

# **LAPORAN TUGAS BESAR 3 IF2211**

## **STRATEGI ALGORITMA**

Pemanfaatan Pattern Matching dalam Membangun Sistem

Deteksi Individu Berbasis Biometrik Melalui Citra Sidik Jari



### **Kelompok Click :**

- |                                 |          |
|---------------------------------|----------|
| 1. Rici Trisna Putra            | 13522026 |
| 2. Keanu Amadius Gonza Wrahatno | 13522082 |
| 3. Dimas Bagoes Hendrianto      | 13522112 |

**PROGRAM STUDI TEKNIK INFORMATIKA  
SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA  
INSTITUT TEKNOLOGI BANDUNG  
2024**

## **DAFTAR ISI**

DAFTAR ISI	1
BAB 1	2
BAB 2	5
2.1 Pattern Matching	5
2.2 KMP	5
2.3 BM	8
2.4 Regular Expression	10
2.5 Persentase kemiripan	11
2.6 Aplikasi	11
BAB 3	13
3.1 Pemecahan Masalah	13
3.2 Penyelesaian Solusi	14
3.3 Fitur	17
3.4 Ilustrasi Kasus	18
BAB 4	20
4.1 Spesifikasi Teknis	20
4.2 Penggunaan Aplikasi	26
4.3 Hasil Pengujian	28
4.3.1 Pencarian rute menggunakan algoritma BFS dengan single path	28
4.3.2 Pencarian rute menggunakan algoritma BFS dengan multiple path	32
4.3.3 Pencarian rute menggunakan algoritma IDS dengan single path	36
4.3.4 Pencarian rute menggunakan algoritma IDS dengan multiple path	40
4.4 Analisis	45
BAB 5	46
5.1 Kesimpulan	46
5.2 Saran	46
5.3 Refleksi	46
Daftar Pustaka	47
Lampiran	48

## BAB 1

### DESKRIPSI TUGAS



**Gambar 1.1 Ilustrasi fingerprint recognition.**

(Sumber: <https://www.inews.id/techno/elektronik/mengenal-teknologi-biometrik-terbaru-sistem-pemindai-sidik-jari>)

Di era digital ini, keamanan data dan akses menjadi semakin penting. 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.

Namun pada Tugas Besar ini 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.

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. Selanjutnya, data pixel tersebut akan dikonversi menjadi binary lalu 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.

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

(Sumber : Dokumen Asisten)

Setelah dihasilkan serangkaian karakter ASCII 8-bit, dilakukan pencarian sidik jari yang paling mirip dengan sidik jari yang menjadi masukan pengguna dilakukan dengan algoritma pencocokan string Knuth-Morris-Pratt (KMP) dan Boyer-Moore (BM). Jika tidak ada satupun sidik jari pada basis data yang exact match dengan sidik jari, maka gunakan sidik jari paling mirip dengan kesamaan diatas nilai tertentu (threshold) menggunakan Longest Common Subsequence (LCS).

Lalu akan ada biodata yang akan dicari berdasarkan kesamaan sidik jari. Biodata tersebut yaitu NIK, nama, tempat/tanggal lahir, jenis kelamin, golongan darah, alamat, agama, status perkawinan, pekerjaan, dan kewarganegaraan. Relasi ini dibuat dalam sebuah basis data dengan skema detail seperti yang tertera pada bagian di bawah ini.

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

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

**Gambar 1.3 Skema basis data yang digunakan.**

(Sumber : Dokumen Asisten)

Seorang pribadi dapat memiliki lebih dari satu berkas citra sidik jari (relasi one-to-many). Akan tetapi, seperti yang dapat dilihat pada skema relasional di atas, keduanya tidak terhubung dengan sebuah relasi. Hal ini disebabkan karena pada kasus dunia nyata, data yang disimpan bisa saja mengalami korup. Dengan membuat atribut kolom yang mungkin korup adalah atribut nama pada tabel biodata, maka atribut nama pada tabel sidik\_jari tidak dapat memiliki foreign-key yang mereferensi ke tabel biodata. Jenis data korup adalah bahasa alay Indonesia.

Variasi	Hasil
Kata orisinil	Bintang Dwi Marthen
Kombinasi huruf besar-kecil	bintang DwI mArthen
Penggunaan angka	B1nt4n6 Dw1 M4rthen
Penyingkatan	Bntng Dw Mrthen
Kombinasi ketiganya	b1ntN6 Dw mrthn

**Tabel 1.2 Kombinasi ketiga variasi bahasa alay Indonesia**

(Sumber : Dokumen Penulis)

Cara yang dapat digunakan untuk menangani ini adalah dengan menggunakan Regular Expression (Regex). Lalu melakukan konversi pola karakter alay hingga dapat dikembalikan ke bentuk alfabetik yang bersesuaian. Setelah menggunakan Regex, Anda akan diminta kembali untuk melakukan pattern matching antara nama yang bersesuaian dengan algoritma KMP dan BM dengan ketentuan yang sama seperti saat pencocokan sidik jari.

## BAB 2

### LANDASAN TEORI

#### 2.1 *Pattern Matching*

Pattern Matching merupakan teknik yang digunakan untuk mengidentifikasi pola-pola yang ada dalam suatu data untuk menentukan kecocokan. Elemen pada pattern matching yaitu T dan P. Dimana T adalah *text* string yang panjangnya n karakter dan P adalah *pattern* string dengan panjang m karakter yang akan dicari pada *text*.

#### 2.2 *KMP*

Algoritma Knuth-Morris-Pratt (KMP) adalah algoritma pencarian string yang efisien untuk menemukan kemunculan sebuah *pattern* dalam sebuah *text* dengan *left to right order*. Dikembangkan oleh Donald Knuth, Vaughan Pratt, dan James H. Morris, algoritma ini menghindari pencocokan ulang karakter yang sudah dibandingkan sebelumnya. Ini membuat KMP lebih efisien dibandingkan dengan algoritma pencarian string yang lebih sederhana seperti pencarian brute force.

KMP bekerja dalam dua fase utama: fase pra-pemrosesan dan fase pencarian. Dalam fase pra-pemrosesan, KMP membuat sebuah tabel bantuan yang disebut tabel "border function" dengan indeks k. Tabel ini menyimpan informasi tentang panjang proper prefix yang juga merupakan suffix untuk setiap posisi dalam pola. Proper prefix adalah bagian awal dari string yang tidak mencakup keseluruhan string itu sendiri ( $n-1$ ), dan suffix adalah bagian akhir dari string. Tabel ini membantu algoritma untuk mengetahui seberapa jauh harus melompat saat terjadi ketidakcocokan selama fase pencarian, menghindari perbandingan yang tidak perlu.

Selama fase pencarian, KMP menggunakan tabel border function untuk menentukan pergeseran pola saat terjadi ketidakcocokan. Algoritma KMP akan memulai melakukan pencocokan pada karakter pertama di teks dan karakter pertama di pola. Anggap i adalah indeks pada teks dan j adalah indeks pada pola. Jika ada ketidakcocokan antara suatu karakter dalam teks dan karakter dalam pola, algoritma menggunakan informasi dari tabel border function untuk mengubah j ke posisi berikutnya yang paling mungkin mencocokkan tanpa mengulang perbandingan karakter yang sudah dibandingkan sebelumnya.

Ini dilakukan dengan mengubah indeks  $j$  seperti yang ditunjukkan oleh tabel border function untuk posisi ketidakcocokan tersebut. Indeks  $j$  yang baru didapat dari tabel border function dengan indeks  $k$  adalah indeks  $j-1$  saat karakter  $j$  terjadi ketidak cocokan dengan karakter  $i$  pada teks. Dengan cara ini, KMP menghindari pengulangan perbandingan dan memastikan bahwa setiap karakter dalam teks dan pola dibandingkan paling banyak satu kali, sehingga menghasilkan kompleksitas waktu pencarian yang linier,  $O(n + m)$ , di mana  $n$  adalah panjang teks dan  $m$  adalah panjang pola.

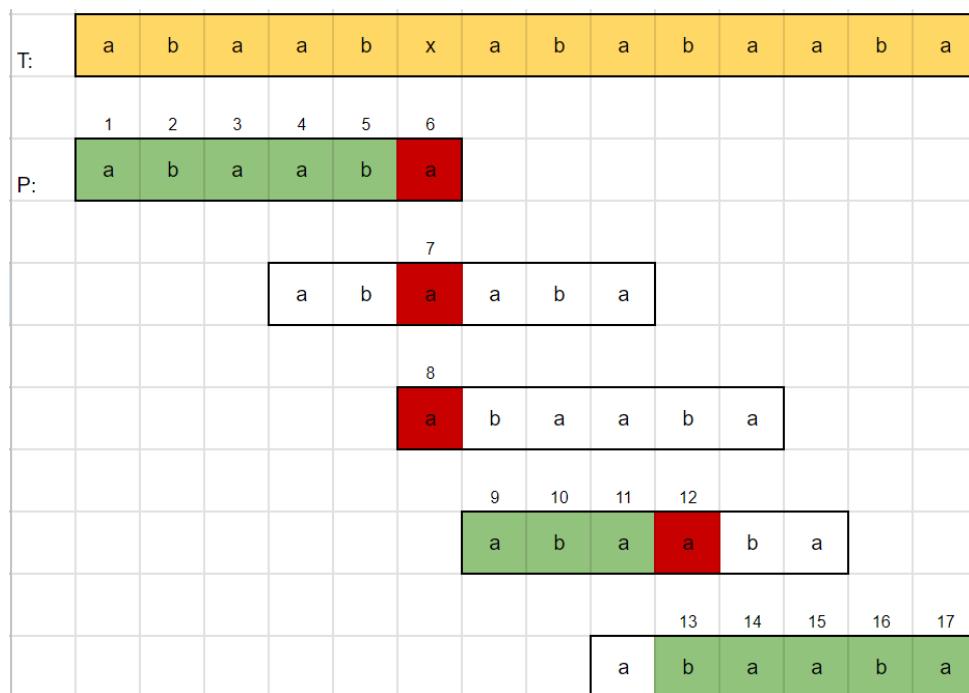
Misalkan ada teks “*abaabxababaaba*” dan pola “*abaaba*”. Diketahui  $i = 0,13$  dan  $j = 0,5$ . Langkah pertama yaitu membuat tabel border function.

