

LAPORAN TUGAS BESAR 3

IF2211 STRATEGI ALGORITMA

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



Disusun oleh:

Agil Fadillah Sabri 13522006

Benjamin Sihombing 13522054

Matthew V. Hutabarat 13522093

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

DAFTAR ISI

DAFTAR ISI.....	1
DAFTAR GAMBAR.....	2
DAFTAR TABEL.....	3
BAB 1 DESKRIPSI TUGAS.....	4
1.1. Latar Belakang.....	4
1.2. Penjelasan Implementasi.....	5
BAB 2 LANDASAN TEORI.....	9
2.1. Algoritma Knuth-Morris-Pratt.....	9
2.2. Algoritma Boyer-Moore.....	10
2.3. Regular Expression.....	12
2.4. Algoritma Hamming Distance.....	12
2.5. Aplikasi Desktop dengan C#.....	13
BAB 3 ANALISIS PEMECAHAN MASALAH.....	14
3.1. Langkah-Langkah Pemecahan Masalah.....	14
3.2. Proses Penyelesaian Masalah dengan Algoritma KMP dan BM.....	15
3.3. Fitur dan Arsitektur Aplikasi Desktop.....	16
3.4. Ilustrasi Kasus.....	17
BAB 4 IMPLEMENTASI DAN PENGUJIAN.....	18
4.1. Struktur Data, Fungsi, dan Prosedur.....	18
4.2. Tata Cara Menggunakan Aplikasi.....	34
4.3. Hasil Pengujian.....	36
4.4. Analisis Pengujian.....	42
BAB 5 KESIMPULAN, SARAN, TANGGAPAN DAN REFLEKSI.....	43
5.1. Kesimpulan.....	43
5.2. Saran.....	43
5.3. Tanggapan.....	43
5.4. Refleksi.....	43
LAMPIRAN.....	44
DAFTAR PUSTAKA.....	45

DAFTAR GAMBAR

Gambar 1. Ilustrasi Fingerprint Recognition pada Deteksi Berbasis Biometrik.....	4
Gambar 2. Skema Implementasi Konversi Citra Sidik Jari.....	5
Gambar 3. Skema Basis Data yang Digunakan.....	7
Gambar 4. Ilustrasi Algoritma Knuth-Morris-Pratt.....	10
Gambar 5. Skenario Miss-Match 1.....	11
Gambar 6. Skenario Miss-Match 2.....	11
Gambar 5. Skenario Miss-Match 3.....	11
Gambar 6. Ilustrasi Algoritma Boyer-Moore.....	11
Gambar 7. Regular Expression.....	12
Gambar 8. Skema Basis Data.....	14
Gambar 9. Tangkapan Layar Regex.cs.....	19
Gambar 10. Tangkapan Layar BM.cs.....	20
Gambar 11. Tangkapan Layar BM_2D.cs.....	23
Gambar 12. Tangkapan Layar KMP.cs.....	28
Gambar 13. Tangkapan Layar HammingDistance.cs.....	28
Gambar 14. Tangkapan Layar Util.cs.....	33
Gambar 15. Remove Connection.....	35
Gambar 16. Koneksi Ulang ke Database.....	35
Gambar 17. Test-Case 1 KMP.....	36
Gambar 18. Test-Case 1 BM.....	36
Gambar 19. Test-Case 2 KMP.....	37
Gambar 20. Test-Case 2 BM.....	37
Gambar 21. Test-Case 3 KMP.....	38
Gambar 22. Test-Case 3 BM.....	38
Gambar 23. Test-Case 4 KMP.....	39
Gambar 24. Test-Case 4 BM.....	39
Gambar 25. Test-Case 5 KMP.....	40
Gambar 26. Test-Case 5 BM.....	40
Gambar 27. Test-Case 6 KMP.....	41
Gambar 28. Test-Case 6 BM.....	41

DAFTAR TABEL

Tabel 1. Hasil Konversi Citra Sidik Jari menjadi Binary dan Konversi ke ASCII 8-bits.....	6
Tabel 2. Kombinasi Ketiga Variasi Bahasa Alay Indonesia.....	7
Tabel 3. Hasil Pengujian.....	41

BAB 1

DESKRIPSI TUGAS

1.1. Latar Belakang



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.

1.2. Penjelasan Implementasi

Pada tugas ini, Anda diminta untuk mengimplementasikan sebuah program yang dapat melakukan identifikasi biometrik berbasis sidik jari. Proses implementasi dilakukan dengan menggunakan algoritma Boyer-Moore dan Knuth-Morris-Pratt, sesuai dengan yang diajarkan pada materi dan salindia kuliah.

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



Gambar 2. Skema Implementasi Konversi Citra Sidik Jari

Sumber: Dokumentasi Penulis

Gambar yang digunakan pada proses *pattern matching* kedua algoritma tersebut adalah gambar sidik jari penuh berukuran $m \times n$ pixel yang diambil sebesar 30 pixel setiap kali proses pencocokan data. Untuk tugas ini, Anda **dibebaskan** untuk mengambil jumlah pixel asalkan **didasarkan pada alasan yang masuk akal** (dijelaskan pada laporan) dan **penanganan kasus ujung yang baik** (misal jika ternyata ukuran citra sidik jari tidak habis dibagi dengan ukuran pixel yang dipilih). Selanjutnya, data pixel tersebut akan dikonversi menjadi binary. Seperti yang mungkin Anda ketahui sesuai materi kuliah (*ya kalau masuk kelas*), karena binary hanya memiliki variasi karakter satu atau nol, maka proses *pattern matching* akan membuat proses pencocokan karakter menjadi lambat karena harus sering mengulangi proses pencocokan *pattern*. Cara yang dapat dilakukan untuk mengatasi hal tersebut adalah dengan mengelompokkan setiap baris kode biner per 8 bit sehingga membentuk karakter ASCII. Karakter ASCII 8-bit ini yang akan mewakili proses pencocokan dengan string data.

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

Citra Sidik Jari	Potongan nilai binary	Potongan ASCII 8-bits
	1010000010100000101000001 0100000101000001010000010 1000001010000010100000101 0000010100000101000001010 0000101000001010000010100 0001010000010100000101000 0010100000101000001010000 0101000001010000010100000 10100000101000001010	iÿÿÿÿÿÿÿÿÿÿû:Âÿÿ<Ìÿüu wûÿkûÿ,Pþÿ_ÿÿ' »ÿÿD¾?ÿ&¼j¤Úa+È/Zíÿÿq ¼ÿúTý~Rûut`íÿÿÿÿÿ iÿÿÿÿÿÿÿÿöÿÿ"bÿÿÁüþÿÈ 'þÿœ2ÿúYÿÿ\ ^ÿþþ fÿÿà !>ÿÿþÿÿÿÿÿÿs <ðÿÖ ²ÿÿQûÿ Nÿášÿÿÿÿÿ iÿÿÿÿÿÿÿÿÿÿõrÙðÿC½ÿÿ‡â ÿ<>ÿÙþþ6Gÿÿ;Úÿÿfâÿý

Tabel 1. Hasil Konversi Citra Sidik Jari menjadi Binary dan Konversi ke ASCII 8-bits

Sumber: Dokumentasi Penulis

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

Fungsi dari deteksi biometrik adalah mengidentifikasi seseorang, oleh sebab itu, pada proses implementasi program ini, sebuah citra sidik jari akan dicocokkan dengan biodata seseorang. Biodata yang dicocokkan terdiri atas data-data yang terdapat pada KTP, antara lain : NIK, nama, tempat/tanggal lahir, jenis kelamin, golongan darah, alamat, agama, status perkawinan, pekerjaan, dan kewarganegaraan. Relasi ini dibuat dalam sebuah basis data dengan skema detail seperti yang tertera pada bagian di bawah ini. Sebagai tambahan, struktur relasi basis data telah disediakan, silakan gunakan dump sql [berikut](#). Atribut berkas_citra yang disimpan pada tabel sidik_jari adalah alamat dari citra dalam repositori (test/...), lokasi penyimpanan citra dalam folder **test**, bisa dilihat pada struktur repositori di bagian [Pengumpulan Tugas](#).

'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 3. Skema Basis Data yang Digunakan

Sumber: Dokumentasi Penulis

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 (silakan *review* kembali materi IF2240 bagian basis data relasional). Pada tugas besar kali ini, kita akan coba melakukan simulasi implementasi data korup yang **hanya mungkin terjadi pada atribut nama di tabel biodata** (asumsikan kolom lain pada setiap tabel tidak mengalami korup). Akan tetapi, karena tujuan utama program adalah mengenali identitas seseorang secara lengkap hanya dengan menggunakan sidik jari, maka harus dilakukan sebuah skema untuk menangani data korup tersebut.

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

Variasi	Hasil
Kata 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 2. Kombinasi Ketiga Variasi Bahasa Alay Indonesia

Sumber: Dokumentasi Penulis

Cara yang dapat digunakan untuk menangani ini adalah dengan menggunakan Regular Expression (Regex). Lakukan konversi pola karakter alay hingga dapat dikembalikan ke bentuk alfabetik yang bersesuaian. Setelah menggunakan Regex, Anda akan diminta kembali untuk melakukan *pattern matching* antara nama yang bersesuaian dengan algoritma KMP dan BM dengan ketentuan yang sama seperti saat [pencocokan sidik jari](#). Sebagai referensi bahasa alay, Anda dapat menggunakan *website* alay generator yang terdapat pada bagian referensi.

