

IF2211 Strategi Algoritma
**Pemanfaatan Pattern Matching dalam Membangun Sistem Deteksi Individu Berbasis
Biometrik Melalui Citra Sidik Jari**

Laporan Tugas Besar 3

Disusun untuk memenuhi tugas mata kuliah Strategi Algoritma
pada Semester 2 (dua) Tahun Akademik 2023/2024



Oleh

Maulvi Ziadinda Maulana	13522122
Shabrina Maharani	13522134
Ahmad Rafi Maliki	13522137

Kelompok mamama

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

DAFTAR ISI

DAFTAR ISI	2
BAB 1 DESKRIPSI TUGAS	3
BAB 2 LANDASAN TEORI	5
2.1 Pattern Matching	5
2.2 Algoritma KMP	6
2.3 Algoritma BM	8
2.4 Algoritma Regex	9
2.5 Teknik Pengukuran Persentase Kemiripan	13
2.6 Aplikasi Dekstop	15
BAB 3 ANALISIS PEMECAHAN MASALAH	16
3.1 Langkah-Langkah Pemecahan Masalah	16
3.2 Proses Penyelesaian Solusi dengan Algoritma KMP dan BM	18
3.2.1 KMP	18
3.2.2 BM	20
3.3 Fitur Fungsional dan Arsitektur Aplikasi Dekstop	22
3.4 Contoh Ilustrasi Kasus	23
BAB 4 IMPLEMENTASI DAN PENGUJIAN	25
4.1 Spesifikasi Teknis Program	25
4.2 Source Code	40
4.3 Penjelasan Tata Cara Penggunaan Program	49
4.4 Hasil Pengujian	51
4.2.1 Home Page	51
4.2.2 About Us Page	51
4.2.3 Pengujian KMP	51
4.2.4 Pengujian Boyer Moore	54
4.5 Analisis Hasil Pengujian	58
BAB 5 KESIMPULAN DAN SARAN	60
5.1 Kesimpulan	60
5.2 Saran	60
5.3 Refleksi	60
DAFTAR PUSTAKA	62

BAB 1 DESKRIPSI TUGAS

Di era digital ini, keamanan dan akses data menjadi semakin krusial. Kemajuan teknologi membuka peluang bagi metode identifikasi yang lebih canggih dan praktis. Beberapa metode umum seperti kata sandi atau PIN memiliki kelemahan, seperti rentan terhadap pelupukan atau pencurian. Sebagai solusi alternatif, penggunaan teknologi biometrik semakin populer, dengan salah satu teknologi utamanya adalah identifikasi sidik jari. Setiap sidik jari memiliki pola unik yang tidak dapat ditiru, menjadikannya pilihan yang cocok untuk identifikasi individu. Teknik penting dalam identifikasi sidik jari adalah pattern matching, yang memungkinkan sistem untuk mencocokkan pola sidik jari dengan database. Algoritma yang umum digunakan adalah Bozorth dan Boyer-Moore, yang memungkinkan pengenalan sidik jari secara cepat dan akurat, bahkan jika sidik jari yang ditangkap tidak sempurna. Dengan menggabungkan teknologi identifikasi sidik jari dan pattern matching, dapat dibangun sistem identifikasi biometrik yang aman, handal, dan mudah digunakan. Sistem ini dapat diaplikasikan dalam berbagai bidang, seperti kontrol akses, absensi karyawan, dan verifikasi identitas dalam transaksi keuangan.



Gambar 1.1 Ilustrasi Identifikasi Sidik Jari

(Sumber: <https://mistar.id/wp-content/uploads/2023/12/Sidik-Jari.jpg>)

Tugas Besar ini meminta implementasi sistem identifikasi individu berbasis biometrik menggunakan sidik jari. Sistem ini akan menggunakan algoritma KMP, BM, dan Regular Expression untuk mencocokkan sidik jari dengan biodata yang mungkin rusak. Program akan terhubung dengan basis data SQL yang telah mencocokkan citra sidik jari dengan individu, dan akan menerima input citra sidik jari untuk dicocokkan. Jika ada kecocokan di atas batas tertentu, program akan menampilkan biodata individu tersebut; jika tidak, akan muncul pesan bahwa sidik jari tidak dikenali. Program juga memiliki antarmuka yang ramah pengguna dan memungkinkan pengguna memilih algoritma yang ingin digunakan. Biodata yang ditampilkan akan dibenarkan menggunakan Regex dan dicari kecocokan terdekat menggunakan KMP atau BM. Beberapa fitur tambahan juga dapat ditambahkan untuk meningkatkan kegunaan program.

Spesifikasi Tugas Besar

Pada Tugas Besar ini, buatlah sebuah sistem yang dapat melakukan identifikasi individu berbasis biometrik dengan menggunakan sidik jari dengan detail sebagai berikut.