- Indeks  $k = 0$ , pola sampai  $j=0$  (“a”) tidak memiliki prefix yang juga sufix
- Indeks  $k = 1$ , pola sampai  $j=1$  (“ab”) tidak memiliki prefix yang juga sufix
- Indeks  $k = 2$ , pola sampai  $j=2$  (“aba”) memiliki prefix dengan panjang 1 yang juga merupakan sufix “a”
- Indeks  $k = 3$ , pola sampai  $j=3$  (“abaa”) memiliki prefix dengan panjang 1 yang juga merupakan sufix yaitu “a”
- Indeks  $k = 4$ , pola sampai  $j=4$  (“abaab”) memiliki prefix dengan panjang 2 yang juga merupakan sufix yaitu “ab”

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

$k$	0	1	2	3	4
$b[k]$	0	0	1	1	2
$i$	0	1	2	3	4

Lalu lakukan langkah pencocokan mulai dari  $i = 0$  dan  $j = 0$ .



- Pada pencocokan ke-6 karakter pada pola tidak sama dengan karakter pada teks sehingga dipanggil fungsi border dengan  $k = j - 1 = 5 - 1 = 4$ . Diketahui  $b[4] = 2$  maka pencocokan ke-7 dilakukan dari indeks  $j = 2$ .
- Karena pada pencocokan ke-7 karakter pola tidak sama dengan karakter teks, maka dipanggil lagi fungsi border dengan  $k = 2 - 1 = 1$ . Diketahui  $b[1] = 0$ . Maka pencocokan ke-8 dimulai dari  $j=0$ .
- Pada pencocokan ke-8, karakter pada pola tidak sama dengan karakter pada teks. Karena indeks  $j = 0$  maka lakukan pergeseran 1 kali ke kanan.
- Pada pencocokan ke-12, karakter pada pola tidak sama dengan karakter pada teks sehingga dipanggil fungsi border dengan  $k = 3-1 = 2$ . Karena  $b[2] = 1$ , maka pencocokan ke-13 dimulai dari indeks  $j = 1$ .
- Pada pencocokan ke-17 merupakan karakter akhir dari pola dan karakter tersebut cocok dengan karakter teks. Sehingga algoritma berhenti dan menghasilkan true.

Keunggulan utama dari algoritma KMP adalah efisiensinya karena algoritma ini tidak perlu mudur dalam mengecek karakter di teks. Algoritma ini efisien dalam pencarian string di berbagai aplikasi seperti pencarian teks dalam dokumen, analisis DNA dalam bioinformatika, dan deteksi pola dalam data besar. KMP sangat efisien karena waktu komputasi tidak bergantung

pada pola berulang atau kemunculan karakter dalam teks, tetapi pada panjang pola dan teks itu sendiri. Hal ini membuat KMP sangat efektif dalam situasi di mana teks dan pola memiliki panjang yang besar atau ketika pola sering berubah, memungkinkan pencarian yang cepat dan responsif tanpa menghabiskan sumber daya komputasi yang berlebihan.

### 2.3 BM

Algoritma Boyer-Moore (BM) adalah salah satu algoritma pencarian string paling efisien yang dikembangkan oleh Robert S. Boyer dan J Strother Moore pada tahun 1977. Algoritma ini memiliki kemampuan untuk melompati sejumlah besar karakter dalam teks selama pencarian, sehingga mempercepat proses pencocokan pola. Boyer-Moore melakukan pencarian dari kanan ke kiri pola sehingga memungkinkan untuk lompatan yang lebih besar saat terjadi ketidakcocokan.

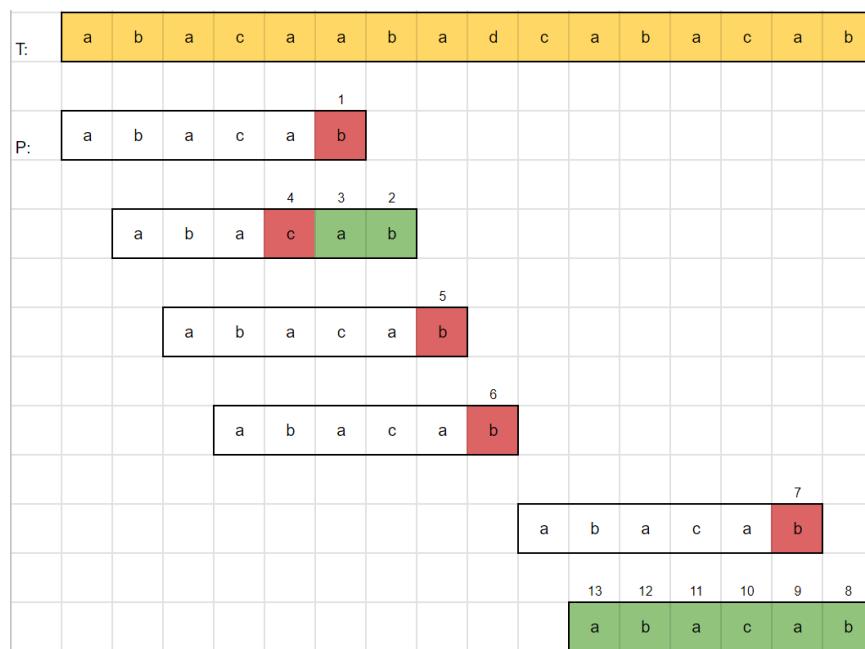
Algoritma Boyer-Moore menggunakan 2 teknik. Teknik yang pertama yaitu looking-glass teknik dimana teknik ini akan mencari P di T dengan bergerak secara mundur (backwards) di P dimulai dari karakter terakhirnya.

Teknik kedua yaitu character-jump dimana teknik ini akan mengecek karakter di teks dan pola. Apabila ada ketidaksamaan, maka akan ada 3 kemungkinan. Misalkan karakter di  $T[i]$  adalah x.

- Kemungkinan pertama saat P mengandung x, maka geser P ke kanan sehingga kemunculan terakhir x di P sejajar dengan  $T[i]$ . Lalu mulai pencocokan dari kanan ke kiri P. Pergeseran bisa dilakukan dengan melihat tabel last occurrence function  $L()$ .
- Kemungkinan kedua saat P mengandung x, namun tidak memungkinkan menggeser P ke kanan untuk menyajarkan kemunculan terakhir x di P dengan  $T[i]$ . Maka geser P ke kanan sebanyak 1 dan mulai pencocokan dari kanan ke kiri P.
- Kemungkinan ke 3 yaitu saat P tidak mengandung x sehingga geser P sehingga P[0] sejajar dengan  $T[i+1]$ .

Untuk menggeser pada kasus 1, lakukan dengan membuat tabel last occurrence function terlebih dahulu. Tabel last occurrence function atau  $L(x)$  dibuat dengan mendefinisikan index i terbesar dimana  $P[i] == x$  atau -1 jika karakter tidak ada di dalam pola.

x	a	b	c	d
L(x)	4	5	3	-1



- Pada pencocokan ke-1 merupakan kasus ke 1 dengan melihat tabel  $L(a)$  maka j akan digeser ke index 4.
- Pada pencocokan ke-4 merupakan kasus ke 2 karena  $L(a)$  ada di index 4 sedangkan P tidak bisa digeser ke kiri sehingga P digeser kekanan sebesar 1.
- Pada pencocokan ke-5 merupakan hal yang sama dengan pencocokan ke-1
- Pada pencocokan ke-6 merupakan kasus ke 3 karena  $L(d) = -1$  yang berarti tidak ada karakter di di P sehingga P digeser samai  $P[0] = T[i+1]$
- Pada pencocokan ke-7 merupakan kasus yang sama dengan pencocokan ke-1

Kompleksitas waktu terburuk dari Boyer-Moore adalah  $O(nm)$ , di mana n adalah panjang teks dan m adalah panjang pola, tetapi pada kasus rata-rata, algoritma ini sangat cepat dengan kompleksitas waktu mendekati  $O(n/m)$ , membuatnya ideal untuk pencarian string dalam teks besar atau dalam aplikasi yang membutuhkan pencarian cepat dan efisien.

## 2.4 Regular Expression

Regular expressions (regex atau regexp) adalah suatu cara untuk mendeskripsikan pola pencarian dan manipulasi teks yang sangat kuat dan fleksibel. Regex menggunakan sekumpulan simbol dan sintaks khusus untuk mendefinisikan pola yang dapat dicocokkan, ditemukan, dan dimanipulasi dalam string teks. Dengan regex, Anda dapat melakukan berbagai tugas pemrosesan teks seperti validasi format, pencarian kata atau frasa, penggantian substring, dan ekstraksi informasi dari teks.

Regex memiliki banyak simbol dan konstruk yang memungkinkan deskripsi pola yang sangat spesifik. Misalnya, simbol seperti `'\d'` mencocokkan digit (angka 0-9), `'\w'` mencocokkan karakter alfanumerik (huruf dan angka), dan `'\s'` mencocokkan spasi atau karakter whitespace lainnya. Selain itu, karakter seperti `.` mencocokkan sembarang karakter kecuali newline, dan `'\*', `'+', dan `'?` digunakan untuk menunjukkan jumlah pengulangan. Kurung siku `[]` digunakan untuk membuat karakter kelas, seperti `'[a-z]'` yang mencocokkan huruf kecil dari a hingga z.

Regex juga mendukung penggunaan grup dan back-references, yang memungkinkan bagian dari pola untuk ditangkap dan digunakan kembali. Misalnya, pola `(abc)\1` akan mencocokkan "abcabc" karena `'\1'` mengacu pada grup tangkapan pertama, yaitu "abc". Selain itu, regex mendukung berbagai operasi lanjutan seperti lookahead dan lookbehind assertions, yang memungkinkan Anda untuk mencocokkan teks berdasarkan konteks di sekitarnya tanpa benar-benar menyertakan konteks tersebut dalam hasil pencocokan.

Regex digunakan di berbagai bahasa pemrograman dan alat teks, seperti Python, Perl, Java, JavaScript, dan editor teks seperti Vim dan sed. Keberadaan regex dalam alat-alat ini mempermudah tugas pemrosesan teks yang kompleks menjadi lebih sederhana dan dapat diotomatisasi. Misalnya, dalam pengembangan web, regex sering digunakan untuk memvalidasi input pengguna seperti alamat email, nomor telepon, dan kode pos. Di bidang ilmu data, regex dapat digunakan untuk mengekstrak data penting dari teks tidak terstruktur. Kemampuan regex

untuk mendeskripsikan pola pencarian teks secara singkat dan efisien menjadikannya alat yang sangat berharga dalam pemrograman dan pengolahan teks.