Setelah menemukan nama yang bersesuaian, Anda dapat menggunakan basis data untuk mengembalikan detail biodata lengkap individu. Jika seluruh prosedur diatas diimplementasikan dengan baik, maka sebuah sistem deteksi individu berbasis biometrik dengan sidik jari telah berhasil untuk diimplementasikan.

BAB 2

LANDASAN TEORI

2.1. Algoritma Knuth-Morris-Pratt

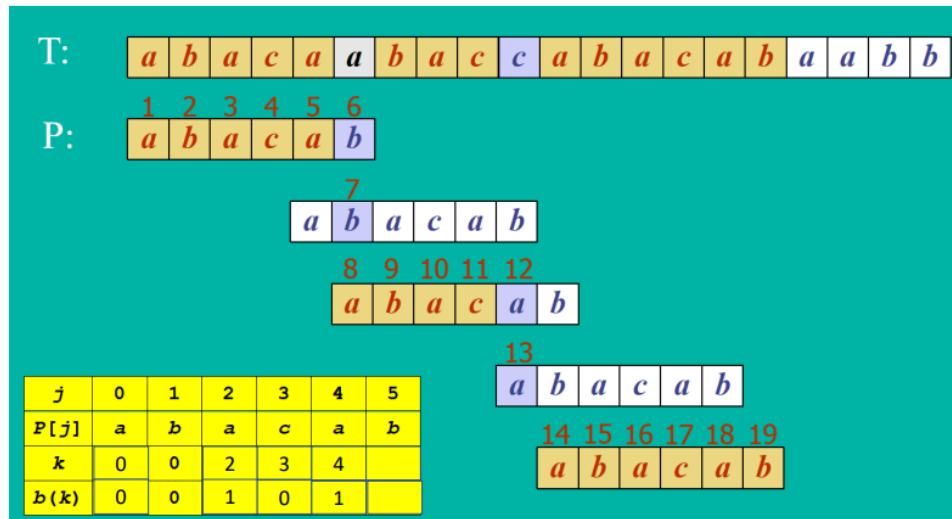
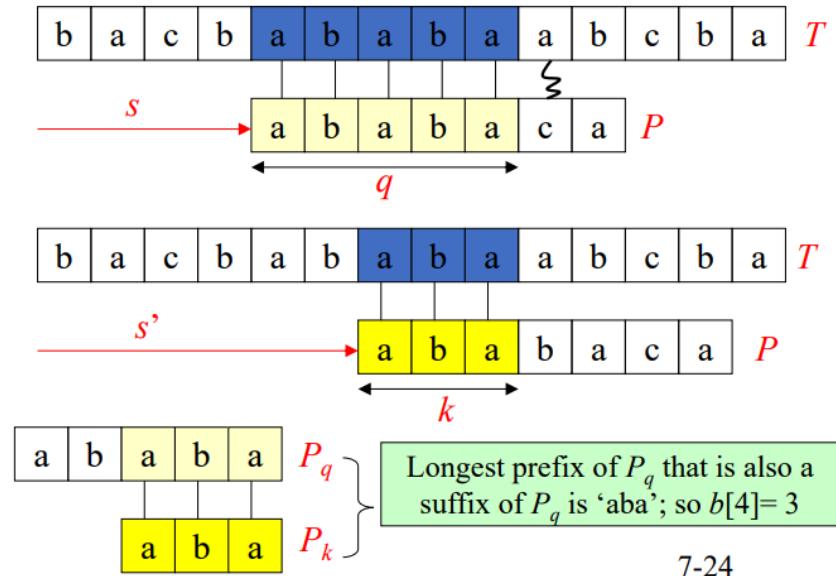
Algoritma Knuth-Morris-Pratt adalah algoritma *pattern matching* yang menelusuri teks dari kiri ke kanan. Algoritma ini memiliki kemiripan dengan algoritma *brute force*. Tetapi berbeda dengan algoritma *brute force*, pergeseran *pattern* pada algoritma Knuth-Morris-Pratt dilakukan dengan lebih baik/cerdas. Ada sebuah fungsi pinggiran (*border*) yang menghasilkan suatu *list* yang bernama *Longest Prefix Suffix*. LPS akan menyimpan data prefiks terpanjang yang juga merupakan suffix untuk setiap posisi dalam pola. *List* ini akan digunakan untuk melakukan pergeseran yang lebih baik. Berikut ini langkah-langkah algoritma Knuth-Morris-Pratt:

1. Inisialisasi dua indeks, i untuk teks dan j untuk pola, keduanya dimulai dari 0.
2. Cocokkan karakter pola $\text{pattern}[j]$ dengan karakter teks $\text{text}[i]$.
 - a. Jika $\text{pattern}[j] == \text{text}[i]$, gerakkan keduanya maju ($i++$ dan $j++$).
 - b. Jika j mencapai panjang pola ($j == M$), berarti pola ditemukan. Catat posisi awal pencocokan ($i - j$), lalu gunakan LPS array untuk menentukan posisi baru j ($j = \text{lps}[j - 1]$).
 - c. Jika terjadi ketidakcocokan setelah beberapa pencocokan karakter ($\text{pattern}[j] != \text{text}[i]$):
 - Jika $j != 0$, gunakan LPS array untuk melompat ke posisi baru j ($j = \text{lps}[j - 1]$).
 - Jika $j == 0$, gerakkan indeks teks i maju satu langkah ($i++$).

Untuk menghasilkan *list Longest Prefix Suffix*, dilakukan langkah-langkah berikut:

1. Inisialisasi array dengan panjangnya sama dengan panjang *pattern* dan setiap elemen diisi dengan nilai 0.
2. Isi nilai setiap elemen:
 - a. Selama i kurang dari panjang pola (M):
 - Jika karakter pada indeks i dari pola cocok dengan karakter pada indeks $length$ dari pola ($\text{pattern}[i] == \text{pattern}[length]$), tambah $length$ sebesar 1 ($length++$), set $\text{lps}[i]$ sama dengan $length$, dan tambah i sebesar 1 ($i++$).
 - Jika tidak cocok ($\text{pattern}[i] != \text{pattern}[length]$):
 - Jika $length$ tidak sama dengan 0, set $length$ menjadi $\text{lps}[length - 1]$.
 - Jika $length$ sama dengan 0, set $\text{lps}[i]$ ke 0 dan tambah i sebesar 1 ($i++$).

Berikut contoh ilustrasi algoritma Knuth-Morris-Pratt:

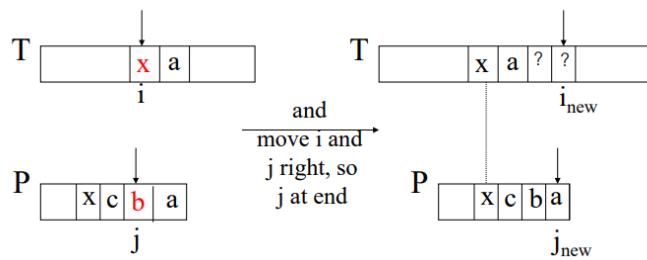


Gambar 4. Ilustrasi Algoritma Knuth-Morris-Pratt

2.2. Algoritma Boyer-Moore

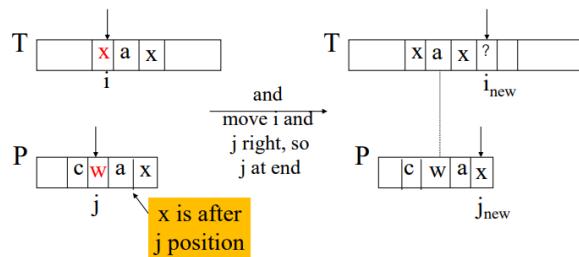
Algoritma Boyer-Moore adalah algoritma *pattern matching* yang berlandaskan teknik *looking-glass* dan *character-jump*. Teknik *looking-glass* adalah teknik mencari *pattern* di dalam *text* dengan melakukan komparasi secara mundur (dari kanan ke kiri). Teknik *character-jump* adalah teknik pemindahan karakter yang dibandingkan jika karakter yang sedang dibandingkan tidak sama. Ada 3 skenario di dalam *character-jump*:

1. Jika karakter saat ini dari *text* (misalnya x) ada di sebelah kiri karakter saat ini dari *pattern*, geser *pattern* dengan menyejajarkan x dari *text* dan *pattern* dan lakukan komparasi ulang dari ujung kanan *pattern*.



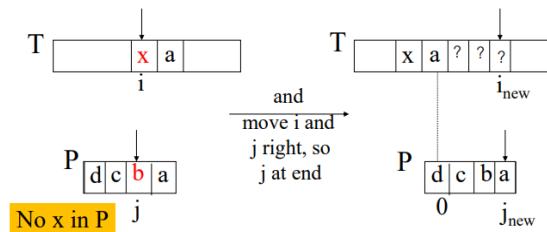
Gambar 5. Skenario Miss-Match 1

2. Jika karakter saat ini dari *text* (misalnya x) tidak ada di sebelah kiri dan ada di sebelah kanan karakter saat ini dari *pattern*, geser *pattern* ke kanan sepanjang 1 dan lakukan komparasi ulang dari ujung kanan *pattern*.



