yaitu `badCharHeuristic()` untuk preprocessing pola dan `Search()` untuk pencarian pola dalam teks. Variabel NO_OF_CHARS mendefinisikan jumlah karakter unik yang dapat muncul dalam teks dan diinisialisasi dengan nilai 256. Fungsi `max(int a, int b)` adalah fungsi yang mengembalikan nilai maksimum dari dua bilangan bulat.

Fungsi `badCharHeuristic()` bertanggung jawab untuk mengisi array badchar dengan informasi terakhir dari setiap karakter dalam pola. Fungsi ini pertama-tama menginisialisasi seluruh elemen array badchar dengan nilai -1 yang menandakan bahwa karakter tersebut belum muncul dalam pola. Kemudian, fungsi ini mengisi nilai-nilai sebenarnya dari kemunculan terakhir setiap karakter dalam pola dengan mengiterasi melalui pola dan menyimpan indeks kemunculan terakhir dari setiap karakter dalam array badchar.

Fungsi `Search()` adalah fungsi utama yang melakukan pencarian pola dalam teks menggunakan array badchar yang telah diproses. Variabel `m` menyimpan panjang pola, dan `n` menyimpan panjang teks. Array badchar diinisialisasi dan diisi dengan memanggil fungsi `badCharHeuristic`. Proses pencarian dimulai dengan variabel `s` yang digunakan untuk menyimpan pergeseran pola relatif terhadap teks dan diinisialisasi dengan nol. Sebuah loop dijalankan selama nilai `s` kurang dari atau sama dengan `n - m` yang berarti pola masih bisa dibandingkan dengan sisa teks. Variabel `j` diinisialisasi dengan `m - 1` yang merupakan indeks karakter terakhir dalam pola.

Loop dalam pencarian pola mengurangi nilai `j` selama karakter dalam pola dan teks cocok pada pergeseran saat ini `s`. Jika `j` menjadi kurang dari nol, berarti seluruh pola cocok dengan teks pada pergeseran saat ini `s` dan fungsi mengembalikan true. Jika ada karakter yang tidak cocok, pola digeser sehingga karakter buruk dalam teks sejajar dengan kemunculan terakhirnya dalam pola menggunakan nilai dari array badchar. Fungsi `max(1, j - badchar[txt[s + j]])` memastikan pergeseran positif sehingga mencegah pola bergeser mundur.

Jika pola tidak ditemukan setelah seluruh iterasi, fungsi mengembalikan false. Dengan menggunakan heuristik karakter buruk, algoritma Boyer-Moore mampu mempercepat pencarian pola dalam teks dengan menghindari perbandingan yang tidak perlu. Algoritma ini melakukan

pencarian lebih efisien dibandingkan metode pencocokan pola sederhana dengan memanfaatkan informasi kemunculan terakhir dari karakter dalam pola.

3.4 Proses Penyelesaian Masalah Dengan Algoritma Levenshtein Distance

Implementasi algoritma Levenshtein Distance dalam kode ini bertujuan untuk menghitung jarak edit minimum antara dua ASCII sidik jari. Jarak edit ini mengukur seberapa banyak operasi edit (penyisipan, penghapusan, atau penggantian karakter) yang diperlukan untuk mengubah satu string menjadi string lainnya. Kode ini terdiri dari dua fungsi utama, yaitu `Compute()` untuk menghitung jarak Levenshtein dan `Similarity()` untuk menghitung tingkat kemiripan antara dua string berdasarkan jarak Levenshtein.