## ***2.5 Persentase kemiripan***

Kemiripan adalah suatu kondisi atau keadaan di mana dua atau lebih objek, entitas, atau fenomena memiliki kesamaan dalam satu atau lebih aspek atau karakteristik. Kemiripan mengukur sejauh mana objek-objek tersebut berbagi atribut yang sama, baik itu bentuk, struktur, fungsi, atau sifat lainnya. Dalam konteks tugas besar kali ini, kemiripan yang dimaksud adalah kemiripan antara dua gambar sidik jari, yaitu sidik jari dari masukan pengguna dengan sidik jari yang ada di basis data. Ada beberapa pendekatan menggunakan banyak algoritma dalam pencocokan, kami memilih LCS dalam mencari kemiripan gambar.

Longest Common Subsequence (LCS) adalah algoritma yang bertujuan untuk menemukan subsekuens terpanjang yang dapat ditemukan dalam dua string tanpa mengubah urutan karakter. Berbeda dengan substring, elemen-elemen dalam subsekuens tidak harus bersebelahan dalam string asli tetapi harus muncul dalam urutan yang sama.

Algoritma LCS bekerja dengan menggunakan pendekatan pemrograman dinamis. Pertama, sebuah matriks dua dimensi dibangun untuk menyimpan hasil perhitungan panjang subsekuens terpanjang dari berbagai substring. Matriks ini diisi dengan mengiterasi melalui kedua string, membandingkan karakter satu per satu. Jika karakter cocok, nilai dalam matriks diperbarui berdasarkan nilai sebelumnya, sedangkan jika tidak cocok, nilai maksimum dari dua kemungkinan diperhitungkan. Proses ini memastikan bahwa semua kemungkinan subsekuens dievaluasi secara efisien. Pada akhirnya, panjang LCS ditemukan di sel terakhir dari matriks.

## ***2.6 Aplikasi***

Aplikasi yang kami buat dalam bahasa C# dengan kakas Visual Studio .NET. Program dapat menerima masukan sebuah citra sidik jari yang ingin dicocokkan. Program kami memiliki basis data SQL untuk mencocokkan berkas citra sidik jari yang telah ada dengan seorang pribadi. Apabila citra tersebut memiliki kecocokan di atas batas tertentu dengan citra yang sudah ada, maka akan ditunjukkan biodata orang tersebut. Apabila di bawah nilai yang telah ditentukan

tersebut, memunculkan pesan bahwa sidik jari tidak dikenali. Program memiliki keluaran yang minimal mengandung seluruh data orang tersebut. Pengguna dapat memilih algoritma yang ingin digunakan antara KMP atau BM. Biodata yang ditampilkan harus biodata yang memiliki nama yang benar kami menggunakan Regex untuk memperbaiki nama yang rusak dan gunakan KMP atau BM untuk mencari orang yang paling sesuai.

## BAB 3

### ANALISIS PEMECAHAN MASALAH

#### 3.1 Pemecahan Masalah

Masalah yang ingin dipecahkan adalah mencari kemiripan dari sebuah gambar sidik jari (*image user*) dengan gambar sidik jari lain (*image database*) kemudian menampilkan dari user yang dirasa paling mirip ke aplikasi. Masalah tersebut dapat dipecahkan menggunakan algoritma Knuth-Morris-Pratt (KMP) atau algoritma Boyer-Moore (BM).

Permasalahan tersebut dapat didekomposisi menjadi beberapa permasalahan sebagai berikut :

1. Memahami algoritma KMP, BM, dan LCS dan cara mengimplementasikannya dalam mencari kemiripan antara dua gambar sidik jari.
2. Memahami data struktur yang akan dipakai dalam implementasi algoritma KMP, BM, dan LCS.
3. Memahami mekanisme *string matching* yaitu pencocokan dua buah string yang berbeda
4. Memahami bahasa pemrograman C# yang akan digunakan untuk mengimplementasi algoritma KMP, BM, dan LCS.
5. Mengimplementasi algoritma KMP, BM, dan LCS yang sederhana yang akan digunakan sebagai *template/dasar* implementasi pencarian kemiripan antara dua gambar sidik jari.
6. Mengembangkan algoritma KMP, BM, dan LCS untuk mencari semua rute terpendek antara dua node
7. Mengimplementasikan algoritma KMP, BM, dan LCS untuk mencari kemiripan antara dua gambar sidik jari
8. Memahami mekanisme pengembangan sebuah aplikasi desktop menggunakan Windows Form
9. Memahami mekanisme mengintegrasikan aplikasi desktop dengan basis data
10. Mengembangkan visualisasi sebuah aplikasi desktop dengan kakas Visual Studio

Permasalahan tersebut dapat dipecah menjadi beberapa bagian :

1. Fungsi yang dapat mengambil ASCII *string* dari gambar sidik jari

2. Fungsi yang dapat mencari kemiripan dari dua gambar sidik jari menggunakan algoritma Knuth-Morris-Pratt (KMP)
3. Fungsi yang dapat mencari kemiripan dari dua gambar sidik jari menggunakan algoritma Boyer-Moore (BM)
4. Fungsi yang dapat mencari kemiripan dari dua gambar sidik jari menggunakan algoritma Longest Common Subsequence (LCS)
5. Fungsi regular expression dari nama yang diperoleh dari database sidik jari
6. Fungsi pencocokan nama hasil normalisasi dari regular expression dengan nama yang ada di database biodata

### **3.2 Penyelesaian Solusi**

Untuk menyelesaikan masalah ini, penulisan menganggap gambar sidik jari dari pengguna sebagai acuan, hal ini berarti gambar sidik jari akan dikonversi ke ASCII dan berfungsi sebagai *text*. Gambar sidik jari di database sebagai pembanding juga akan dikonversi ke ASCII dan berfungsi sebagai *pattern*.

Pencocokan dilakukan dengan iterasi untuk setiap gambar sidik jari dari database. Text dan pattern kemudian dicocokkan dengan algoritma KMP atau BM sesuai dengan pilihan pengguna, jika ternyata terdapat sidik jari yang cocok maka akan ditampilkan gambar sidik jari yang sesuai beserta dengan persentase kemiripannya yang harus diatas threshold yang telah kami tentukan. Kemiripan akan ditentukan menggunakan algoritma LCS.

Dari database sidik jari akan diperoleh nama pemilik dari sidik jari tersebut berupa sebuah string yang bisa jadi sudah dalam bentuk normal atau masih alay, sehingga diperlukan regular expression untuk menormalisasi string tadi. Setelah didapatkan string dalam bentuk normal akan dilakukan pencocokan lagi dengan iterasi untuk setiap nama di database biodata menggunakan algoritma yang sama yaitu LCS. Nama dari biodata yang paling mirip yang akan dijadikan hasil dan datanya akan ditampilkan.

Text	ASCII hasil konversi gambar sidik jari user
Pattern	ASCII hasil konversi gambar sidik jari database
Regex	Nama pemilik sidik jari dari database sidik jari

Kemiripan	Panjangnya string hasil LCS dibandingkan dengan string terbanding dalam persentase
-----------	--

**Tabel 3.2.1 Elemen pada algoritma KMP, BM, dan LCS**

Berikut adalah proses pada implementasi convert gambar ke ASCII sebagai *text*:

1. Gambar yang diupload harus berupa BMP
2. Algoritma akan mengambil pixel dalam gambar dalam bentuk 0 dan 1. Dimana putih akan diubah menjadi 0 dan hitam diubah menjadi 1
3. Tiap 1 kolom di 8 baris akan diubah menjadi huruf ASCII
4. Terus iterasi sampai kolom paling akhir, lalu lanjut dengan mengubah 8 baris setelahnya dari kolom pertama,
5. Lakukan hal ini terus menerus sampai baris habis dengan kelipatan 8.

1	0	0	1	1	0
0	1	1	1	1	0
1	0	1	1	0	1
1	0	0	1	1	0
0	0	1	1	0	1
0	1	0	0	1	0
1	1	0	1	1	0
1	1	1	0	0	1
1	0	0	0	0	1
1	0	1	1	1	0
0	1	0	1	1	0
1	0	0	0	1	1
0	1	0	1	0	0
1	0	1	0	0	1
1	0	1	0	1	0
0	1	1	1	0	0
1	1	1	0	1	0
0	0	1	1	0	0
0	1	1	0	1	0
1	0	0	1	0	1
0	0	1	1	0	1
1	1	0	0	1	1
0	1	0	0	1	1
0	0	1	1	0	1

Seperti gambar disamping, algoritma akan memproses tabel warna kuning terlebih dahulu dengan mengambil 8 baris pertama.

Setelah itu barulah warna orange yang diproses dengan memulai dari kolom 1 sampai akhir (6).

Yang terakhir barulah warna hijau yang diproses dengan memulai dari kolom 1 sampai akhir (6).

Berikut adalah proses pada implementasi convert gambar ke ASCII sebagai *Pattern*:

1. Algoritma sama seperti pada text, namun pada pattern hanya diambil bagian tengahnya saja
2. Cari 8 baris di tengah, lalu untuk tiap kolom ubah menjadi huruf ASCII

1	0	0	1	1	0
0	1	1	1	1	0
1	0	1	1	0	1
1	0	0	1	1	0
0	0	1	1	0	1
0	1	0	0	1	0
1	1	0	1	1	0
1	1	1	0	0	1
1	0	0	0	0	1
1	0	1	1	1	0
0	1	0	1	1	0
1	0	0	0	1	1
0	1	0	1	0	0
1	0	1	0	0	1
1	0	1	0	1	0
0	1	1	1	0	0
1	1	1	0	1	0
0	0	1	1	0	0
0	1	1	0	1	0
1	0	0	1	0	1
0	0	1	1	0	1
1	1	0	0	1	1
0	1	0	0	1	1
0	0	1	1	0	1

Hanya tabel warna orange yang diproses dan diubah menjadi huruf ASCII urut dari kolom pertama hingga akhir (6).

Berikut adalah proses pada implementasi algoritma KMP :

1. Sebelum melakukan pencarian dalam text, algoritma KMP terlebih dahulu memproses pola untuk membuat "border array". Border array mencatat panjang prefiks terpanjang dari pola yang juga merupakan sufiks.
2. Algoritma mulai mencocokkan karakter pola dengan karakter text dari awal. Jika karakter cocok, indeks untuk text (i) dan indeks untuk pattern (j) keduanya bertambah.