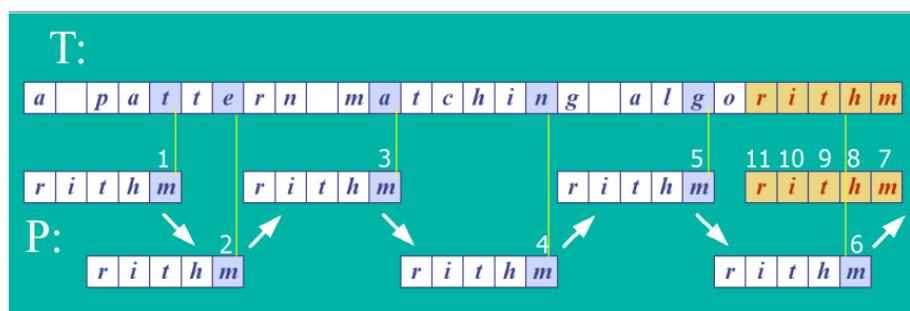
Gambar 6. Skenario Miss-Match 2

3. Jika dua skenario di atas tidak terjadi, geser *pattern* hingga ujung kiri *pattern* sejajar dengan setelah karakter saat ini di *text* dan lakukan komparasi ulang dari ujung kanan *pattern*.



Gambar 5. Skenario Miss-Match 3

Berikut ini contoh penggunaan *string matching* dengan algoritma Boyer-Moore:



Gambar 6. Ilustrasi Algoritma Boyer-Moore

2.3. Regular Expression

Regular expression adalah sebuah notasi yang dapat digunakan untuk mendeskripsikan pola dari kata yang ingin dicari. *Regular expression* bisa digunakan untuk melakukan *pattern matching*. Berikut ini beberapa *pattern* dasar *Regular expression*:

Basic Regular Expression Patterns

brackets [] : disjunction

RE	Match	Example Patterns
/ [wW] oodchuck/	Woodchuck or woodchuck	“ <u>Woodchuck</u> ”
/ [abc] /	‘a’, ‘b’, or ‘c’	“In uomini, in soldati”
/ [1234567890] /	any digit	“plenty of <u>7</u> to 5”

Brackets [] ditambah garis sambung: range

RE	Match	Example Patterns Matched
/ [A-Z] /	an uppercase letter	“we should call it ‘ <u>Drenched Blossoms</u> ’”
/ [a-z] /	a lowercase letter	“ <u>my beans were impatient to be hoed!</u> ”
/ [0-9] /	a single digit	“Chapter <u>1</u> ; Down the Rabbit Hole”

Basic Regular Expression Patterns

- caret ^ : negasi

RE	Match (single characters)	Example Patterns Matched
[^A-Z]	not an uppercase letter	“Oyin pripetchik”
[^Ss]	neither ‘S’ nor ‘s’	“I have no exquisite reason for it”
[^_ .]	not a period	“ <u>our resident Djinn</u> ”
[e^]	either ‘e’ or ‘^’	“look up <u>e</u> now”
a^b	the pattern ‘a^b’	“look up <u>a</u> <u>b</u> now”

- Tanda tanya ? : bisa ada bisa tidak

RE	Match	Example Patterns
woodchucks?	woodchuck or woodchucks	“ <u>woodchuck</u> ”
colou?r	color or colour	“ <u>colour</u> ”

- Titik: . any character

RE	Match	Example Patterns
/beg.n/	any character between beg and n	begin, <u>beg</u> n, begun

Gambar 7. Regular Expression

2.4. Algoritma Hamming Distance

Hamming Distance antara dua string adalah jumlah perbedaan tiap karakter dari 2 string tersebut. Berikut ini algoritma menghitung *hamming distance*:

```
function countHammingDistance(string s1, string s2) -> int
if s1.Length != s2.Length then
    -> -1
else
    n <- 0
    i traversal [0..Length]
        if s1[i] == s2[i] then
            n <- n + 1
    -> n
```

2.5. Aplikasi Desktop dengan C#

Salah satu kerangka kerja *antarmuka pengguna* (UI) yang memungkinkan pengembang untuk membuat aplikasi klien desktop adalah *Windows Presentation Foundation* (WPF). WPF dikembangkan oleh Microsoft yang merupakan bagian dari .NET Framework yang kemudian diintegrasikan ke dalam .NET Core dan platform .NET 5+ yang lebih baru. Keunggulan dari WPF sebagai berikut:

1. Interface yang Interaktif

WPF menyediakan *interface* dengan interaktivitas yang tinggi seperti animasi, efek, dan grafis 3D yang menarik.

2. Pemisahan Logika dan Tampilan

WPF Memisahkan logika pemrograman dan tampilan dengan XAML sebagai tampilan dan CS sebagai logika. Pemisahan ini bertujuan untuk meningkatkan modularitas.

3. Resolusi yang tinggi

WPF secara otomatis menyesuaikan tampilan untuk berbagai resolusi dan DPI, membuatnya ideal untuk aplikasi yang perlu mendukung berbagai perangkat.

BAB 3

ANALISIS PEMECAHAN MASALAH

3.1. Langkah-Langkah Pemecahan Masalah

Pada aplikasi, masalah yang ingin diselesaikan adalah mencari biodata dari input gambar sidik jari seseorang. Untuk mencari biodata tersebut, aplikasi melakukan pencarian *database*. Berikut ini skema basis data yang digunakan.

'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 8. Skema Basis Data

Untuk melakukan pencarian biodata, dilakukan langkah-langkah berikut:

1. Terima input gambar sidik jari dari pengguna.
2. Ubah gambar tersebut menjadi bentuk string yang bisa diproses oleh algoritma *string matching*.
3. Bandingkan semua gambar pada berkas_citra di tabel sidik_jari menggunakan algoritma Knuth-Morris-Pratt atau Boyer-Moore.
4. Jika gambar input cocok dengan gambar di tabel sidik_jari, nama yang bersesuaian dengan gambar akan digunakan untuk mencari biodata di tabel biodata.
5. Jika tidak ada gambar input yang cocok dengan gambar di tabel sidik_jari, akan dicari gambar yang paling mirip dengan perhitungan hamming distance dan nama yang bersesuaian dengan gambar akan digunakan untuk mencari biodata di tabel biodata.
6. Namun, data biodata mungkin saja korup sehingga nama pada tabel tersebut bisa berbentuk bahasa *alay*. Perbaiki data korup tersebut menggunakan *regular expression*.
7. Jika tidak menemukan gambar yang mirip gambar input, tampilkan pesan bahwa tidak ada gambar yang mirip.
8. Jika ada gambar yang cocok atau mirip, aplikasi tampilkan biodata yang didapat.

3.2. Proses Penyelesaian Masalah dengan Algoritma KMP dan BM

Sebelum melakukan algoritma KMP dan BM, gambar sidik jari harus diubah terlebih dahulu menjadi bentuk string yang bisa digunakan algoritma KMP dan BM. Pertama, setiap pixel di dalam gambar akan direpresentasikan ke dalam matriks yang bernilai 0 atau 1. Elemen matriks bernilai 0 jika warna pixel tersebut gelap dan bernilai 1 jika berwarna terang. Setelah mendapatkan matriks biner yang merepresentasikan gambar sidik jari tersebut, proses selanjutnya akan dilakukan oleh algoritma *pattern matching*.

Untuk proses pencocokan antara *pattern* dengan *text*, jika seandainya lebar *text* bukanlah kelipatan 8, maka akan dilakukan iterasi untuk melakukan pemotongan agar setiap kemungkinan karakter ASCII yang dapat dibentuk dapat diproses. Contohnya seperti jika lebar *text* adalah 10, maka iterasi pertama akan mengambil piksel [1..8], iterasi kedua akan mengambil piksel [2..9], dan iterasi ketiga akan mengambil piksel [3..10]. Hal ini dilakukan karena jika *pattern* yang diambil bergeser satu piksel saja, maka karakter ASCII yang terbentuk akan berubah.

Pada proses pencocokan pola sidik jari dengan algoritma, *pattern* akan diambil dari bagian tengah gambar sebesar 32×30 piksel yang kemudian diubah menjadi ASCII sehingga matriks akhirnya akan berukuran 1×30 (masing masing baris akan berisi string sebanyak 4 karakter). Pengambilan ukuran 32×30 bertujuan meningkatkan akurasi dari pencocokan.

1. KMP

Pada proses pencocokan menggunakan algortima Knuth-Morris-Pratt, akan dilakukan iterasi sebanyak $(height - 30) \times 30$ atau kurang. Iterasi sebanyak $(height - 30)$ adalah iterasi *row* dari gambar di basis data (*text*). Sedangkan, iterasi sebanyak 30 adalah iterasi pengecekan antara *row* gambar di basis data (*text*) dengan *row* gambar input (*pattern*).

2. BM

Pada proses pencocokan pola sidik jari dengan algoritma Boyer-Moore, *pattern* akan diambil dari bagian tengah gambar sebesar 32×30 piksel yang kemudian diubah menjadi ASCII sehingga matriks akhirnya akan berukuran 1×30 (masing masing baris akan berisi string sebanyak 4 karakter).