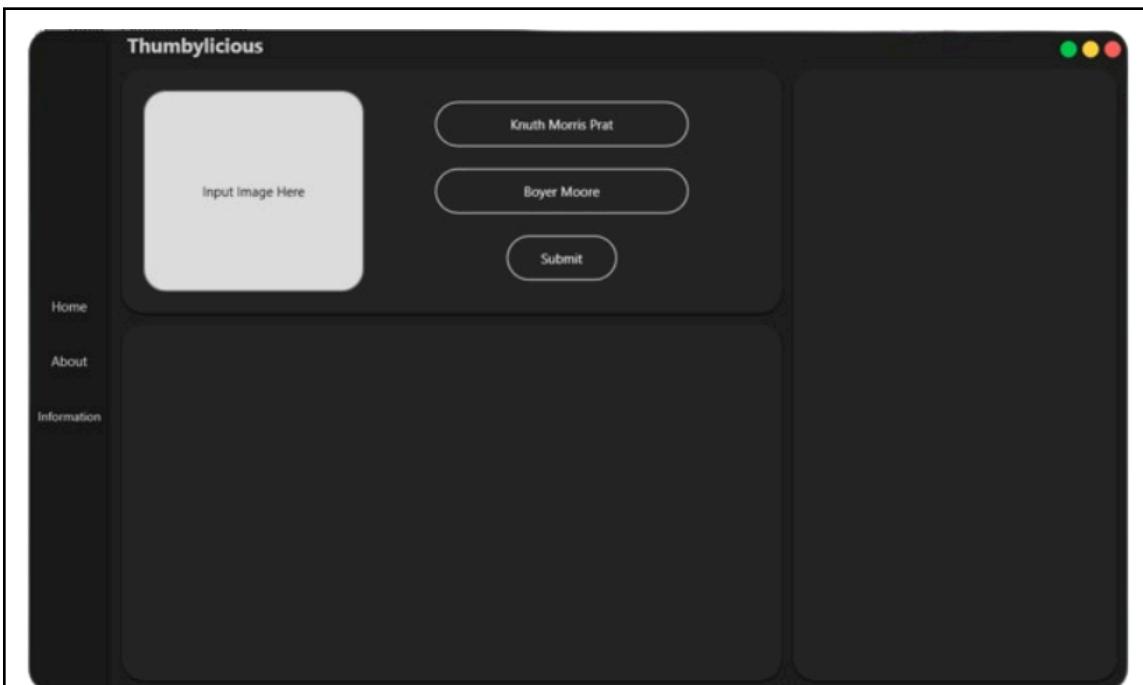
Fungsi `Compute()` menerima dua string sebagai input yaitu str1 dan str2. Pertama, sebuah matriks dua dimensi distance diinisialisasi dengan ukuran $[str1.Length + 1, str2.Length + 1]$. Matriks ini akan digunakan untuk menyimpan jarak edit pada setiap titik perbandingan antara karakter-karakter dalam kedua string. Langkah awal dalam fungsi ini adalah menginisialisasi baris dan kolom pertama dari matriks. Baris pertama diisi dengan nilai yang diinkrement dari 0 hingga $str1.Length$, dan kolom pertama diisi dengan nilai yang diinkrement dari 0 hingga $str2.Length$. Inisialisasi ini menandakan bahwa jika salah satu string kosong, jarak editnya sama dengan panjang string yang lain karena seluruh karakter harus disisipkan atau dihapus.

Setelah inisialisasi, algoritma mengisi sisa matriks dengan nilai jarak edit minimum. Proses ini dilakukan dengan dua loop bersarang yang berjalan melalui setiap karakter dalam kedua string. Pada setiap posisi $[i, j]$ dalam matriks, algoritma menghitung biaya penggantian karakter (cost) yang bernilai 0 jika karakter pada posisi tersebut sama dan bernilai 1 jika berbeda. Nilai jarak edit pada posisi $[i, j]$ kemudian dihitung sebagai nilai minimum dari tiga kemungkinan operasi yaitu penghapusan karakter dari str1 ($distance[i-1, j] + 1$), penyisipan karakter ke str1 ($distance[i, j-1] + 1$), dan penggantian karakter pada str1 ($distance[i-1, j-1] + cost$).

Setelah seluruh matriks terisi, nilai pada sel terakhir di sudut kanan bawah $distance[str1.Length, str2.Length]$ merupakan jarak Levenshtein antara kedua string. Fungsi ini kemudian mengembalikan nilai tersebut.

Fungsi `Similarity()` menggunakan hasil dari fungsi `Compute()` untuk menghitung tingkat kemiripan antara dua string. Fungsi ini pertama-tama memanggil `Compute()` untuk mendapatkan jarak Levenshtein antara kedua string. Kemudian, fungsi ini menghitung panjang maksimum dari kedua string dengan menggunakan `Math.Max`. Tingkat kemiripan dihitung sebagai 1.0 dikurangi rasio antara jarak Levenshtein dan panjang maksimum dari kedua string. Hasilnya adalah nilai antara 0 dan 1, di mana 1 menunjukkan bahwa kedua string identik dan 0 menunjukkan bahwa tidak ada kesamaan antara kedua string.

3.5 Fitur Fungsional dan Arsitektur Aplikasi Desktop



Pada Home page, user bisa menginput gambar sidik jari yang ingin mereka identifikasi, input gambar bisa diklik dan memilih file secara manual dan juga user bisa menginput image dengan cara drag and drop, user juga bisa memilih algoritma dengan button yang sudah disediakan

The screenshot shows a user interface for a fingerprint identification application. On the left, there is a vertical navigation bar with links for Home, About, and Information. The main area is divided into two sections: 'Thumbylicious' on the left and 'KMP' on the right. Both sections show a large fingerprint image and a list of results.