3. Jika seluruh pola cocok dengan bagian teks (ketika  $j$  mencapai panjang pola), maka pola ditemukan.
4. Jika terjadi ketidakcocokan setelah beberapa karakter cocok (saat pattern ( $j$ ) tidak sama dengan text ( $i$ ), algoritma tidak memulai pencarian dari awal lagi. Sebaliknya, menggunakan border array, indeks pola  $j$  diatur ke nilai yang disimpan dalam border array.
5. Jika ketidakcocokan terjadi pada karakter pertama dari pola ( $j$  sama dengan 0), maka indeks text  $i$  akan digeser satu posisi ke kanan. Namun, jika beberapa karakter sudah cocok sebelumnya ( $j > 0$ ), indeks text  $i$  tidak diubah, dan indeks pola  $j$  diatur ulang berdasarkan border array.

Berikut adalah proses pada implementasi algoritma BM :

1. Algoritma BM menggunakan tabel yang mencatat posisi terakhir setiap karakter dalam pola. Fungsi BuildLast membuat tabel ini dengan ukuran untuk semua karakter. Setiap entri dalam tabel berisi indeks terakhir kemunculan karakter tersebut dalam pola. Jika karakter tidak ada dalam pola, entri diatur ke -1.
2. Pencocokan dimulai dari karakter paling kanan pola dengan karakter yang sesuai dalam teks.
3. Jika terjadi ketidakcocokan, tabel pencocokan terakhir digunakan untuk menentukan seberapa jauh pola dapat digeser ke kanan, dengan menghindari pencocokan ulang terhadap karakter yang sudah diperiksa.
4. Ketika karakter dalam teks tidak cocok dengan karakter dalam pola, algoritma menggunakan tabel pencocokan terakhir untuk menentukan seberapa jauh pola harus digeser.
5. Jika karakter teks tidak ada dalam pola, pola digeser melewati seluruh pola. Jika karakter ada dalam pola, pola digeser sehingga karakter teks sejajar dengan kemunculan terakhirnya dalam pola.

Berikut adalah proses pada implementasi Regex :

1. Pola regex yang digunakan adalah "[0123456][ [a-z]]". Pola ini memiliki dua bagian:
  - [0123456]: mencocokkan setiap karakter angka dari 0 hingga 6.

- [ [a-z]]: mencocokkan setiap huruf kecil dari 'a' hingga 'z' dan juga spasi.

Kedua bagian ini dipisahkan oleh tanda |, yang berarti "atau", sehingga pola mencocokkan angka 0-6 atau huruf kecil atau spasi.

2. Fungsi ini menggunakan pernyataan switch untuk menggantikan karakter yang sesuai:

- "6" digantikan dengan "g".
- "1" digantikan dengan "i".
- "0" digantikan dengan "o".
- "5" digantikan dengan "s".
- "2" digantikan dengan "z".
- "4" digantikan dengan "a".
- "3" digantikan dengan "e".

3. Jika karakter tidak cocok dengan kasus di atas, karakter tersebut dikembalikan tanpa perubahan.
4. Regex.Replace(resultAngka.ToLower(), @"\b[a-z]", match => match.Value.ToUpper()) menggantikan huruf pertama setiap kata dalam string yang dihasilkan (resultAngka) dengan huruf kapital.
5. resultAngka.ToLower() memastikan bahwa seluruh string diubah menjadi huruf kecil sebelum menggantikan huruf pertama setiap kata.
6. Pola regex \b[a-z] mencocokkan huruf kecil di awal kata (ditandai oleh \b, yang merupakan batas kata).

Berikut adalah proses pada implementasi algoritma LCS :

1. Algoritma LCS menggunakan pendekatan pemrograman dinamis dengan membangun tabel dua dimensi c untuk menyimpan panjang subsekuens terpanjang untuk pasangan substring dari X dan Y. Tabel c berukuran  $(m + 1) \times (n + 1)$ , di mana m adalah panjang string X dan n adalah panjang string Y. Setiap entri  $c[i, j]$  akan menyimpan panjang LCS dari substring  $X[0..i-1]$  dan  $Y[0..j-1]$ .
2. Baris dan kolom pertama dari tabel c diinisialisasi dengan nilai 0. Ini mewakili kasus dasar ketika salah satu string memiliki panjang 0, sehingga LCS juga memiliki panjang 0.  $c[i, 0]$  dan  $c[0, j]$  diatur ke 0 untuk semua i dan j.

3. Algoritma mengisi tabel c dengan membandingkan karakter dari string X dan Y.
4. Jika karakter pada posisi i-1 di X sama dengan karakter pada posisi j-1 di Y, maka  $c[i, j] = c[i-1, j-1] + 1$ . Ini berarti karakter tersebut menjadi bagian dari LCS.
5. Jika tidak sama, maka  $c[i, j] = \text{Math.Max}(c[i-1, j], c[i, j-1])$ . Ini berarti LCS adalah LCS terpanjang dari salah satu string yang diperpendek satu karakter

Berikut adalah proses connection dengan database menggunakan MySql :

1. Database harus ada terlebih dahulu dalam MySql di penyimpanan lokal
2. Lalu sesuaikan port, password, username pada ConnectionString
3. Disediakan fungsi GetNamaSidikJari dengan masukan string path ke gambar yang dihasilkan. Contoh: “test\gambar (1).BMP”.
4. Fungsi GetNamaSidikJari akan menghasilkan nama. Namun karena nama korupt / menggunakan bahasa alay, harus dikonversikan terlebih dahulu dengan regex untuk mendapatkan nama yang sebenarnya.
5. Setalah mendapatkan nama yang sebenarnya, buat List<String> berisikan semua nama yang ada pada tabel biodata dengan fungsi GetBiodataEntries\_NamaOnly()
6. Lalu cocokan semua nama pada list dengan nama hasil fungsi GetNamaSidikJari
7. Setelah menemukan nama yang cocok, gunakan fungsi GetBiodata(string nama) dengan masukan nama yang cocok tadi. Fungsi tersebut akan menyimpan Dictionary berupa biodata.

### **3.3 Fitur**

Fitur Utama :

1. Mencari data diri orang yang paling sesuai antara database sidik jari, database biodata, dengan gambar sidik jari masukan pengguna
2. Dialog untuk pengambilan gambar

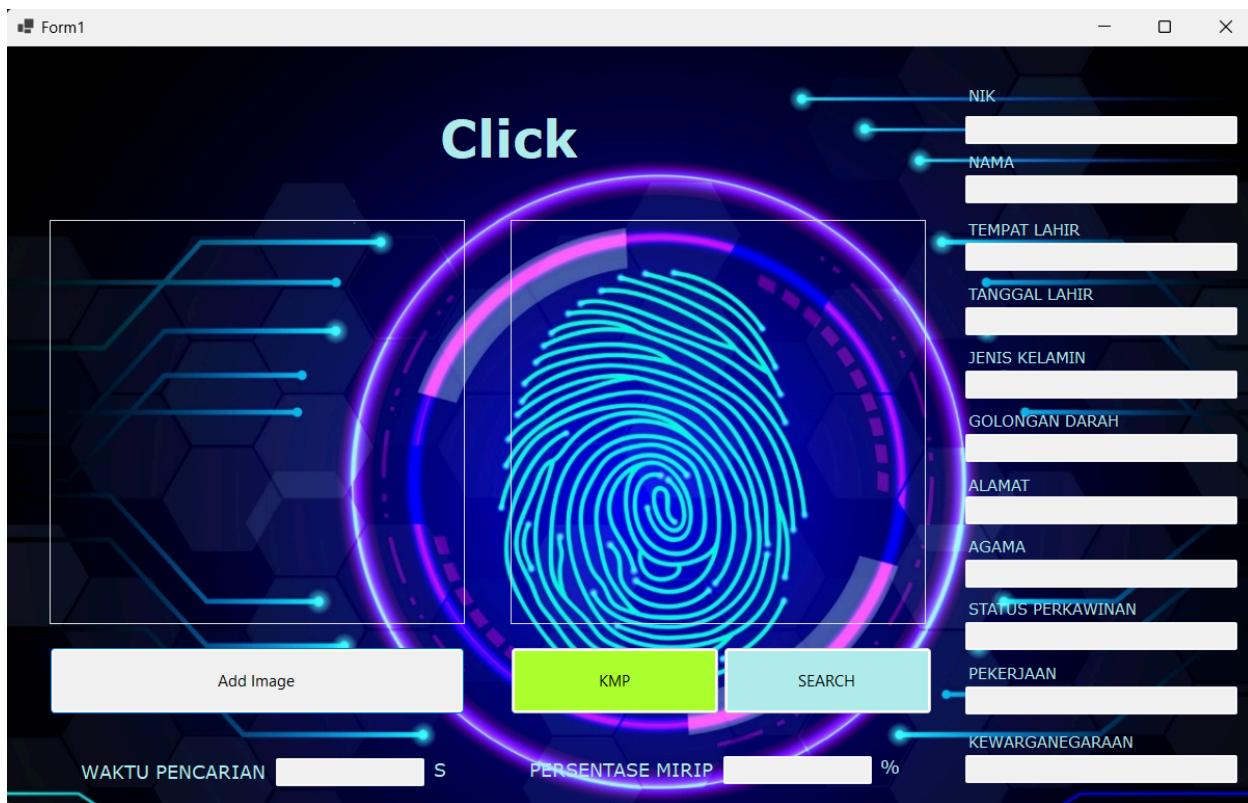
Teknologi yang kami gunakan :