Karena *pattern* yang diambil berdimensi dua, sedangkan algoritma Boyer-Moore pada dasarnya digunakan untuk pencocokan string satu dimensi, maka dilakukan perluasan algoritma Boyer-Moore agar dapat melakukan proses pencocokan string berdimensi dua. Adapun caranya adalah sebagai berikut:

- Setiap baris matriks dianalogikan seperti satu buah karakter di dalam algoritma Boyer-Moore satu dimensi.
- Untuk tabel *last occurrence*, karena pada satu baris bisa terdapat tak terhingga kemungkinan susunan string yang dapat terbentuk, maka tabel *last occurrence* diganti dengan suatu fungsi yang digunakan untuk menentukan apakah terdapat baris pada *pattern* yang muncul pada baris text tempat *mismatch* terjadi. Dengan cara ini, teknik *character jump* dapat dilakukan. Adapun mekanismenya sama seperti pada teknik *character jump* untuk pencocokan string satu dimensi.

- Untuk proses pencocokan *pattern* dengan *text*, juga digunakan teknik *looking glass*, yaitu mencocokan *pattern* mulai dari baris terakhir dan bergerak menuju baris pertama.
- Untuk pencocokan antara satu baris *pattern* dengan *text*, dilakukan dengan algoritma pencocokan string Boyer-Moore satu dimensi.
- Secara garis besar, algoritma Boyer-Moore dua dimensi ini sama seperti algoritma Boyer-Moore satu dimensi, dengan tabel *last-occurrence* diganti dengan suatu fungsi yang mencari posisi pattern yang muncul di baris text tempat terjadi *mismatch*.

3. Hamming Distance

Jika KMP dan BM tidak bisa menemukan gambar yang cocok (*exact match*), gambar yang paling mirip yang akan digunakan. Suatu gambar dikatakan mirip jika persentase kemiripannya $\geq 90\%$. Persentase kemiripan tersebut dicari menggunakan *hamming distance*. *Hamming distance* akan mencari jumlah perbedaan pixel antara 2 buah gambar (string). Pada implementasi hamming distance, perhitungan dilakukan menggunakan tipe biner bukan ASCII.

3.3. Fitur dan Arsitektur Aplikasi Desktop

Aplikasi ini dibuat menggunakan Windows Presentation Foundation (WPF) yang men-support bahasa C#. Penyimpanan data menggunakan database menggunakan SQLite3. Kami menggunakan Python untuk *seeding* database dengan bantuan library *Faker* dan *Mockaroo* agar proses *seeding* lebih cepat. Untuk dataset sendiri kami mendapatkannya dari Kaggle ([link dataset](#)).

Fitur fitur yang tersedia pada aplikasi kami cukup ringkas karena fitur fungsionalnya cuma satu yaitu mencocokkan sidik jari. Berikut ini fitur fitur tersebut:

1. Kontainer Masukan
Kontainer ini bertujuan untuk memasukkan sidik jari yang ingin dicari di database. Sidik jari yang dimasukkan harus berformat BMP.
2. Toggle Algoritma KMP / BM
Toggle ini memilih algoritma yang dipakai menggunakan algoritma KMP atau BM
3. Tombol Pencarian
Tombol ini bertujuan memulai pencarian sidik jari dengan menggunakan algoritma yang dipilih user.
4. Kontainer Keluaran
Kontainer ini bertujuan untuk menampilkan sidik jari yang kemiripannya mencapai lebih dari 90%. Jika tidak ada sidik jari yang mirip maka Kontainer ini akan menampilkan pesan “No Fingerprint Match”
5. Kontainer Biodata
Jika terdapat sidik jari yang memiliki kemiripan lebih dari 90% maka kontainer ini akan muncul dan menampilkan biodata dari sidik jari yang mirip tersebut. Kontainer ini tidak akan terlihat jika tidak ada sidik jari yang mirip.

3.4. Ilustrasi Kasus

Suatu sidik jari ingin diidentifikasi identitasnya. Gambar sidik jari tersebut akan menjadi input aplikasi ini. Aplikasi ini akan pencarian identitas/biodata dari gambar sidik jari tersebut. Pencarian akan memiliki 3 skenario. Berikut ini 3 skenario yang mungkin terjadi:

1. Sidik jari ditemukan

Program akan menampilkan gambar sidik jari dan biodata dari basis data.

2. Sidik jari tidak ditemukan

Program akan memberikan pesan bahwa tidak ditemukan gambar sidik jari yang cocok dengan gambar sidik jari input.

3. Sidik jari yang mirip ditemukan

Program akan menampilkan gambar sidik jari dan biodata dari basis data. Program juga akan menampilkan persentase kemiripan gambar dari basis data dan gambar input.

BAB 4

IMPLEMENTASI DAN PENGUJIAN

4.1. Struktur Data, Fungsi, dan Prosedur

1. Struktur Data

```
// tidak ada struktur khusus yang dibuat  
// hanya memakai array yang tersedia secara bawaan di C#  
type[] Array;  
type[][] Matrix;
```

2. Fungsi dan Prosedur

- Regex.cs

```
public class Regex {  
    private static readonly string[] listRegex = {  
        @"\[Aa4]?",  
        "[Bb8]",  
        @"\[Cc]",  
        "[Dd]",  
        @"\[Ee3]?",  
        "[Ff]",  
        "[Gg6]",  
        "[Hh]",  
        @"\[Ii1]?",  
        "[Jj7]",  
        "[Kk]",  
        "[Ll1]",  
        "[Mm]",  
        "[Nn]",  
        "[Oo0]",  
        "[Pp]",  
        "[Qq]",  
        "[Rr]",  
        @"\[Ss5]",  
        "[Tt7]",  
        @"\[Uu]?",  
        "[Vv]",  
        "[Ww]",  
        "[Xx]",  
        "[Yy]",  
        "[Zz2]",  
        @"\[\s]*"  
    };  
};
```

```

public static string StringToRegex_TrueToAlay(string input) {
    // function to convert a string to regex pattern
    string result = "";
    foreach (char c in input) {
        if (char.ToUpper(c) == ' ') {
            result += listRegex[26];
        }
        else {
            int index = char.ToUpper(c) - 'A';
            if (index >= 0 && index < listRegex.Length - 1) {
                result += listRegex[index];
            }
        }
    }
    return result;
}

```

```

public static bool IsMatch(string input, string pattern) {
    // function to check if input matches pattern
    string regexPattern = StringToRegex_TrueToAlay(pattern);
    return System.Text.RegularExpressions.Regex.IsMatch(input, regexPattern);
}

public static List<string> FindMatches(string input, string pattern) {
    // function to find all matches of pattern in input
    string regexPattern = StringToRegex_TrueToAlay(pattern);
    MatchCollection matches = System.Text.RegularExpressions.Regex.Matches(input, regexPattern);
    List<string> matchList = new List<string>();

    foreach (Match match in matches) {
        matchList.Add(match.Value);
    }

    return matchList;
}

```

Gambar 9. Tangkapan Layar Regex.cs

- **BM.cs**

```
class BM {  
    protected static int NO_OF_CHARS = 256;  
    protected static string[] LastOccurrence = new string[NO_OF_CHARS];  
  
    protected static void Preprocess(string pattern) {  
        // find the last occurrence of each character in the pattern  
        for (int i = 0; i < NO_OF_CHARS; i++) {  
            LastOccurrence[i] = "-1";  
        }  
  
        for (int i = 0; i < pattern.Length; i++) {  
            LastOccurrence[pattern[i]] = i.ToString();  
        }  
    }  
  
    public static bool BoyerMoore(string text, string pattern) {  
        // function to search for pattern in text using Boyer-Moore algorithm  
        Preprocess(pattern);  
        int m = pattern.Length;  
        int n = text.Length;  
  
        int s = 0;           // s is position of start character of pattern relative to text  
        while (s <= (n - m)) {  
            int j = m - 1;  
            while (j >= 0 && pattern[j] == text[s + j]) {    // while characters of pattern and text match  
                j--;  
            }  
  
            if (j < 0) {          // pattern found  
                return true;  
            }  
            else {               // character mismatch  
                s += Math.Max(1, j - int.Parse(LastOccurrence[text[s + j]]));  
            }  
        }  
  
        return false;  
    }  
}
```