Thumbylicious Results:

- RESULT 1:
 - Knuth Morris Pratt
 - Boyer Moore
- RESULT 2:
 - NIK: 373825
 - Nama: Christopher Scott
 - Tempat Lahir: Surabaya
 - Tanggal Lahir: 8/13/1959 12:00:00 AM
 - Jenis Kelamin: Laki-Laki
 - Golongan Darah: O
 - Alamat: Jl. lagi sama mantan
 - Agama: Katolik
 - Status Perkawinan: Cerai
 - Pekerjaan: Programmer
 - Kewarganegaraan: Amerika
- Similarity: 91.31693198263386 %
- RESULT 3:

KMP Results:

- NIK: 52326502
- Nama: Edward Johnson
- Tempat Lahir: Jogja
- Tanggal Lahir: 4/14/1974 12:00:00 AM
- Jenis Kelamin: Perempuan
- Golongan Darah: A
- Alamat: Jl. in aja dulu
- Agama: Konghucu
- Status Perkawinan: Cerai
- Pekerjaan: Polisi
- Kewarganegaraan: Amerika

Similarity: 100 %
Execution Time: 2105.1747 ms

Pada Home page, user bisa melihat data terakurat di sebelah kanan layar dan top 5 data dengan akurasi terurut menurun, user juga dapat meng-cancel pemakaian sidik jari yang mereka ingin identifikasi

The screenshot shows a dark-themed mobile application window titled "Thumbylicious". On the left is a vertical navigation bar with three items: "Home", "About", and "Information". The main content area has a title "About Thumbylicious" and a descriptive paragraph: "Thumbylicious dirancang untuk mengidentifikasi individu berdasarkan sidik jari menggunakan teknologi biometrik. Dengan memanfaatkan algoritma pattern matching seperti Boyer-Moore dan Knuth-Morris-Pratt, serta metode perhitungan kemiripan Levenshtein Distance, aplikasi ini dapat melakukan pencocokan sidik jari dengan cepat dan akurat."

Pada About page, user bisa melihat penjelasan singkat mengenai aplikasi Thumbylicious ini

The screenshot shows a dark-themed mobile application window titled "Thumbylicious". On the left is a vertical navigation bar with three items: "Home", "About", and "Information". The main content area features a large title "How to Use" and four numbered steps: 1. Upload gambar sidik jari dengan ekstensi .BMP; 2. Pilih algoritma BM atau KMP untuk mencocokkan sidik jari; 3. Klik tombol "Submit" untuk memulai pencocokan sidik jari; 4. Biodata dengan similaritas tertinggi akan ditampilkan di sebelah kanan layar. Sebanyak 4 biodata lainnya dengan similaritas terurut akan ditampilkan di bagian bawah layar.

Pada Information page user bisa memperoleh informasi mengenai cara pemakaian aplikasi

3.6 Contoh Ilustrasi Kasus

Contoh ilustrasi kasus penggunaan aplikasi Thumbylicious adalah berikut:

- Upload gambar sidik jadi dengan ekstensi .BMP
- Pilih algoritma BM atau KMP untuk mencocokkan sidik jari
- Klik button Submit
- Biodata dengan similaritas tertinggi akan ditampilkan di sebelah kanan layar. Sebanyak 4 biodata lainnya dengan similaritas terurut akan ditampilkan di bagian bawah layar

BAB 4

IMPLEMENTASI DAN PENGUJIAN

4.1 Spesifikasi Teknis Program

4.1.1 Struktur Data

- a) Tabel dalam basis data

Basis data terdiri dari dua tabel, biodata dan sidik_jari

Tables_in_tubes3	
	biodata
	sidik_jari
2 rows in set (0.040 sec)	

- b) Deskripsi tabel Biodata

Nama yang berada di tabel ini adalah nama korup dengan metode besar-kecil, angka, dan penyingkatan.

Field	Type	Null	Key	Default	Extra
NIK	varchar(16)	NO	PRI	NULL	
nama	varchar(100)	YES		NULL	
tempat_lahir	varchar(50)	YES		NULL	
tanggal_lahir	date	YES		NULL	
jenis_kelamin	enum('Laki-Laki','Perempuan')	YES		NULL	
golongan_darah	varchar(5)	YES		NULL	
alamat	varchar(255)	YES		NULL	
agama	varchar(50)	YES		NULL	
status_perkawinan	enum('Belum Menikah','Menikah','Cerai')	YES		NULL	
pekerjaan	varchar(100)	YES		NULL	
kewarganegaraan	varchar(50)	YES		NULL	

- c) Deskripsi tabel Sidik_Jari

Berkas_citra berisi *path* menuju gambar sidik jari

Nama berisi nama pemilik sidik jari

Field	Type	Null	Key	Default	Extra
berkas_citra	text	YES		NULL	
nama	varchar(100)	YES		NULL	

2 rows in set (0.046 sec)