1. WindowsFormApps
2. .NET

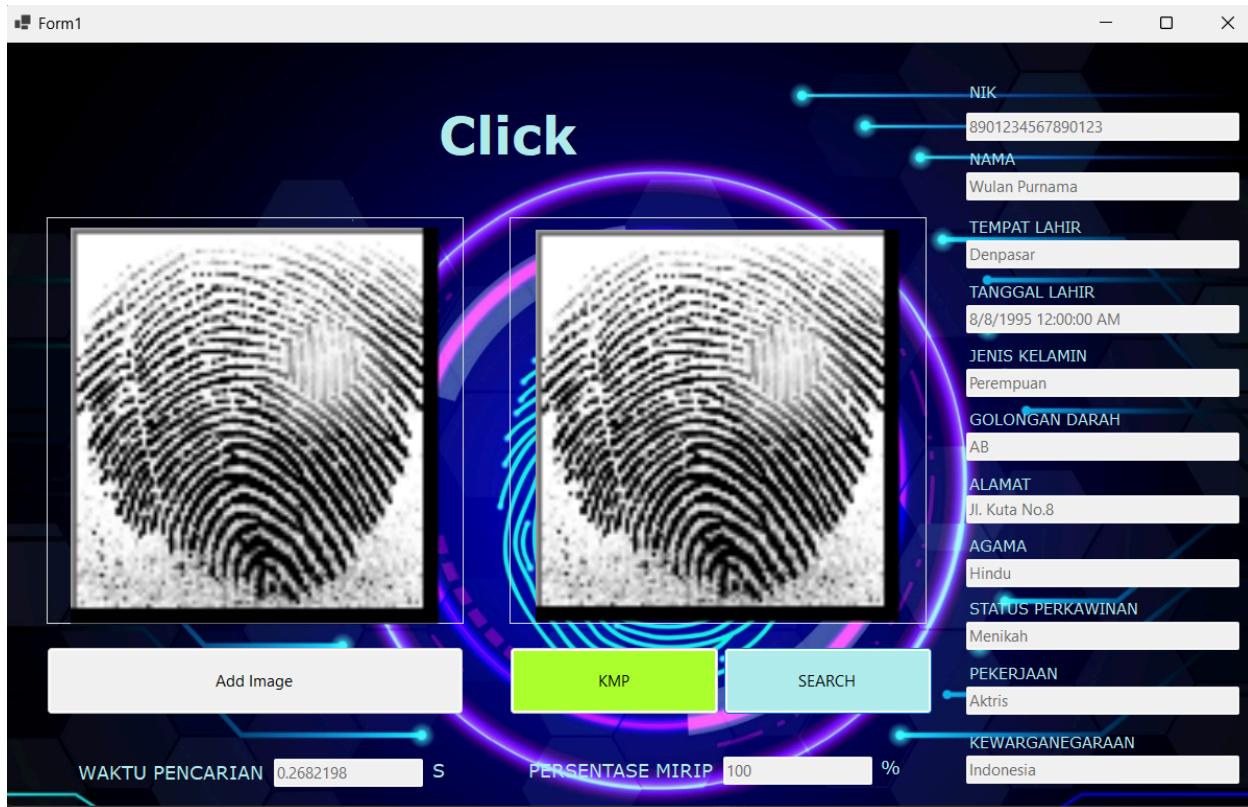
3. NuGet
4. Sql
5. C#

### 3.4 Ilustrasi Kasus

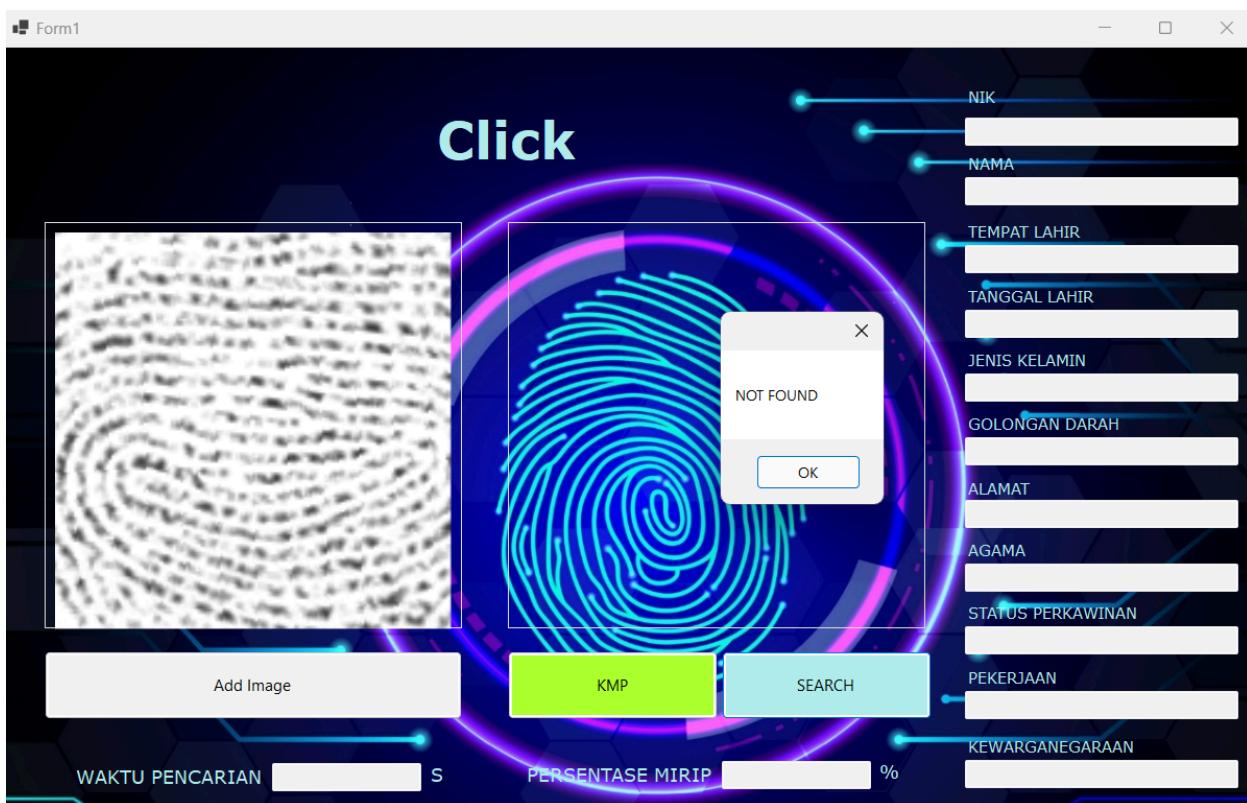
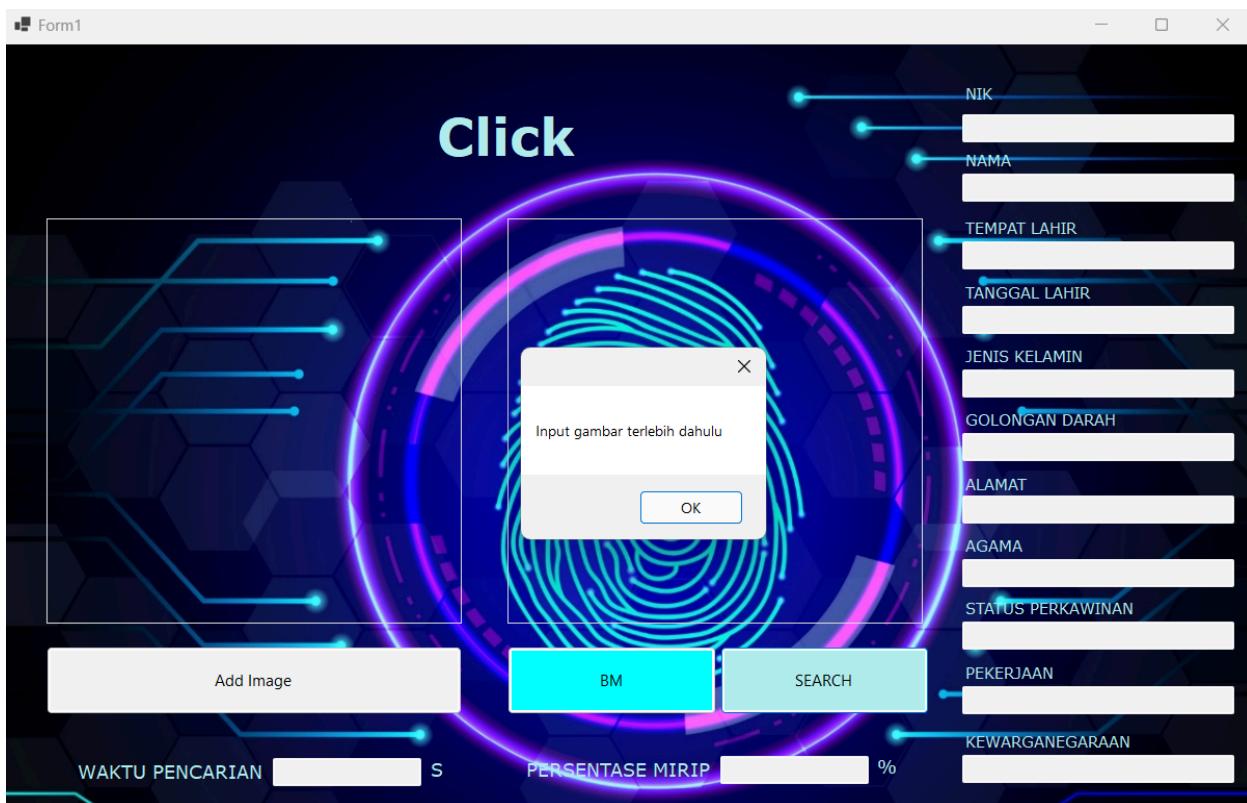
**Halaman Utama**



### Input Gambar serta Hasil Pencarian



### Tampilan Pencarian yang Gagal



## BAB 4

### Implementasi dan Pengujian

#### 4.1 Spesifikasi Teknis

Algoritma\_KMP.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace WinFormsApp1
{
    public class Algorithm_KMP
    {
        public static int KMP(string text, string pattern)
        {
            int n = text.Length;
            int m = pattern.Length;
            int[] b = borderKMP(pattern);
            int i = 0;
            int j = 0;

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

```
        return -1;
    }
    public static int[] borderKMP(string pattern)
    {
        int m = pattern.Length;
        int[] b = new int[m];
        b[0] = 0;
        int j = 0;
        int i = 1;

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

### Algoritma\_LCS.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace WinFormsApp1
{
```

```
public class Algorithm_LCS
{
    public static int[,] LCSLength(string X, string Y)
    {
        int m = X.Length;
        int n = Y.Length;
        int[,] c = new int[m + 1, n + 1];

        for (int i = 0; i <= m; i++)
        {
            for (int j = 0; j <= n; j++)
            {
                if (i == 0 || j == 0)
                    c[i, j] = 0;
                else if (X[i - 1] == Y[j - 1])
                    c[i, j] = c[i - 1, j - 1] + 1;
                else
                    c[i, j] = Math.Max(c[i - 1, j], c[i, j - 1]);
            }
        }

        return c;
    }

    public static string PrintLCS(int[,] c, string X, string Y, int i, int j)
    {
        if (i == 0 || j == 0)
            return "";

        if (X[i - 1] == Y[j - 1])
            return PrintLCS(c, X, Y, i - 1, j - 1) + X[i - 1];

        if (c[i, j - 1] > c[i - 1, j])
            return PrintLCS(c, X, Y, i, j - 1);
        else
            return PrintLCS(c, X, Y, i - 1, j);
    }
}
```