Gambar 10. Tangkapan Layar BM.cs

- BM 2D.cs

```
class BM_2D : BM {
    private static int FindMatches(string[] pattern, string textNotMatched) {
        // function to find the number of matches of pattern in matriks
        // replacement for the last occurrence in one-dimensional boyer moore
        for (int i = pattern.Length - 1; i >= 0; i--) {
            if (BoyerMooer(textNotMatched, pattern[i])) {
                return i;
            }
        }
        return -1;
    }

    public static bool BoyerMoore2D(string[] text, string[] pattern) {
        // function to search for pattern in text (2 dimensional array) using Boyer-Moore algorithm
        int m = pattern.Length;
        int n = text.Length;

        int s = 0; // s is position of start character of pattern relative to text
        while (s <= (n - m)) {
            int j = m - 1;
            while (j >= 0 && BoyerMooer(text[s + j], pattern[j])) { // while characters of pattern and text are same
                j--;
            }

            if (j < 0) { // pattern found
                return true;
            }
            else { // character mismatch
                s += Math.Max(1, j - FindMatches(pattern, text[s + j]));
            }
        }

        return false;
    }
}
```

```

public static float imageBM_Algorithm(String imageText, String imagePattern) {
    // function to search for pattern in text using Knuth-Morris-Pratt algorithm
    // text and pattern get from image
    // imageText is path to image text
    // imagePattern is path to image pattern

    string[] selectedPattern = new string[30];
    byte[][] l1;
    byte[][] l2;
    try {
        /** ***** Pattern **** */
        // Load the image
        using (Image image = Image.FromFile(imagePattern)) {
            // image to binary
            byte[][] imageBytes2d = Util.ImageToByteArray(image);
            l1 = imageBytes2d;
            imageBytes2d = Util.twoDPattern(imageBytes2d, image.Width, image.Height);

            // using (StreamWriter writer = new StreamWriter("2dPattern.txt")) {
            //     for (int y = 0; y < image.Height; y++) {
            //         for (int x = 0; x < (image.Width / 8) * 8; x++) {
            //             writer.Write(imageBytes2d[y][x]);
            //         }
            //         writer.WriteLine(); // Move to the next line after each row
            //     }
            // }

            // binary to ascii
            string[] base64String = Util.bitsToString2D(imageBytes2d);
            for (int i = 0; i < 30; i++)
            {
                selectedPattern[i] = base64String[i];
            }

            // write into file (testing)
            // File.WriteAllText("asciiPattern.txt", selectedPattern);
            // Console.WriteLine("Selected Pattern: " + selectedPattern);
        }
    }
}

```

```


    /**
     * **** Text ****
     */
    using (Image image = Image.FromFile(imageText)) {
        // image to binary
        byte[][] imageBytes2d = Util.ImageToByteArray(image);
        l2 = imageBytes2d;

        // using (StreamWriter writer = new StreamWriter("2dText.txt")) {
        //     for (int y = 0; y < image.Height; y++) {
        //         for (int x = 0; x < image.Width; x++) {
        //             writer.Write(imageBytes2d[y][x]);
        //         }
        //         writer.WriteLine(); // Move to the next line after each row
        //     }
        // }

        /**
         * **** KMP ****
         */
        // search for pattern in text
        for (int i = 0; i < (image.Width % 8) + 1; i++) {
            String[] mtx = Util.getText(imageBytes2d, i);
            if (BoyerMoore2D(mtx, selectedPattern)) {
                return 1;
            }
        }
        return HammingDistance.CalculateHammingDistance(l1, l2);
    }

    catch (Exception ex) { // handle error
        Console.WriteLine($"Error: {ex.Message}");
        return 0;
    }
}


```

Gambar 11. Tangkapan Layar BM_2D.cs

- KMP.cs

```
private static void ComputeLPSArray(string pattern, int m, int[] lps) {  
    // function to compute the longest proper prefix which is also suffix  
    int length = 0;  
    lps[0] = 0;  
  
    int i = 1;  
    while (i < m) {  
        if (pattern[i] == pattern[length]) {  
            length++;  
            lps[i] = length;  
            i++;  
        }  
        else {  
            if (length != 0) {  
                length = lps[length - 1];  
            }  
            else {  
                lps[i] = 0;  
                i++;  
            }  
        }  
    }  
}
```

```
public static bool KMP_Algorithm(string text, string pattern) {
    // function to search for pattern in text using Knuth-Morris-Pratt algorithm
    int m = pattern.Length;
    int n = text.Length;

    int[] lps = new int[m];
    ComputeLPSArray(pattern, m, lps);

    int i = 0; // index for text
    int j = 0; // index for pattern

    while (i < n) {
        if (pattern[j] == text[i]) {
            i++;
            j++;
        }

        if (j == m) {
            return true;
        }

        else if (i < n && pattern[j] != text[i]) {
            if (j != 0) {
                j = lps[j - 1];
            }
            else {
                i++;
            }
        }
    }
    return false;
}
```

```

public static float imageKMP_Algorithm(String imageText, String imagePattern) {
    // function to search for pattern in text using Knuth-Morris-Pratt algorithm
    // text and pattern get from image
    // imageText is path to image text
    // imagePattern is path to image pattern

    string selectedPattern = "";
    byte[][] l1;
    byte[][] l2;
    try {
        /** ***** Pattern **** */
        // Load the image
        using (Image image = Image.FromFile(imagePattern)) {
            // image to binary
            byte[][] imageBytes2d = Util.ImageToByteArray(image);
            l1 = imageBytes2d;
            imageBytes2d = Util.twoDPattern(imageBytes2d, image.Width, image.Height);

            using (StreamWriter writer = new StreamWriter("2dPattern.txt")) {
                for (int y = 0; y < image.Height; y++) {
                    for (int x = 0; x < (image.Width / 8) * 8; x++) {
                        writer.WriteLine(imageBytes2d[y][x]);
                    }
                    writer.WriteLine(); // Move to the next line after each row
                }
            }

            // get 1d pattern (take from middle of image)
            byte[] imageBytes1d = Util.get1DPattern(imageBytes2d, image.Width, image.Height);
            using (StreamWriter writer = new StreamWriter("1dPattern.txt")) {
                for (int y = 0; y < imageBytes1d.Length; y++) {
                    writer.WriteLine(imageBytes1d[y]); // Move to the next line after each row

                }
            }
        }
    }
}
// binary to ascii

```

```

string base64String = Util.bitsToString(imageBytes1d);
selectedPattern = base64String;

// write into file (testing)
// File.WriteAllText("asciiPattern.txt", selectedPattern);
// Console.WriteLine("Selected Pattern: " + selectedPattern);
}

/** ***** Text **** */
using (Image image = Image.FromFile(imageText)) {
    // image to binary
    byte[][] imageBytes2d = Util.ImageToByteArray(image);
    l2 = imageBytes2d;

    using (StreamWriter writer = new StreamWriter("2dText.txt")) {
        for (int y = 0; y < image.Height; y++) {
            for (int x = 0; x < image.Width; x++) {
                writer.Write(imageBytes2d[y][x]);
            }
            writer.WriteLine(); // Move to the next line after each row
        }
    }

    // change 2d to 1d
    byte[] imageBytes1d = Util.arrayOfStringToString(imageBytes2d, image.Width, image.Height);
    using (StreamWriter writer = new StreamWriter("1dText.txt")) {
        for (int y = 0; y < imageBytes1d.Length; y++) {
            writer.Write(imageBytes1d[y]); // Move to the next line after each row
        }
    }

    // binary to ascii
    string base64String = Convert.ToBase64String(imageBytes1d);
}

```

```

    /**
     * ***** KMP *****
     */
    // search for pattern in text
    for (int i = 0; i < (image.Width % 8) + 1; i++) {
        String[] mtx = Util.getText(imageBytes2d, i);
        for (int j = 0; j < mtx.Length; j++) {
            if (KMP_Algorithm(mtx[j], selectedPattern)) {
                return 1;
            }
        }
    }
    return HammingDistance.CalculateHammingDistance(l1, l2);
}

catch (Exception ex) { // handle error
    // Console.WriteLine($"Error: {ex.Message}");
    return 0;
}
}

```

Gambar 12. Tangkapan Layar KMP.cs

- **HammingDistance**

```

using System;
using System.Collections.Generic;

class HammingDistance {
    public static float CalculateHammingDistance(byte[][] str1, byte[][] str2) {
        // function to calculate hamming distance between two matrices of characters
        int numberOfCharacters = str1.Length * str1.Length;
        float distance = 0;
        for (int i = 0; i < str1.Length; i++) {
            for (int j = 0; j < str1.Length; j++) {
                if (str1[i][j] != str2[i][j]) {
                    distance += 1;
                }
            }
        }
        return (float)(1 - (distance / numberOfCharacters));
    }
}