4.1.2 Fungsi & Prosedur

```
public class Regex
```

```
public class Regex
{
    private static readonly Dictionary<string, string>
regexPatterns = new Dictionary<string, string>
    {
        {"A", "[Aa4]?"}, {"B", "[Bb]"}, {"C", "[Cc]"}, {"D",
"[Dd]"}, {"E", "[Ee3]?"},
        {"F", "[Ff]"}, {"G", "[Gg6]"}, {"H", "[Hh]"}, {"I",
"[Ii1]?"}, {"J", "[Jj]"},
        {"K", "[Kk]"}, {"L", "[Ll]"}, {"M", "[Mm]"}, {"N",
"[Nn]"}, {"O", "[Oo0]?"},
        {"P", "[Pp]"}, {"Q", "[Qq]"}, {"R", "[Rr]"}, {"S",
"[Ss5]"}, {"T", "[Tt]"},
        {"U", "[Uu]?"}, {"V", "[Vv]"}, {"W", "[Ww]"}, {"X",
"[Xx]"}, {"Y", "[Yy]"},
        {"Z", "[Zz2]"}, {" ", "[ ]"}
    };

    public string Alter(string text)
    {
        text = text.ToUpper();
        foreach (var kvp in regexPatterns)
        {
            text = text.Replace(kvp.Key, kvp.Value);
        }
        return text;
    }
}
```

public class LevenshteinDistance

```
● ● ●

public class LevenshteinDistance
{
    public static int Compute(string str1, string str2)
    {
        int[,] distance = new int[str1.Length + 1, str2.Length +
1];

        // Initialize the first row and column of the matrix
        for (int i = 0; i <= str1.Length; i++)
        {
            distance[i, 0] = i;
        }

        for (int j = 0; j <= str2.Length; j++)
        {
            distance[0, j] = j;
        }

        // Fill in the matrix with minimum edit distances
        for (int i = 1; i <= str1.Length; i++)
        {
            for (int j = 1; j <= str2.Length; j++)
            {
                int cost = (str1[i - 1] == str2[j - 1]) ? 0 : 1;
                distance[i, j] = Math.Min(
                    Math.Min(distance[i - 1, j] + 1,           // deletion
                            distance[i, j - 1] + 1),          // insertion
                    distance[i - 1, j - 1] + cost);      // substitution
            }
        }

        // The bottom-right cell contains the Levenshtein distance
        return distance[str1.Length, str2.Length];
    }

    public static double Similarity(string str1, string str2)
    {
        int distance = Compute(str1, str2);
        int maxLength = Math.Max(str1.Length, str2.Length);
        return 1.0 - (double)distance / maxLength;
    }
}
```

Boyer-Moorepublic static bool Search(char[] txt, char[] pat)

```
● ● ●

// A pattern searching function that uses Bad Character
Heuristic of Boyer Moore Algorithm
public static bool Search(char[] txt, char[] pat)
{
    int m = pat.Length;
    int n = txt.Length;

    int[] badchar = new int[NO_OF_CHARS];

    // Fill the bad character array by calling the
    // preprocessing function badCharHeuristic() for given pattern
    badCharHeuristic(pat, m, badchar);

    int s = 0;

    while (s <= (n - m))
    {
        int j = m - 1;

        // Keep reducing index j of pattern while characters
        // of pattern and text are matching at this shift s
        while (j >= 0 && pat[j] == txt[s + j])
            j--;

        // If the pattern is present at current shift, then
        // index j will become -1 after the above loop
        if (j < 0)
        {
            return true;
        }

        else
        {
            s += max(1, j - badchar[txt[s + j]]);
        }
    }
    return false;
}
```

```
public static bool badCharHeuristic(char[] str, int size, int[] badchar)
```

```
● ● ●  
static int NO_OF_CHARS = 256;  
  
// A utility function to get maximum of two integers  
static int max(int a, int b) { return (a > b) ? a : b; }  
  
// The preprocessing function for Boyer Moore's bad character  
heuristic  
public static bool badCharHeuristic(char[] str, int size,  
int[] badchar)  
{  
    int i;  
  
    // Initialize all occurrences as -1  
    for (i = 0; i < NO_OF_CHARS; i++)  
        badchar[i] = -1;  
  
    for (i = 0; i < size; i++)  
        badchar[(int)str[i]] = i;  
  
    return true;  
}
```

```
private static void computeLPSArray(string pat, int M, int[] lps)
```

```
public static bool KMPSearch(string pat, string txt)
```

```
● ● ●

public static bool KMPSearch(string pat, string txt)
{
    int M = pat.Length;
    int N = txt.Length;

    // Create lps[] that will hold the longest prefix suffix
    // values for pattern
    int[] lps = new int[M];

    // Index for pat[]
    int j = 0;

    // Preprocess the pattern (calculate lps[] array)
    computeLPSArray(pat, M, lps);

    int i = 0;
    while ((N - i) >= (M - j))
    {
        if (pat[j] == txt[i])
        {
            j++;
            i++;
        }
        if (j == M)
        {
            return true;
        }

        else if (i < N && pat[j] != txt[i])
        {
            if (j != 0)
                j = lps[j - 1];
            else
                i = i + 1;
        }
    }
    return false;
}
```