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Text.RegularExpressions;
using System.Threading.Tasks;
using System.Linq;

namespace WinFormsApp1
{
    public class Algorithm_Regex
    {
        public string Reg(string input)
        {
            string pattern = "[0123456]|[[a-z]]";
            string replacement(Match match)
            {
                switch (match.Value)
                {
                    case "6": return "g";
                    case "1": return "i";
                    case "0": return "o";
                    case "5": return "s";
                    case "2": return "z";
                    case "4": return "a";
                    case "3": return "e";
                    default: return match.Value;
                }
            }

            string resultAngka = Regex.Replace(input, pattern, new
MatchEvaluator(replacement));
            string resultTitle = Regex.Replace(resultAngka.ToLower(), @"\b[a-z]",

match => match.Value.ToUpper());

            return resultTitle;
        }
    }
}
```

### Algoritma\_BM.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace WinFormsApp1
{
    public class Algoritma_BM
    {
        public static int BM(string text, string pattern)
        {
            if (string.IsNullOrEmpty(text) || string.IsNullOrEmpty(pattern))
                return -1; // no match if text or pattern is null or empty

            int[] last = BuildLast(pattern);
            int n = text.Length;
            int m = pattern.Length;
            int i = m - 1;

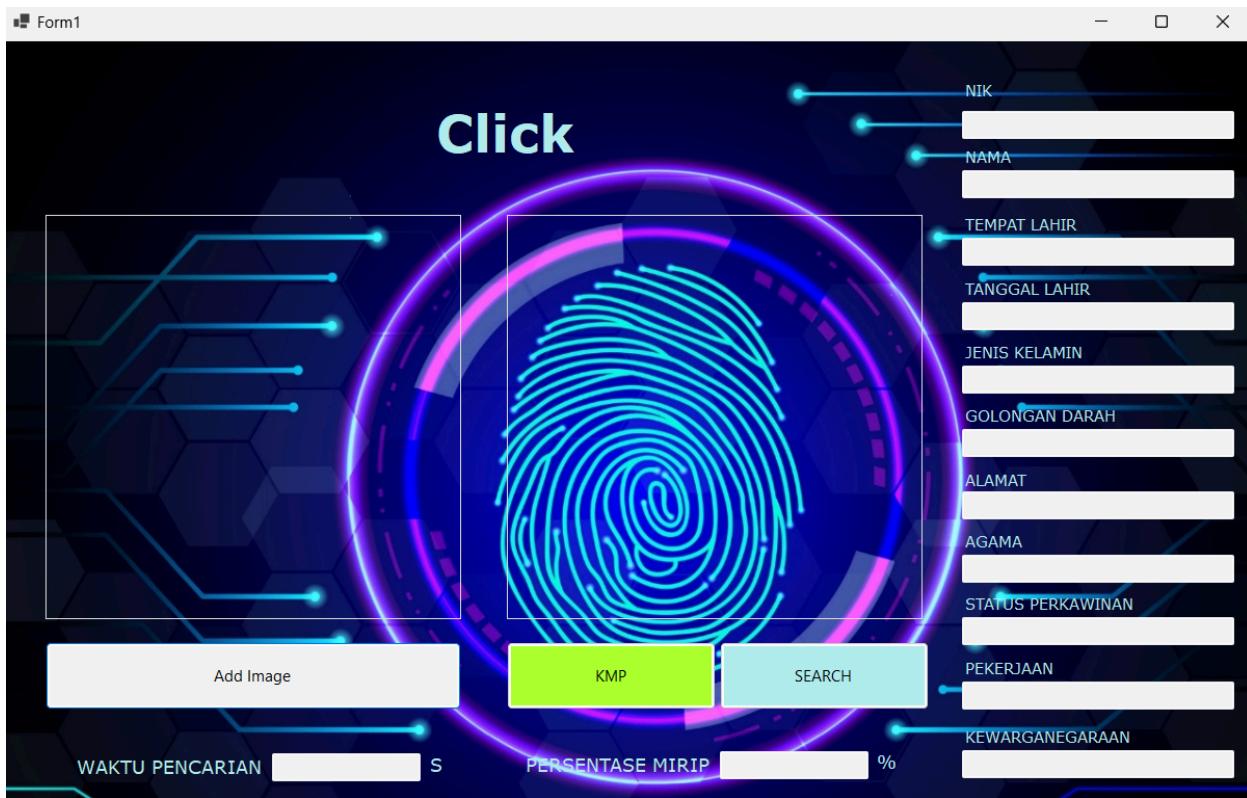
            if (i > n - 1)
                return -1; // no match if pattern is longer than text

            int j = m - 1;

            do
            {
                if (pattern[j] == text[i])
                {
                    if (j == 0)
                        return i; // match
                    else
                    {
                        i--;
                        j--;
                    }
                }
                else
            }
```

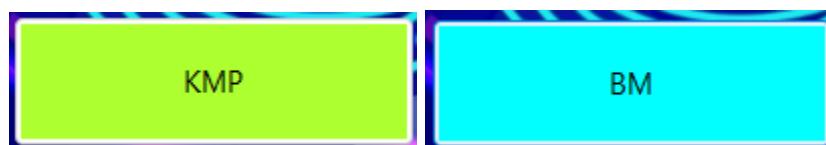
```
{  
    int lo = last[text[i]]; // last occurrence  
    i = i + m - Math.Min(j, 1 + lo);  
    j = m - 1;  
}  
} while (i <= n - 1);  
  
return -1; // no match  
}  
  
public static int[] BuildLast(string pattern)  
{  
    int[] last = new int[char.MaxValue + 1]; // support for all Unicode  
characters  
    for (int i = 0; i < last.Length; i++)  
        last[i] = -1; // initialize array  
  
    for (int i = 0; i < pattern.Length; i++)  
        last[pattern[i]] = i;  
  
    return last;  
}  
}  
}
```

#### 4.2 Penggunaan Aplikasi

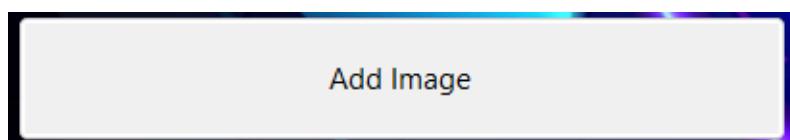


Garis besar penggunaan aplikasi Click adalah sebagai berikut :

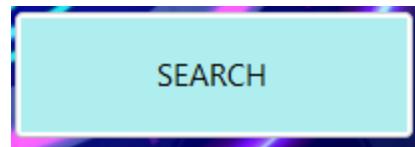
1. Pilih terlebih dahulu jenis algoritma yang ingin digunakan dengan menekan tombol pilihan algoritma untuk berganti dari satu jenis algoritma ke algoritma lain.



2. Pilih gambar yang ingin dicocokkan dengan menekan tombol Add Image



3. Mulai Pencarian dengan menekan tombol Search



- Hasil Identitas akan terlihat pada bagian kanan aplikasi

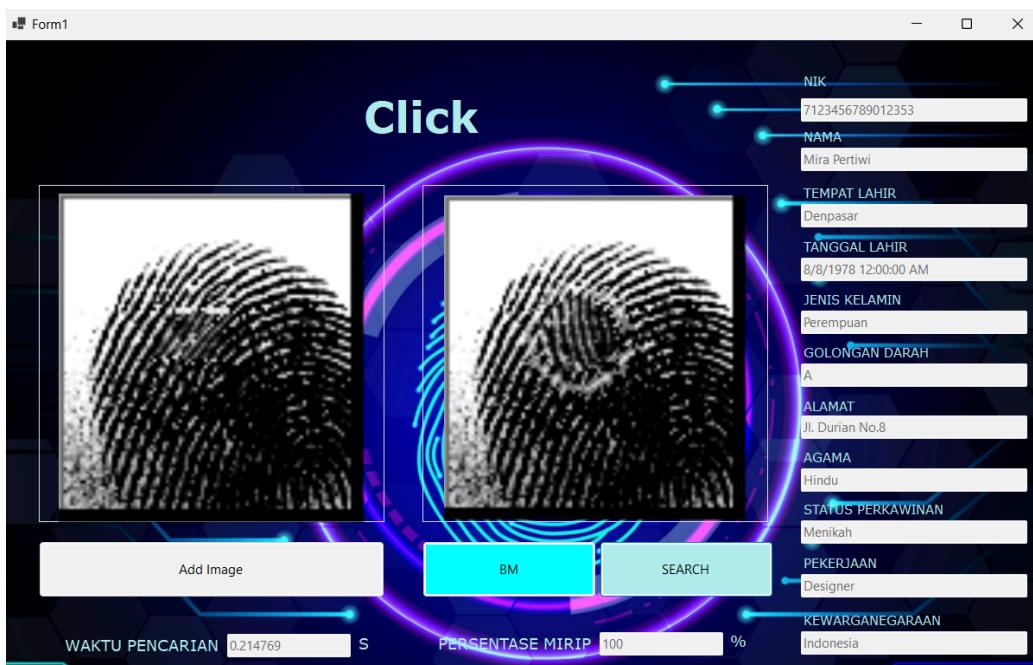
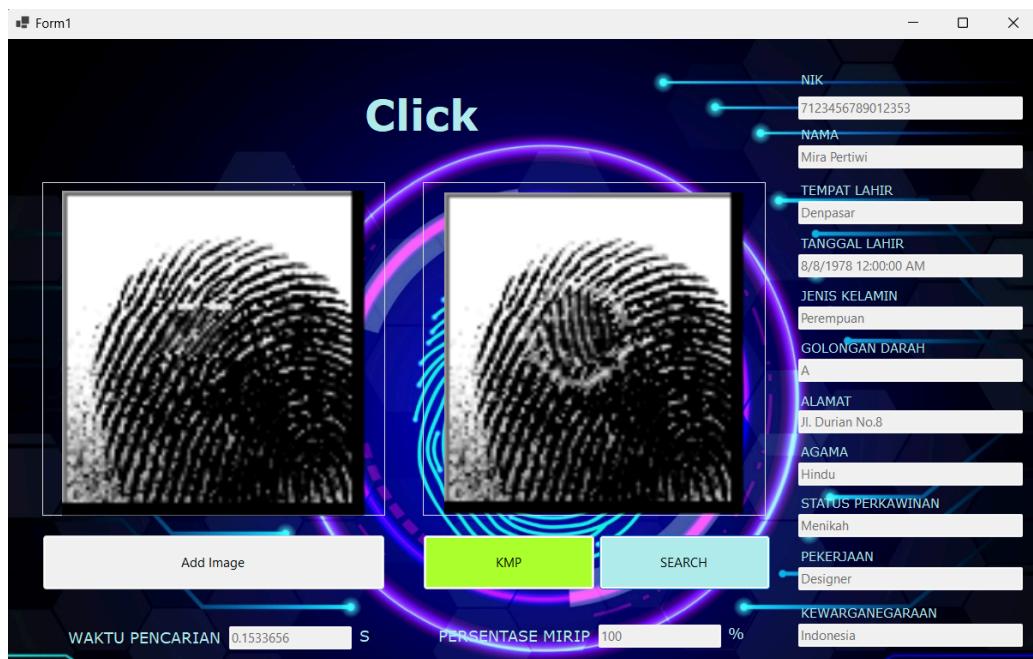
NIK	6789012345678901
NAMA	Tina Melati
TEMPAT LAHIR	Yogyakarta
TANGGAL LAHIR	6/6/1993 12:00:00 AM
JENIS KELAMIN	Perempuan
GOLONGAN DARAH	A
ALAMAT	Jl. Malioboro No.6
AGAMA	Katolik
STATUS PERKAWINAN	Menikah
PEKERJAAN	Penulis
KEWARGANEGARAAN	Indonesia