```

Gambar 13. Tangkapan Layar HammingDistance.cs

- Util.cs

```
class Util {  
    public static byte[] arrayToString(byte[][] mtx, int w, int h) {  
        // function array of string (2D) to string (1D)  
        byte[] result = new byte[h * w];  
        for (int y = 0; y < h; y++) {  
            for (int x = 0; x < w; x++) {  
                result[y * w + x] = mtx[y][x];  
            }  
        }  
        return result;  
    }  
  
    public static byte[][] ImageToByteArray(Image img) {  
        // function to convert image into binary  
        // string imagePath = pic ;  
        int THRESHOLD = 127;  
  
        Bitmap image = new Bitmap(img);  
  
        int width = image.Width;  
        int height = image.Height;  
  
        byte[][] binaryPixels = new byte[height][];  
        for (int i = 0; i < height; i++) {  
            binaryPixels[i] = new byte[width];  
        }  
  
        for (int y = 0; y < height; y++) {  
            for (int x = 0; x < width; x++) {  
                Color pixelColor = image.GetPixel(x, y);  
                int grayValue = (int)(0.299 * pixelColor.R + 0.587 * pixelColor.G + 0.114 * pixelColor.B);  
  
                if (grayValue > THRESHOLD) {  
                    binaryPixels[y][x] = 1;  
                }  
                else {  
                    binaryPixels[y][x] = 0;  
                }  
            }  
        }  
        return binaryPixels;  
    }  
}
```

```
public static String[] getText(byte[][] mtx, int idx) {
    // function to binary text into ASCII
    int width = 8 * (mtx[0].Length / 8);
    String[] result = new string[mtx.Length];

    byte[] line = new byte[width / 8];
    byte temp = 0;
    for (int i = 0; i < mtx.Length; i++) {
        for (int j = 0; j < width; j++) {
            temp <<= 1;
            temp += mtx[i][j + idx];
            if (j % 8 == 7) {
                line[j / 8] = temp;
                temp = 0;
            }
        }
        string s = System.Text.Encoding.Latin1.GetString(line);
        result[i] = s;
    }
    return result;
}
```

```
public static byte[] get1DPattern(byte[][] mtx, int w, int h) {
    // function to convert image into binary with cutting border
    // get the middle part of the image
    byte[] result = new byte[(w / 8) * 8];

    for (int x = 0; x < (w / 8) * 8; x++) {
        result[x] = mtx[(h / 2)][x];
    }
    return result;
}

public static byte[][] twoDPattern(byte[][] mtx, int w, int h) {
    // cut the image to make the width multiple of 8
    byte[][] result = new byte[h][];
    for (int y = 0; y < h; y++) {
        result[y] = new byte[(w / 8) * 8];
        for (int x = 0; x < (w / 8) * 8; x++) {
            result[y][x] = mtx[y][x];
        }
    }
    return result;
}
```

```
public static string[] bitsToString2D(byte[][] mtx) {
    // get the middle 30*32 matrix of binary
    // the convert it to array of string
    string[] result = new string[mtx.Length];
    int border = mtx.Length - 30;
    int up = border / 2;

    for (int i = 0; i < 30; i++) {
        byte[] list = new byte[mtx[0].Length];

        for (int j = 0; j < mtx[0].Length; j++) {
            list[j] = mtx[i + up][j];
        }

        string s = bitsToString(list);
        result[i] = s;
    }
    return result;
}
```

```
public static string bitsToString(byte[] list) {
    // convert binary to string
    // then cut the middle 4 bytes
    byte[] line = new byte[list.Length / 8];
    byte temp = 0;
    for (int i = 0; i < list.Length; i++) {
        temp <<= 1;
        temp += list[i];
        if (i % 8 == 7) {
            line[i / 8] = temp;
            temp = 0;
        }
    }

    int border = line.Length - 4;
    int left = border / 2;

    byte[] finalLine = new byte[4];
    for (byte i = 0; i < 4; i++) {
        finalLine[i] = line[i + left];
    }
    return System.Text.Encoding.Latin1.GetString(finalLine);
}
```

Gambar 14. Tangkapan Layar Util.cs

4.2. Tata Cara Menggunakan Aplikasi

Untuk menjalankan program ini, ada beberapa aplikasi (*software*) yang diperlukan, seperti Sqlite3, C#, dan WPF dengan *extension* sebagai berikut:

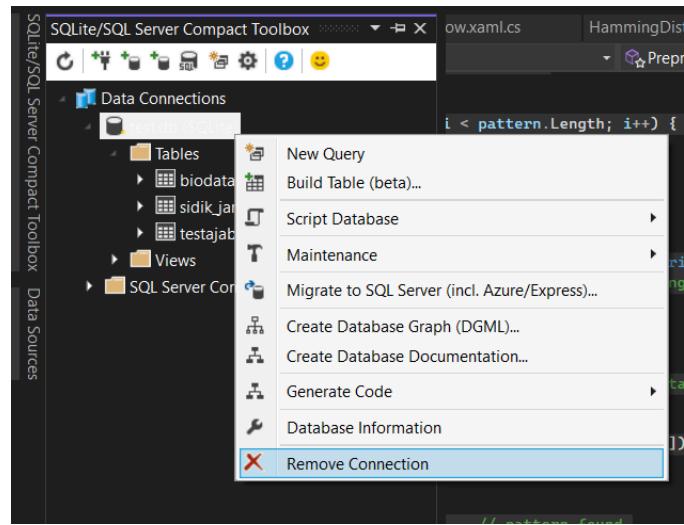
1. System.Drawing
2. System.IO.Compression
3. System.Text.Encodings.Code
4. System.Text.Encoding.CodePages

Untuk menjalankan aplikasi dapat mengikuti langkah-langkah sebagai berikut :

1. Lakukan *clone* repository Tubes3_Bang-Udah-Capek-Bang.
2. Pastikan perangkat sudah mempunyai Sqlite3, C#, dan WPF.
3. Masukkan *project* yang ada yaitu *Fingerprint Detection* dengan lokasi folder Tubes3_Bang-Udah-Capek-Bang/src/ui/Fingerprint Detection/Fingerprint Detection.
4. Buka WPF dan *install* semua ekstensi yang diperlukan dan *apply*.
5. *Build* program dengan menekan tombol segitiga hijau pada bagian menu.
6. Masukkan gambar berformat .BMP pada kontainer sidik jari.
7. Pilih Algoritma KMP atau BM dengan *toggle button* yang ada.
8. Mulai pencarian dengan menekan tombol “SEARCH”.
9. Anda dapat melakukan pencarian lagi dengan menekan kontainer “Inserted Fingerprint” lalu pilih gambar yang ingin dicari.

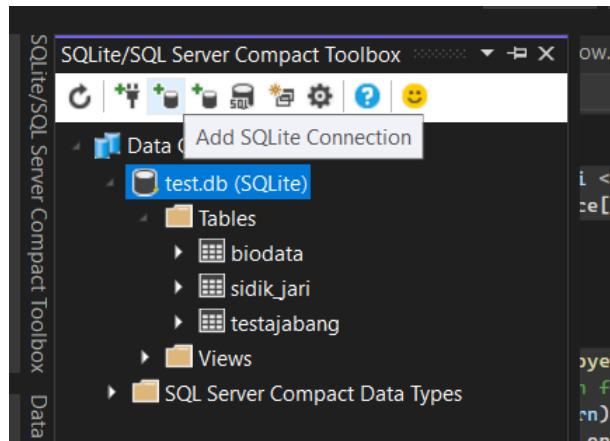
Jika anda ingin menambah dataset dapat mengikuti langkah-langkah sebagai berikut:

1. Hapus database test.db pada folder Tubes3_Bang-Udah-Capek-Bang/src/ui/Fingerprint Detection/Fingerprint Detection.
2. Masukkan gambar dataset yang ingin ditambah ke dalam folder Tubes3_Bang-Udah-Capek-Bang/test. Gambar **HARUS** berformat fingerprintXXXX dengan XXXX merupakan digit dimulai dari 6001.
3. Pindah ke direktori Tubes3_Bang-Udah-Capek-Bang/database.
4. Lakukan *seeding* ulang dataset dengan menjalankan file db_uhuy dan file db_ahay secara berurutan.
5. Masuk ke WPF dan masuk ke menu SQLite Server Compact Toolbox. Klik kanan file test.db dan pilih *Remove Connection*.



Gambar 15. Remove Connection

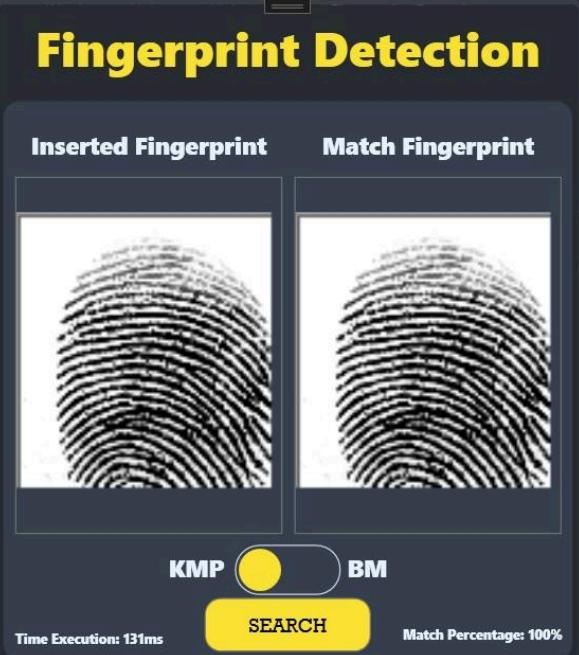
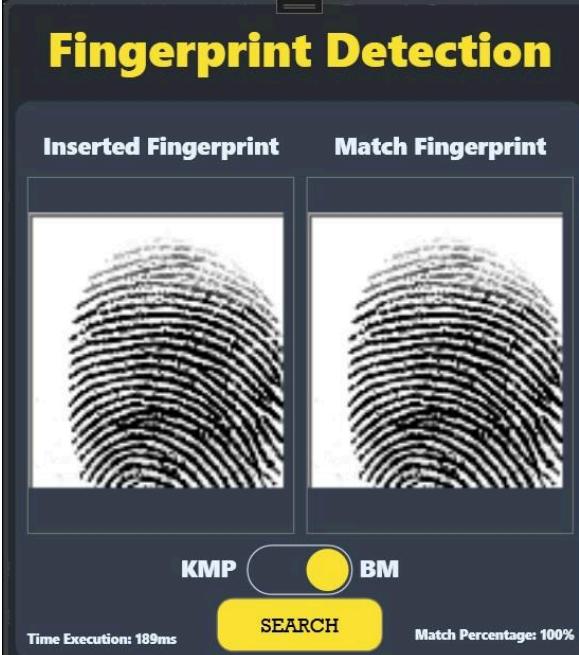
6. Lakukan koneksi ulang dengan menekan tombol Add SQLite Connection