public string ConvertImageToAscii(Image image, int Width, int Height)

```
● ● ●

public string ConvertImageToAscii(Image image, int Width, int Height)
{
    using (Bitmap resizedImage = new Bitmap(image, Width, Height))
    {
        using (MemoryStream stream = new MemoryStream())
        {
            // Menyimpan gambar yang telah diubah ukurannya sebagai JPEG
            resizedImage.Save(stream, ImageFormat.Jpeg);
            byte[] byteArray = stream.ToArray();

            // Mengonversi array byte menjadi string ASCII
            StringBuilder asciiString = new StringBuilder();
            foreach (byte b in byteArray)
            {
                string byteString = Convert.ToString(b, 2).PadLeft(8, '0');
                for (int i = 0; i < byteString.Length; i += 8)
                {
                    if (i + 8 <= byteString.Length)
                    {
                        string bits = byteString.Substring(i, 8);
                        int asciiValue = Convert.ToInt32(bits, 2);
                        char asciiChar = (char)asciiValue;
                        asciiString.Append(asciiChar);
                    }
                }
            }
            return asciiString.ToString();
        }
    }
}
```

public List<Tuple<DataTable, double>> Match(int totalPixels, string filePath, string algorithm)

```

public List<Tuple<DataTable, double>> Match(int totalPixels, string filePath,
string algorithm)
{
    List<Tuple<DataTable, double>> results = new List<Tuple<DataTable, double>>();

    (int Width, int Height) = GetDimensionsForPixels(totalPixels);

    var parser = new Parser();

    Image input = Image.FromFile(filePath);
    string ascii = parser.ConvertImageToAscii(input, Width, Height);

    Dictionary<string, string> fingerprintsDatabase =
DatabaseManager.FetchFingerprintsFromDatabase(Width, Height);

    bool exist = false;
    if (algorithm == "BM")
    {
        foreach (var fingerprint in fingerprintsDatabase)
        {
            bool isMatchBM = BM.Search(ascii.ToCharArray(),
fingerprint.Value.ToCharArray());
            if (isMatchBM)
            {
                Console.WriteLine($"Exact match found with {fingerprint.Key} with
similarity 100%.");
                exist = true;
                break;
            }
        }
    }
    else if (algorithm == "KMP")
    {
        foreach (var fingerprint in fingerprintsDatabase)
        {
            bool isMatchKMP = KMP.KMPSearch(ascii, fingerprint.Value);
            if (isMatchKMP)
            {
                Console.WriteLine($"Exact match found with {fingerprint.Key} with
similarity 100%.");
                exist = true;
                break;
            }
        }
    }

    if (!exist)
    {
        Console.WriteLine("No match fingerprint found");
    }

    // Find similar matches (using Levenshtein Distance, threshold: 80%)
    Console.WriteLine("\nLevenshtein Distance:");
    List<KeyValuePair<string, double>> similarFingerprints = new
List<KeyValuePair<string, double>>();

    foreach (var fingerprint in fingerprintsDatabase)
    {
        double similarity = LevenshteinDistance.Similarity(ascii,
fingerprint.Value);
        similarFingerprints.Add(new KeyValuePair<string, double>(fingerprint.Key,
similarity));
    }

    similarFingerprints.Sort((x, y) => y.Value.CompareTo(x.Value));

    Regex regex = new Regex();
    for (int i = 0; i < Math.Min(5, similarFingerprints.Count); i++)
    {
        // Console.WriteLine($"Text: {similarFingerprints[i].Key}");
        string alteredText = regex.Alter(similarFingerprints[i].Key);
        // Console.WriteLine($"Altered: {alteredText}");
    }
}

```

```
public static DataTable showBiodata(string nama)
{
    using (MySqlConnection connection = new
MySqlConnection(connectionString))
    {
        string query = "SELECT * FROM biodata WHERE nama REGEXP
@nama";
        MySqlCommand cmd = new MySqlCommand(query, connection);
        cmd.Parameters.AddWithValue("@nama", nama);

        DataTable biodataTable = new DataTable();
        try
        {
            connection.Open();
            MySqlDataReader reader = cmd.ExecuteReader();
            biodataTable.Load(reader);
        }
        catch (Exception ex)
        {
            Console.WriteLine("Error: " + ex.Message);
        }
        return biodataTable;
    }
}
```

public static string GetImage(string nama)

```
public static string GetImage(string nama)
{
    string imagePath = null;

    using (MySqlConnection connection = new
MySqlConnection(connectionString))
    {
        string query = "SELECT berkas_citra FROM sidik_jari WHERE
nama = @nama";
        MySqlCommand cmd = new MySqlCommand(query, connection);
        cmd.Parameters.AddWithValue("@nama", nama);

        try
        {
            connection.Open();
            MySqlDataReader reader = cmd.ExecuteReader();

            if (reader.Read())
            {
                imagePath = reader["berkas_citra"].ToString();
            }

            reader.Close();
        }
        catch (Exception ex)
        {
            Console.WriteLine("Error: " + ex.Message);
        }
    }
    return imagePath;
}
```