- Persentase kemiripan dan waktu pencarian akan muncul pada bagian bawah aplikasi

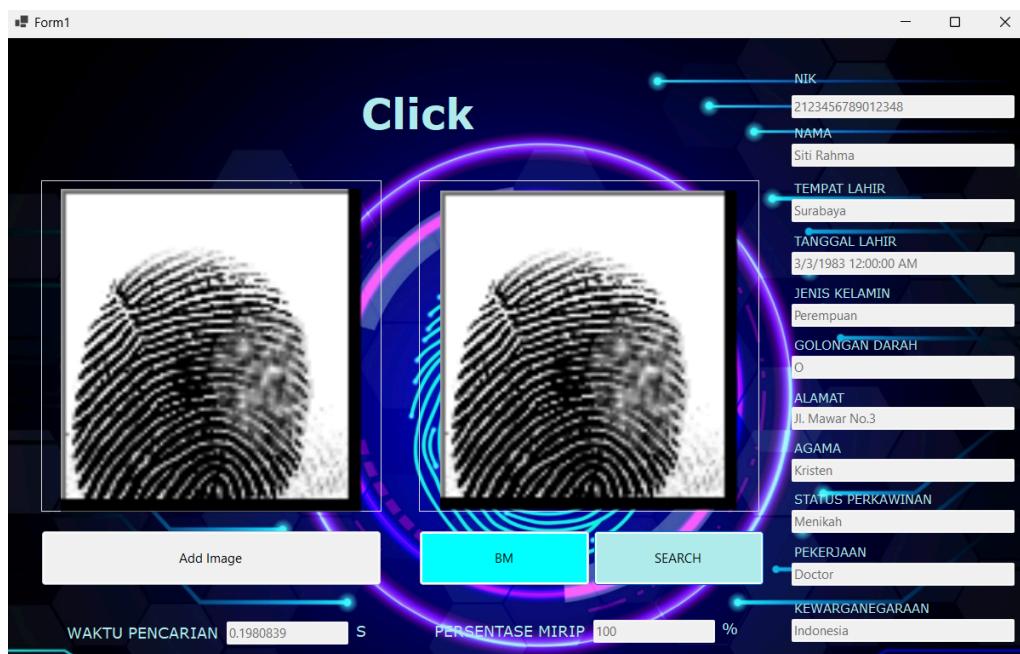


### 4.3 Hasil Pengujian

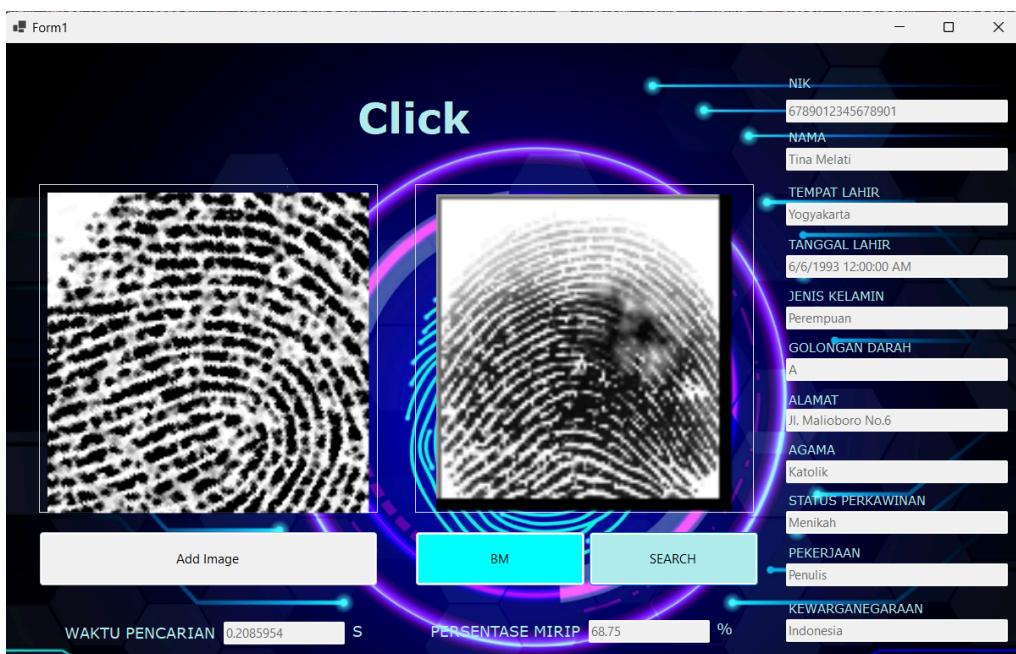
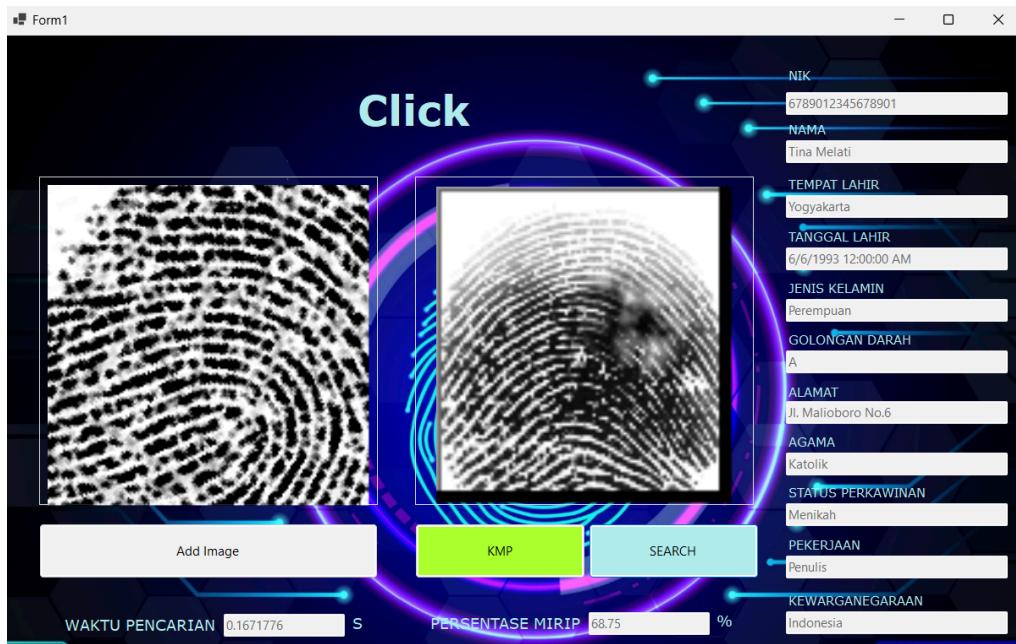
#### Perbedaan KMP dan BMP dalam Kecepatan dengan Input Gambar dari Folder yang sama dengan Test

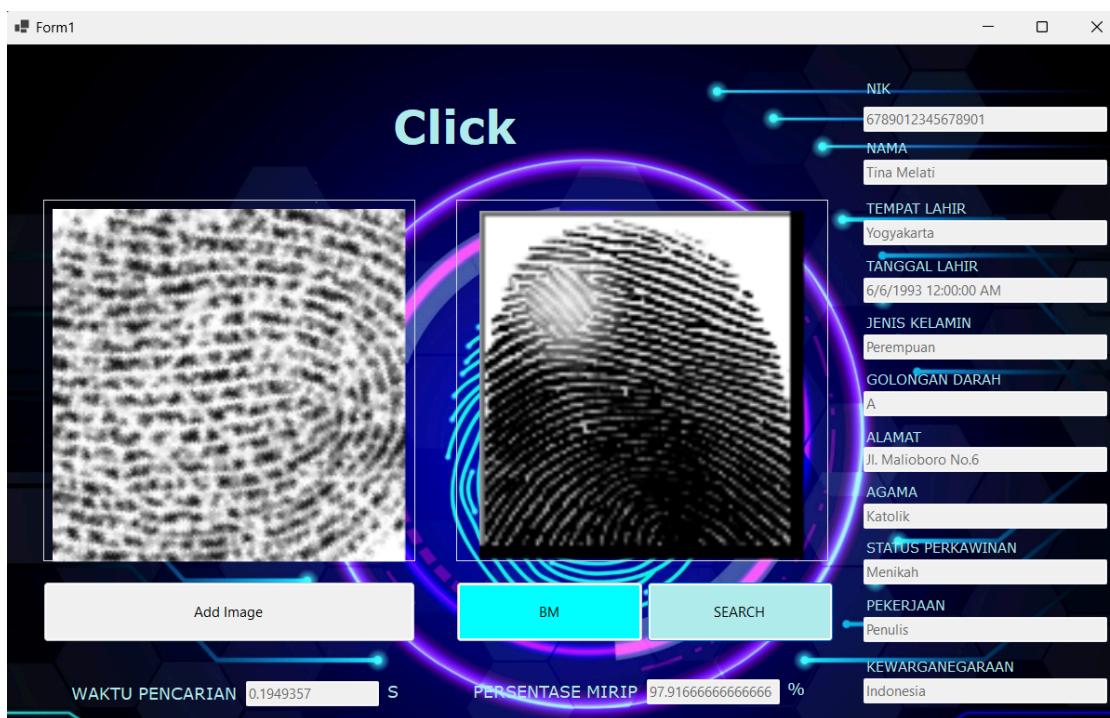
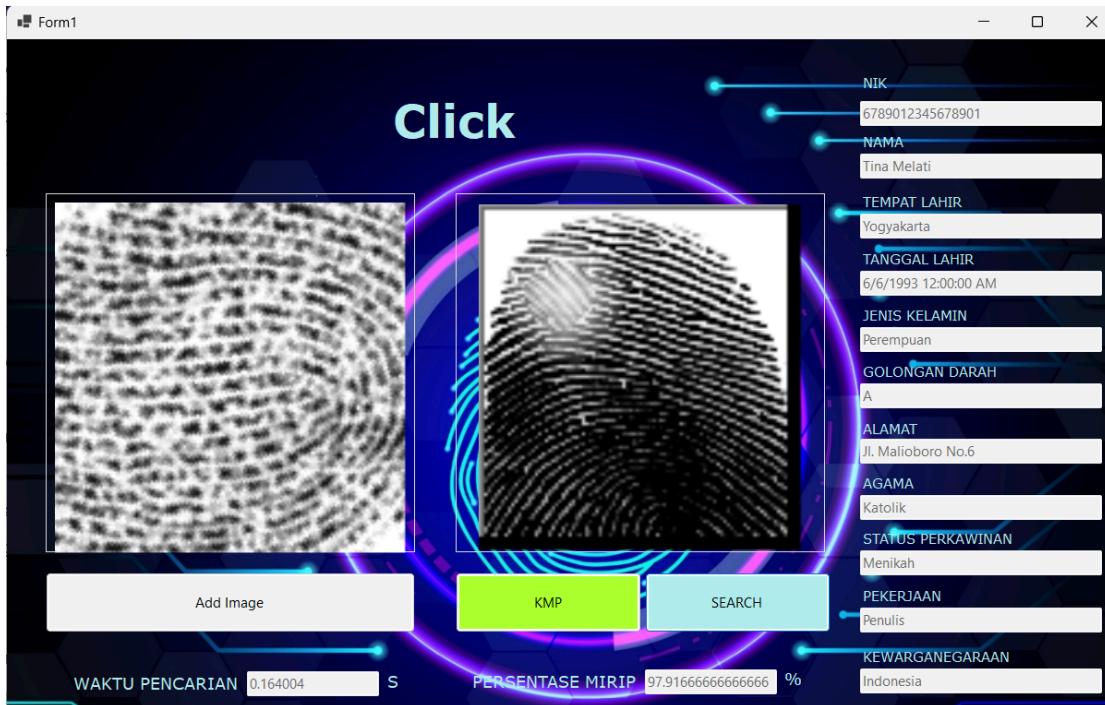