Gambar 16. Koneksi Ulang ke Database

7. Pilih Browse dan masukkan file database di direktori:
Tubes3_Bang-Udah-Capek-Bang/database/test.db

4.3. Hasil Pengujian

Knuth-Morris-Pratt	Boyer-Moore																																																												
 <p>Fingerprint Detection</p> <p>Inserted Fingerprint Match Fingerprint</p> <p>KMP BM</p> <p>SEARCH</p> <p>Match Percentage: 100%</p> <p>Time Execution: 131ms</p> <p>Biodata</p> <table><tbody><tr><td>NIK</td><td>:</td><td>7528482580953277</td></tr><tr><td>Name</td><td>:</td><td>Pang357u S170mpu1</td></tr><tr><td>DOB</td><td>:</td><td>Pontianak</td></tr><tr><td>Gender</td><td>:</td><td>Male</td></tr><tr><td>Blood Type</td><td>:</td><td>O</td></tr><tr><td>Address</td><td>:</td><td>62 John Wall Terrace</td></tr><tr><td>Religion</td><td>:</td><td>Hindu</td></tr><tr><td>Status</td><td>:</td><td>Belum Menikah</td></tr><tr><td>Job</td><td>:</td><td>Safety Technician IV</td></tr><tr><td>Citizen</td><td>:</td><td>Ukraine</td></tr></tbody></table>	NIK	:	7528482580953277	Name	:	Pang357u S170mpu1	DOB	:	Pontianak	Gender	:	Male	Blood Type	:	O	Address	:	62 John Wall Terrace	Religion	:	Hindu	Status	:	Belum Menikah	Job	:	Safety Technician IV	Citizen	:	Ukraine	 <p>Fingerprint Detection</p> <p>Inserted Fingerprint Match Fingerprint</p> <p>KMP BM</p> <p>SEARCH</p> <p>Match Percentage: 100%</p> <p>Time Execution: 189ms</p> <p>Biodata</p> <table><tbody><tr><td>NIK</td><td>:</td><td>7528482580953277</td></tr><tr><td>Name</td><td>:</td><td>Pang357u S170mpu1</td></tr><tr><td>DOB</td><td>:</td><td>Pontianak</td></tr><tr><td>Gender</td><td>:</td><td>Male</td></tr><tr><td>Blood Type</td><td>:</td><td>O</td></tr><tr><td>Address</td><td>:</td><td>62 John Wall Terrace</td></tr><tr><td>Religion</td><td>:</td><td>Hindu</td></tr><tr><td>Status</td><td>:</td><td>Belum Menikah</td></tr><tr><td>Job</td><td>:</td><td>Safety Technician IV</td></tr><tr><td>Citizen</td><td>:</td><td>Ukraine</td></tr></tbody></table>	NIK	:	7528482580953277	Name	:	Pang357u S170mpu1	DOB	:	Pontianak	Gender	:	Male	Blood Type	:	O	Address	:	62 John Wall Terrace	Religion	:	Hindu	Status	:	Belum Menikah	Job	:	Safety Technician IV	Citizen	:	Ukraine
NIK	:	7528482580953277																																																											
Name	:	Pang357u S170mpu1																																																											
DOB	:	Pontianak																																																											
Gender	:	Male																																																											
Blood Type	:	O																																																											
Address	:	62 John Wall Terrace																																																											
Religion	:	Hindu																																																											
Status	:	Belum Menikah																																																											
Job	:	Safety Technician IV																																																											
Citizen	:	Ukraine																																																											
NIK	:	7528482580953277																																																											
Name	:	Pang357u S170mpu1																																																											
DOB	:	Pontianak																																																											
Gender	:	Male																																																											
Blood Type	:	O																																																											
Address	:	62 John Wall Terrace																																																											
Religion	:	Hindu																																																											
Status	:	Belum Menikah																																																											
Job	:	Safety Technician IV																																																											
Citizen	:	Ukraine																																																											
<p>Gambar 17. Test-Case 1 KMP</p> <p>Persentase Kecocokan: 100%</p>	<p>Gambar 18. Test-Case 1 BM</p> <p>Persentase kecocokan: 100%</p>																																																												

Fingerprint Detection

Inserted Fingerprint	Match Fingerprint
	

KMP BM

Time Execution: 36074ms
SEARCH
Match Percentage: 100%

Biodata

NIK	: 3101509659997022
Name	: G4wati Nu6roho
DOB	: Pangkal Pinang
Gender	: Male
Blood Type	: AB
Address	: 70938 Gateway Street
Religion	: Christian
Status	: Belum Menikah
Job	: Payment Adjustment Coordinator
Citizen	: Ukraine

Fingerprint Detection

Inserted Fingerprint	Match Fingerprint
	

KMP BM

Time Execution: 31374ms
SEARCH
Match Percentage: 100%

Biodata

NIK	: 3101509659997022
Name	: G4wati Nu6roho
DOB	: Pangkal Pinang
Gender	: Male
Blood Type	: AB
Address	: 70938 Gateway Street
Religion	: Christian
Status	: Belum Menikah
Job	: Payment Adjustment Coordinator
Citizen	: Ukraine

Gambar 19. Test-Case 2 KMP

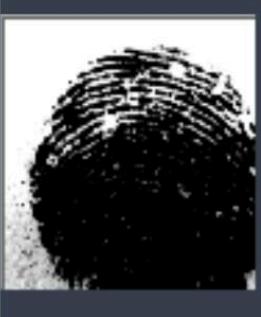
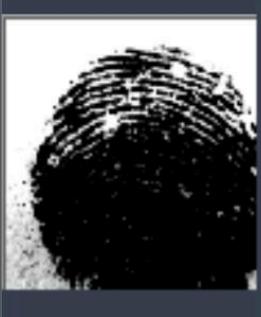
Persentase Kecocokan: 100%

Gambar 20. Test-Case 2 BM

Persentase kecocokan: 100%

Fingerprint Detection

Inserted Fingerprint
Match Fingerprint

KMP
 BM

Time Execution: 12338ms
SEARCH
Match Percentage: 100%

Biodata

NIK	:	5730602939668446
Name	:	Harjas4 Widi4s7u7i
DOB	:	Padang Sidempuan
Gender	:	Female
Blood Type : B		
Address	:	7997 Upham Plaza
Religion	:	Konghucu
Status	:	Belum Menikah
Job	:	Computer Systems Analyst I
Citizen	:	China

Fingerprint Detection

Inserted Fingerprint
Match Fingerprint




KMP
 BM

Time Execution: 11999ms
SEARCH
Match Percentage: 100%

Biodata

NIK	:	5730602939668446
Name	:	Harjas4 Widi4s7u7i
DOB	:	Padang Sidempuan
Gender	:	Female
Blood Type : B		
Address	:	7997 Upham Plaza
Religion	:	Konghucu
Status	:	Belum Menikah
Job	:	Computer Systems Analyst I
Citizen	:	China

Gambar 21. Test-Case 3 KMP

Persentase Kecocokan: 100%

Gambar 22. Test-Case 3 BM

Persentase kecocokan: 100%

Fingerprint Detection

Inserted Fingerprint
Match Fingerprint

KMP
 BM

Time Execution: 22222ms

SEARCH
Match Percentage: 100%

Biodata

NIK	: 1484634108108127
Name	: Latik4 Yun14r
DOB	: Pontianak
Gender	: Female
Blood Type	: B
Address	: 28409 Spaight Pass
Religion	: Buddha
Status	: Menikah
Job	: Paralegal
Citizen	: China

Gambar 23. Test-Case 4 KMP
Persentase Kecocokan: 100%

Fingerprint Detection

Inserted Fingerprint
Match Fingerprint

KMP
 BM

Time Execution: 17049ms

SEARCH
Match Percentage: 100%

Biodata

NIK	: 1484634108108127
Name	: Latik4 Yun14r
DOB	: Pontianak
Gender	: Female
Blood Type	: B
Address	: 28409 Spaight Pass
Religion	: Buddha
Status	: Menikah
Job	: Paralegal
Citizen	: China

Gambar 24. Test-Case 4 BM
Persentase kecocokan: 100%

Fingerprint Detection

Inserted Fingerprint
Match Fingerprint

KMP
 BM

Time Execution: 55961ms
SEARCH
Match Percentage: 94,58%

Biodata

NIK	: 4189881208437802
Name	: Hani Kusm4wa71
DOB	: Serang
Gender	: Female
Blood Type : B	
Address	: 8714 Dahle Parkway
Religion	: Chatolic
Status	: Cerai
Job	: Office Assistant IV
Citizen	: South Korea