public string GetOutputImage(List<Tuple<DataTable, double>> results)

```
● ● ●

public string GetOutputImage(List<Tuple<DataTable, double>>
    results)
{
    if (results == null || results.Count == 0)
    {
        return "GetOutputImage error";
    }

    DataTable firstTable = results[0].Item1;
    if (firstTable.Rows.Count == 0)
    {
        return "GetOutputImage error";
    }

    string nama = firstTable.Rows[0]["nama"].ToString();
    return GetImage(nama);
}
```

```
// Insert nama dan ascii ke table sidik_jari
public static void Insert(string ascii, string name)
{
    using (MySqlConnection connection = new MySqlConnection(connectionString))
    {
        try
        {
            connection.Open();
            Console.WriteLine("Connection to database established successfully.");

            // Query to insert fingerprint data
            string query = "INSERT INTO sidik_jari (berkas_citra, nama) VALUES (@berkas_citra,
@nama)";
            MySqlCommand command = new MySqlCommand(query, connection);

            // Adding parameters to prevent SQL injection
            command.Parameters.AddWithValue("@berkas_citra", ascii);
            command.Parameters.AddWithValue("@nama", name);

            Console.WriteLine("Inserting fingerprint data...");

            // Execute the command
            int rowsAffected = command.ExecuteNonQuery();

            Console.WriteLine($"{rowsAffected} row(s) inserted.");
        }
        catch (MySqlException mysqlEx)
        {
            Console.WriteLine($"MySQL error: {mysqlEx.Message}");
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred Insert: {ex.Message}");
        }
    }
}
```

```
// Fetch data sidik_jari dari database ke dictionary
public static Dictionary<string, string> FetchFingerprintsFromDatabase(int width, int height)
{
    Dictionary<string, string> fingerprintsDatabase = new Dictionary<string, string>();

    using (MySqlConnection connection = new MySqlConnection(connectionString))
    {
        try
        {
            connection.Open();

            string query = "SELECT nama, berkas_citra FROM sidik_jari";
            MySqlCommand command = new MySqlCommand(query, connection);
            var parser = new PatternMatching.Parser();

            using (MySqlDataReader reader = command.ExecuteReader())
            {
                while (reader.Read())
                {
                    string nama = reader["nama"].ToString();
                    string berkas_citra = reader["berkas_citra"].ToString();

                    Image image = Image.FromFile(berkas_citra);
                    string ascii = parser.ConvertImageToAscii(image, width, height);

                    fingerprintsDatabase.Add(nama, ascii);
                }
            }
        }
        catch (MySqlException ex)
        {
            Console.WriteLine($"MySQL error: {ex.Message}");
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred FetchFingerprintsFromDatabase: {ex.Message}");
        }
    }

    return fingerprintsDatabase;
}
```

4.2 Dokumentasi Pengguna

4.3 Hasil pengujian

Input	Algoritma	Hasil
	Knuth-Morris-Path (KMP)	<p>KMP</p>  <p> NIK: 36173355 Nama: Brian Allen Tempat Lahir: Jakarta Tanggal Lahir: 6/15/1997 12:00:00 AM Jenis Kelamin: Perempuan Golongan Darah: AB Alamat: Jl. in aja dulu Agama: Konghucu Status Perkawinan: Cerai Pekerjaan: Wirausahawan Kewarganegaraan: Indonesia </p> <p> Similarity: 100 % Execution Time: 1994.1427 ms </p>
	Boyer-Moore (BM)	<p>BM</p>  <p> NIK: 36173355 Nama: Brian Allen Tempat Lahir: Jakarta Tanggal Lahir: 6/15/1997 12:00:00 AM Jenis Kelamin: Perempuan Golongan Darah: AB Alamat: Jl. in aja dulu Agama: Konghucu Status Perkawinan: Cerai Pekerjaan: Wirausahawan Kewarganegaraan: Indonesia </p> <p> Similarity: 100 % Execution Time: 1835.8727 ms </p>

	<p>Knuth-Morris-Path (KMP)</p>	<p>KMP</p>  <p>NIK: 29709757 Nama: Thomas Mitchell Tempat Lahir: Bali Tanggal Lahir: 8/6/1983 12:00:00 AM Jenis Kelamin: Laki-Laki Golongan Darah: A Alamat: Jl. i saja hidup ini Agama: Katolik Status Perkawinan: Belum Menikah Pekerjaan: Wirausahawan Kewarganegaraan: Jepang</p> <p>Similarity: 100 % Execution Time: 1862.2514 ms</p>
	<p>Boyer-Moore (BM)</p>	<p>BM</p>  <p>NIK: 29709757 Nama: Thomas Mitchell Tempat Lahir: Bali Tanggal Lahir: 8/6/1983 12:00:00 AM Jenis Kelamin: Laki-Laki Golongan Darah: A Alamat: Jl. i saja hidup ini Agama: Katolik Status Perkawinan: Belum Menikah Pekerjaan: Wirausahawan Kewarganegaraan: Jepang</p> <p>Similarity: 100 % Execution Time: 2246.4646 ms</p>