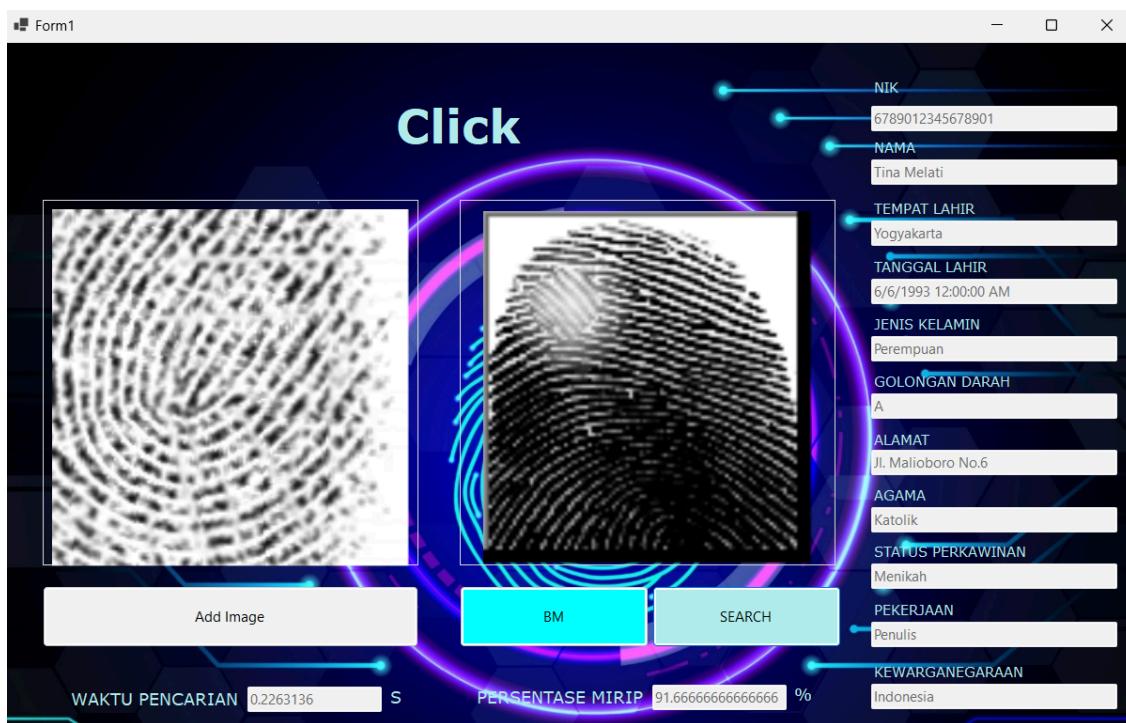
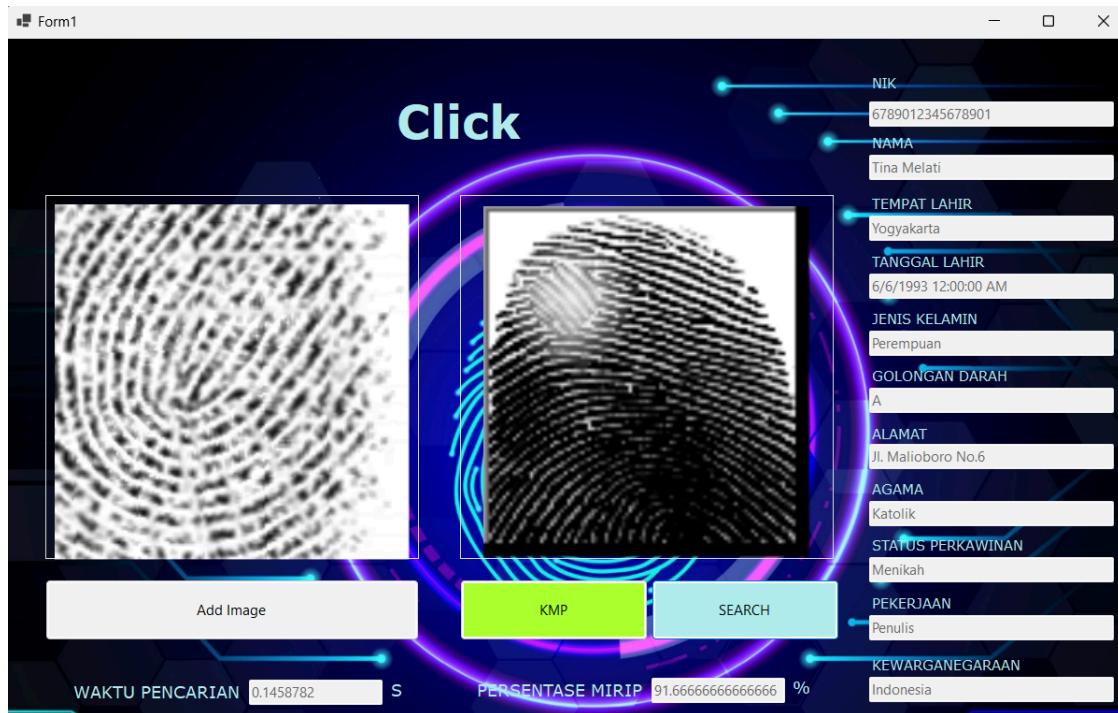




## Perbedaan KMP dan BMP dalam Kecepatan dan Persentase Kemiripan dengan Input Gambar dari Folder yang Berbeda dengan Test







## BAB 5

### Penutup

#### **5.1 Kesimpulan**

Tugas pembuatan aplikasi desktop dengan pemanfaatan Algoritma KMP dan BM merupakan suatu tantangan yang menarik dan relevan dalam mata kuliah Strategi Algoritma. Penerapan algoritma KMP dan BM tersebut memberikan kontribusi signifikan terhadap peningkatan akurasi dan keefektifan sistem dalam mencari hubungan antar satu gamabr sidik jari dengan gambar sidik jari lainnya. Dalam mencari kemiripan antara sidik jari dengan Algoritma LCS juga sangat menarik untuk didalami lebih lagi. Penggunaan metode dan teknik yang tepat dalam ekstraksi fitur, representasi data, dan pengindeksan memberikan kontribusi yang sangat signifikan terhadap hasil yang optimal.

#### **5.2 Saran**

Untuk selanjutnya manajemen waktu harus dapat dijaga dengan baik karena pengerajan masih serba dadakan sehingga kurang tereksekusi maksimal. Selain itu pembagian kerja juga harus lebih merata lagi karena beban kerja yang sekarang masih tidak seimbang sehingga tidak tereksekusi secara efisien. Dalam hal komunikasi juga masih kurang optimal sehingga terdapat miskomunikasi yang menghambat pengerajan. Penggunaan bahasa pemrograman serta kakas yang relatif baru untuk tugas besar kali ini juga menjadi tantangan tersendiri sehingga membuat kami berlatih lebih lagi.

#### **5.3 Refleksi**

Selama pengerajan tugas besar ini dirasa masih kurang dalam hal manajemen waktu karena terdapat banyak tugas besar lain dan pengalokasian waktu untuk tugas besar ini masih kurang sehingga tidak terselesaikan dengan optimal. Selain itu pembagian beban kerja yang kurang seimbang dan terdapat pula masalah komunikasi dalam kelompok.

## Daftar Pustaka

Munir, Rinaldi. 2024, “[Pencocokan string dengan Regular Expression \(Regex\)](https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2018-2019/String-Matching-dengan-Regex-2019.pdf)” Diakses 25 Mei 2024, dari

<https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2018-2019/String-Matching-dengan-Regex-2019.pdf>

Munir, Rinaldi. 2024, “[Pencocokan string \(String matching/pattern matching\)](https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Pencocokan-string-2021.pdf)” Diakses 25 Mei 2024, dari

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

<https://learn.microsoft.com/en-us/visualstudio/ide/create-csharp-winform-visual-studio?view=vs-2022>

<https://alaygenerator.blogspot.com/>

S Supatmi dan I D Sumitra. 2019, “Fingerprint Identification using Bozorth and Boyer-Moore Algorithm” Diakses 25 Mei 2024, dari

<https://iopscience.iop.org/article/10.1088/1757-899X/662/2/022040>

## Lampiran

Github : [https://github.com/keanugonza/Tubes3\\_Click.git](https://github.com/keanugonza/Tubes3_Click.git)

Video : <https://youtu.be/Fp9VFqdScY4>