Gambar 25. Test-Case 5 KMP

Persentase Kecocokan: 94,58%

Fingerprint Detection

Inserted Fingerprint
Match Fingerprint

KMP
 BM

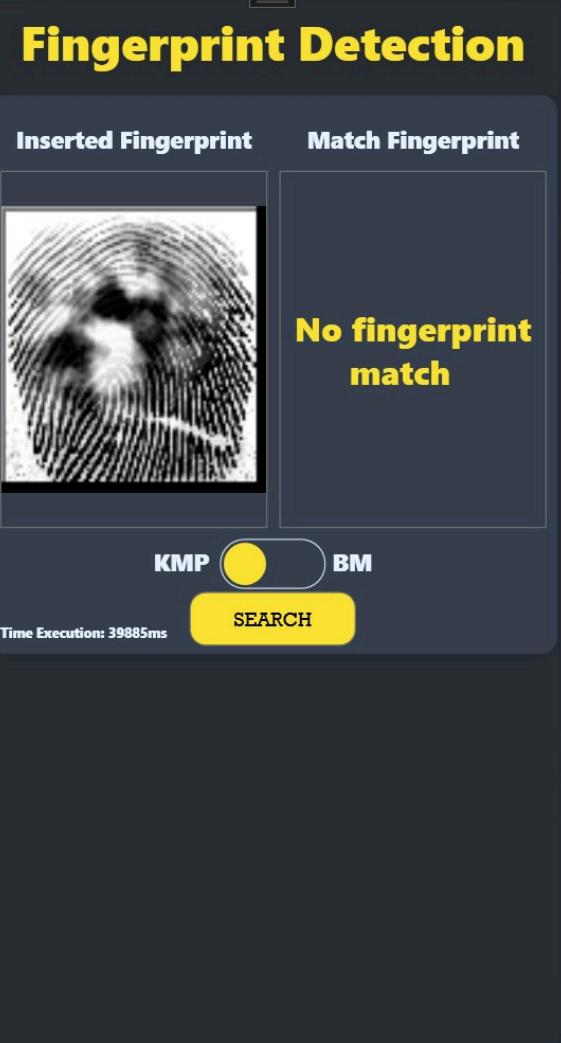
C:\Users\ASUS\Downloads\
SEARCH
Match Percentage: 94,58%

Biodata

NIK	: 1484634108108127
Name	: Latik4 Yun14r
DOB	: Pontianak
Gender	: Female
Blood Type : B	
Address	: 28409 Spaight Pass
Religion	: Buddha
Status	: Menikah
Job	: Paralegal
Citizen	: China

Gambar 26. Test-Case 5 BM

Persentase kecocokan: 94,58%

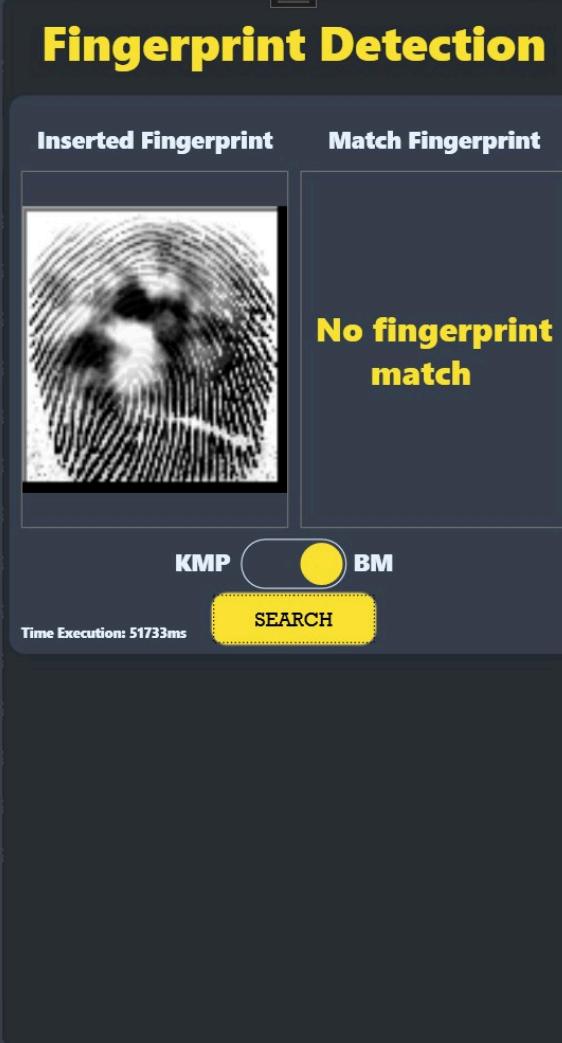


Fingerprint Detection

Inserted Fingerprint	Match Fingerprint
	No fingerprint match

KMP BM
SEARCH

Time Execution: 39885ms



Fingerprint Detection

Inserted Fingerprint	Match Fingerprint
	No fingerprint match

KMP BM
SEARCH

Time Execution: 51733ms

Gambar 27. *Test-Case 6 KMP*

Persentase Kecocokan:%

Gambar 28. *Test-Case 6 BM*

Persentase kecocokan:%

Tabel 3. Hasil Pengujian

4.4. Analisis Pengujian

Hasil pengujian menunjukkan bahwa rata-rata pencarian menggunakan algoritma Boyer-Moore selesai lebih cepat dibandingkan pencarian menggunakan algoritma Knuth-Morris-Pratt. Algoritma Boyer-Moore memiliki keunggulan saat anggota karakter dalam gambar *text* memiliki jumlah yang besar. Sedangkan, Algoritma Knuth-Morris-Pratt keunggulan saat anggota karakter dalam gambar *text* memiliki jumlah yang kecil. Pada program ini, karakter yang digunakan adalah karakter ASCII. Karakter ASCII berjumlah 256 sehingga membuat jumlah anggota karakter cenderung besar. Hasil pengujian berarti sudah tepat karena jumlah anggota karakter pada gambar *text* cenderung besar.

Hasil pengujian juga menunjukkan ada persentase yang tidak 100%. Hal ini bisa terjadi karena ada input gambar yang tidak identik dengan gambar yang ada di basis data, namun input gambar tersebut memiliki kemiripan yang cukup tinggi dengan gambar yang ada di basis data. Gambar yang tidak identik ini merepresentasikan keadaan nyata yaitu, pengambilan gambar pada sidik jari yang sama belum tentu menghasilkan gambar yang identik.

BAB 5

KESIMPULAN, SARAN, TANGGAPAN DAN REFLEKSI

5.1. Kesimpulan

Aplikasi ini berhasil melakukan pencarian biodata dari basis data dengan masukan gambar sidik jari. Gambar sidik jari dari input berhasil dicocokan dengan gambar sidik jari di basis data menggunakan algoritma *string matching* Knuth-Morris-Pratt atau Boyer-Moore. Selain pencocokan secara absolut, aplikasi juga berhasil mencocokan gambar berdasarkan kemiripan dengan menggunakan Hamming Distance. Dari hasil pengujian, ditemukan bahwa pencarian menggunakan algoritma Boyer-Moore memiliki waktu yang lebih singkat dibandingkan dengan pencarian menggunakan algoritma Knuth-Morris-Pratt. Hal ini terjadi karena pada komparasi menggunakan karakter ASCII yang berjumlah 256. Jumlah anggota karakter pada gambar *text* akan cenderung besar. Hal ini sesuai dengan algoritma Boyer-Moore yang memiliki keunggulan saat anggota karakter pada gambar *text* berjumlah besar.

5.2. Saran

Hal yang bisa dilakukan untuk dapat meningkatkan hasil ataupun kinerja dari aplikasi ini:

1. Melakukan *multithread* agar proses pencarian menjadi lebih cepat.

5.3. Tanggapan

Tugas besar ini membingungkan kelompok kami. Pertama, penggunaan C# yang *enviroment*-nya cukup unik (alias aneh). Kedua, perubahan tipe biner ke tipe ASCII untuk mempercepat program bisa kurang efektif. Hal ini disebabkan oleh ASCII yang sebenarnya hanya berukuran 7-bit. Tapi, perubahan biner ke ASCII dilakukan dengan mengubah 8 biner ke ASCII.

5.4. Refleksi

Selama penggeraan tugas besar ini kami mendapatkan beberapa pelajaran. Selain dari semakin meningkatnya pemahaman mengenai algoritma *pattern matching* menggunakan KMP dan BM, kami juga belajar menggunakan bahasa baru, yaitu C#. Secara keseluruhan kami mendapatkan insight dalam banyak hal.

LAMPIRAN

Repository Github

Link repository : https://github.com/NgokNgok04/Tubes3_Bang-Udah-Capek-Bang.git.

DAFTAR PUSTAKA

<https://alaygenerator.blogspot.com/> (Diakses pada 20 Mei 2024).

<https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2019-2020/Modul-Praktikum-NLP-Regex.pdf> (Diakses pada 20 Mei 2024).

<https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Pencocokan-string-2021.pdf> (Diakses pada 20 Mei 2024).

<https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2022-2023/String-Matching-dengan-Regex-2019.pdf> (Diakses pada 20 Mei 2024).

<https://learn.microsoft.com/en-us/dotnet/desktop/wpf/overview/?view=netdesktop-8.0> (Diakses pada 18 Mei 2024)

<https://www.kaggle.com/datasets/ruizgara/socofing> (Diakses pada 20 Mei 2024).