	<p>Knuth-Morris-Path (KMP)</p>	<p>KMP</p>  <p>NIK: 99918051 Nama: Joshua Brown Tempat Lahir: Bandung Tanggal Lahir: 2/14/1995 12:00:00 AM Jenis Kelamin: Laki-Laki Golongan Darah: B Alamat: Jl. doang jadian kagak Agama: Protestan Status Perkawinan: Belum Menikah Pekerjaan: Tidak bekerja Kewarganegaraan: Korea</p> <p>Similarity: 100 % Execution Time: 2013.9208 ms</p>
	<p>Boyer-Moore (BM)</p>	<p>BM</p>  <p>NIK: 99918051 Nama: Joshua Brown Tempat Lahir: Bandung Tanggal Lahir: 2/14/1995 12:00:00 AM Jenis Kelamin: Laki-Laki Golongan Darah: B Alamat: Jl. doang jadian kagak Agama: Protestan Status Perkawinan: Belum Menikah Pekerjaan: Tidak bekerja Kewarganegaraan: Korea</p> <p>Similarity: 100 % Execution Time: 2093.1383 ms</p>

	Knuth-Morris-Path (KMP)	<p>KMP</p>  <p>NIK: 74500422 Nama: Ashley King Tempat Lahir: Bali Tanggal Lahir: 6/20/1952 12:00:00 AM Jenis Kelamin: Laki-Laki Golongan Darah: B Alamat: Jl. in aja dulu Agama: Islam Status Perkawinan: Menikah Pekerjaan: Wirausahawan Kewarganegaraan: Amerika</p> <p>Similarity: 100 % Execution Time: 1753.9451 ms</p>
	Boyer-Moore (BM)	<p>BM</p>  <p>NIK: 74500422 Nama: Ashley King Tempat Lahir: Bali Tanggal Lahir: 6/20/1952 12:00:00 AM Jenis Kelamin: Laki-Laki Golongan Darah: B Alamat: Jl. in aja dulu Agama: Islam Status Perkawinan: Menikah Pekerjaan: Wirausahawan Kewarganegaraan: Amerika</p> <p>Similarity: 100 % Execution Time: 1711.1883 ms</p>

	<p>Knuth-Morris-Path (KMP)</p>	<p>KMP</p>  <p>NIK: 47175091 Nama: Anthony Wilson Tempat Lahir: Surabaya Tanggal Lahir: 4/7/1976 12:00:00 AM Jenis Kelamin: Perempuan Golongan Darah: A Alamat: Jl. doang jadian kagak Agama: Konghucu Status Perkawinan: Cerai Pekerjaan: Polisi Kewarganegaraan: Indonesia Similarity: 100 % Execution Time: 1768.4135 ms</p>
	<p>Boyer-Moore (BM)</p>	<p>BM</p>  <p>NIK: 47175091 Nama: Anthony Wilson Tempat Lahir: Surabaya Tanggal Lahir: 4/7/1976 12:00:00 AM Jenis Kelamin: Perempuan Golongan Darah: A Alamat: Jl. doang jadian kagak Agama: Konghucu Status Perkawinan: Cerai Pekerjaan: Polisi Kewarganegaraan: Indonesia Similarity: 100 % Execution Time: 1847.1471 ms</p>

	Knuth-Morris-Path (KMP)	<p>KMP</p> <p>NIK: 72656150 Nama: William Torres Tempat Lahir: Surabaya Tanggal Lahir: 5/13/1997 12:00:00 AM Jenis Kelamin: Perempuan Golongan Darah: AB Alamat: Jl. jalan yuk Agama: Konghucu Status Perkawinan: Belum Menikah Pekerjaan: Mahasiswa Kewarganegaraan: Indonesia</p> <p>Similarity: 100 % Execution Time: 1759.688 ms</p>
	Boyer-Moore (BM)	<p>BM</p> <p>NIK: 72656150 Nama: William Torres Tempat Lahir: Surabaya Tanggal Lahir: 5/13/1997 12:00:00 AM Jenis Kelamin: Perempuan Golongan Darah: AB Alamat: Jl. jalan yuk Agama: Konghucu Status Perkawinan: Belum Menikah Pekerjaan: Mahasiswa Kewarganegaraan: Indonesia</p> <p>Similarity: 100 % Execution Time: 1747.1423 ms</p>

4.4 Analisis Hasil Pengujian

Berdasarkan hasil pengujian yang dilakukan, program dapat mengeksekusi perintah dengan baik dan program berhasil memiliki keakuratan dan efisiensi yang tinggi.

1. Akurasi Pencocokan:

Algoritma Knuth-Morris-Pratt (KMP) dan Boyer-Moore (BM) mampu mencocokkan sidik jari dengan akurasi yang baik. Hal ini disebabkan oleh kemampuan kedua algoritma tersebut dalam menghindari pemeriksaan ulang karakter yang sudah diketahui cocok.

2. Kecepatan Pencocokan:

Kecepatan pencocokan pola sidik jari oleh kedua algoritma ini sangat tinggi. KMP dan BM memproses pencocokan dalam waktu rata-rata yang singkat.

3. Penyimpanan pada Database:

Program mampu menyimpan data data dan memeriksa berdasarkan data yang disimpan pada database dengan sangat baik dan akurat. Ini menandakan database yang dibuat serta sistem Regex berjalan dengan baik.

Time Complexity

KMP memiliki kompleksitas waktu $O(m + n)$, di mana m adalah panjang pola dan n adalah panjang teks. Kompleksitas ini diperoleh karena KMP hanya melakukan dua kali traversing terhadap pola dan teks.

BM memiliki kompleksitas waktu rata-rata $O(n/m)$, dengan m adalah panjang pola dan n adalah panjang teks. Pada kasus terburuk, kompleksitasnya adalah $O(mn)$, tetapi ini jarang terjadi karena algoritma BM sangat efektif dalam skipping karakter yang tidak cocok.

Kekurangan

KMP memiliki kekurangan yakni memerlukan preprocessing untuk membangun tabel LPS (Longest Prefix Suffix), namun overhead ini cukup kecil dibandingkan kecepatan pencocokan.

BM memiliki kekurangan yakni algoritma ini bisa menjadi lambat pada kasus terburuk ketika pola dan teks memiliki banyak kemiripan, meskipun ini jarang terjadi dan hanya kejadian pada kasus tertentu saja.

BAB 5

PENUTUP

5.1 Kesimpulan

Dalam tugas besar ini, kami telah berhasil mengimplementasikan program untuk mendeteksi sidik jari serta menyambungkannya kedalam database yang berbasis biometrik menggunakan citra sidik jari. Sistem ini memanfaatkan algoritma Knuth-Morris-Pratt (KMP) dan Boyer-Moore (BM) untuk melakukan pencocokan pola sidik jari dengan pola yang tersimpan di basis data. Hasil pengujian menunjukkan bahwa sistem dapat mengenali sidik jari dengan tingkat akurasi yang tinggi dan waktu pencocokan yang efisien. Dengan teknologi ini, diharapkan dapat meningkatkan keamanan dan keandalan sistem identifikasi individu di berbagai aplikasi.

5.2 Saran

Terdapat beberapa saran yang dapat diaplikasikan dalam program kami untuk meningkatkan kualitas dan kinerja dari program yang kami buat yang diringkas sebagai berikut :

1. Integrasi dengan Teknologi Lain: Mengintegrasikan aplikasi yang kami buat dengan teknologi untuk mengenali sidik jari sehingga lebih mudah.
2. Optimasi Algoritma: Melakukan optimasi lebih lanjut pada algoritma pencocokan untuk menangani dataset yang lebih besar dan lebih kompleks.
3. Pengembangan Antarmuka Pengguna: Meningkatkan antarmuka pengguna agar lebih intuitif dan user-friendly, sehingga mempermudah penggunaan oleh berbagai kalangan.

5.3 Refleksi

Terdapat banyak hal yang kami pelajari dari proses pengerjaan tugas besar ini. Kami belajar pentingnya kolaborasi tim dan komunikasi yang efektif untuk menyelesaikan tugas yang kompleks. Kami juga memperdalam pengetahuan mengenai algoritma pencocokan pola dan aplikasinya dalam teknologi biometrik. Tantangan yang kami hadapi, seperti optimasi algoritma dan integrasi dengan basis data, memberikan pengalaman yang sangat berguna dalam pengembangan sistem yang efisien dan handal. Proyek ini telah memberikan kami pengetahuan dan keterampilan baru di bidang strategi algoritma dan pengembangan aplikasi biometrik.

LAMPIRAN

LINK REPOSITORY

Link repository GitHub: https://github.com/attaramajesta/Tubes3_Thumbylicious

LINK VIDEO

Link video Youtube: <https://youtu.be/8gE9MXVHqvc>

Daftar Pustaka

1. Knuth, D. E., Morris, J. H., & Pratt, V. R. (1977). Fast pattern matching in strings. SIAM Journal on Computing, 6(2), 323-350.
2. Boyer, R. S., & Moore, J. S. (1977). A fast string searching algorithm. Communications of the ACM, 20(10), 762-772.
3. Rinaldi Munir, diakses dari <https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2022-2023/String-Matching-dengan-Regex-2019.pdf>
4. Rinaldi Munir, Munir, Rinaldi, <https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2022-2023/String-Matching-dengan-Regex-2019.pdf>
5. Navarro, G., & Raffinot, M. (2002). Flexible pattern matching in strings: Practical on-line search algorithms for texts and biological sequences. Cambridge University Press.
6. Anonymus. (2023, May 15). Boyer–Moore string-search algorithm. In Wikipedia, The Free Encyclopedia.
7. Wikipedia - Boyer–Moore String-Search Algorithm