1. Sistem dibangun dalam bahasa C# yang mengimplementasikan algoritma KMP, BM, dan Regular Expression dalam mencocokkan sidik jari dengan biodata yang berpotensi rusak.
2. Program dapat memiliki basis data SQL yang telah mencocokkan berkas citra sidik jari yang telah ada dengan seorang pribadi. Basis data yang digunakan dibebaskan asalkan bukan No-SQL (sebagai contoh, MySQL, PostgreSQL, SQLite).
3. Program dapat menerima masukan sebuah citra sidik jari yang ingin dicocokkan. Apabila citra tersebut memiliki kecocokan di atas batas tertentu (silakan lakukan tuning nilai yang tepat) dengan citra yang sudah ada, maka tunjukkan biodata orang tersebut. Apabila di bawah nilai yang telah ditentukan tersebut, memunculkan pesan bahwa sidik jari tidak dikenali.
4. Program memiliki keluaran yang minimal mengandung seluruh data yang terdapat pada contoh antarmuka pada bagian penggunaan program
5. Pengguna dapat memilih algoritma yang ingin digunakan antara KMP atau BM.
6. Biodata yang ditampilkan harus biodata yang memiliki nama yang benar (gunakan Regex untuk memperbaiki nama yang rusak dan gunakan KMP atau BM untuk mencari orang yang paling sesuai).
7. Program memiliki antarmuka yang user-friendly. Anda juga dapat menambahkan fitur lain untuk menunjang program yang Anda buat (unsur kreativitas).

Bonus

Bagian ini hanya boleh dikerjakan apabila spesifikasi wajib dari Tugas Besar telah berhasil dipenuhi. Anda tidak diharuskan untuk mengerjakan keseluruhan bonus, tetapi semakin banyak bonus yang dikerjakan, maka akan semakin banyak tambahan nilai yang diperoleh.

1. Lakukan enkripsi terhadap semua data pada basis data
Data yang terdapat dalam KTP merupakan data yang privat dan tidak seharusnya mudah diakses orang yang tidak berwenang. Buatlah sebuah skema enkripsi-dekripsi semua data pada basis data sehingga ketika ada pihak yang melakukan query SQL secara langsung ke basis data, tidak ada data berarti yang didapatkan (dilarang menggunakan library, implementasikan sendiri, semakin aman semakin tinggi nilai bonus ini).
2. Membuat video
Buatlah sebuah video mengenai program yang dibuat dan algoritma yang digunakan pada tugas besar ini. Video yang dibuat harus memiliki audio dan menampilkan wajah dari setiap anggota kelompok. Video tersebut kemudian diupload ke YouTube. Beberapa contoh video tubes tahun-tahun sebelumnya dapat dilihat di YouTube dengan menggunakan kata kunci “Tubes Stima”, “Tugas Besar Stima”, “Strategi Algoritma”, dan lain-lain.

BAB 2 LANDASAN TEORI

2.1 Pattern Matching

Pattern matching adalah metode dalam pemrograman yang memungkinkan pengembang untuk mencari dan mengidentifikasi keberadaan urutan karakter, kata, atau pola dalam data yang lebih besar. Metode ini sangat berharga dalam berbagai aplikasi seperti pengolahan teks, analisis data, dan pengembangan perangkat lunak, karena memfasilitasi pencarian dan manipulasi informasi secara efisien. Dalam konteks pengolahan teks, misalnya, pattern matching memungkinkan sistem untuk menemukan frasa atau kata kunci tertentu dalam dokumen besar, mirip dengan cara kerja fitur pencarian pada mesin pencari atau dalam pengeditan teks.

Konsep dasar dari pattern matching melibatkan dua komponen utama: teks (T) yang merupakan string panjang dan pola (P) yang merupakan string yang lebih pendek untuk dicari dalam teks. Tujuannya adalah untuk menemukan lokasi pertama dalam teks di mana terdapat kecocokan lengkap dengan pola. Sebagai contoh, jika teksnya adalah "the rain in Spain stays mainly on the plain" dan pola yang dicari adalah "main", pattern matching bertujuan untuk menemukan keberadaan "main" pertama kali dalam teks tersebut. Teknik ini memanfaatkan algoritma efisien untuk mempercepat pencarian dan mengurangi kompleksitas komputasi, terutama pada teks yang sangat panjang.

```
Input =      ba aab ab
Pattern = ****ba*****ab
Output : true

Input =      ba aab ab
Pattern = a*ab
Output : false

Input =      ba aab ab
Pattern = ba*a?
Output : true
```

Gambar 2.1 Ilustrasi Pattern Matching

(Sumber: <https://mistar.id/wp-content/uploads/2023/12/Sidik-Jari.jpg>)

Dalam konteks string, konsep-konsep penting yang terlibat dalam pattern matching termasuk pengertian prefix dan suffix dari string. Sebuah prefix adalah substring yang dimulai dari karakter pertama dan berakhir pada indeks k tertentu dalam string, sedangkan suffix adalah substring yang dimulai dari indeks k dan berlangsung hingga akhir string. Pemahaman yang baik tentang prefix dan suffix sangat membantu dalam memahami dan mengimplementasikan algoritma pattern matching yang efektif,

seperti algoritma Knuth-Morris-Pratt atau Boyer-Moore, yang memanfaatkan pola prefix dan suffix untuk mengoptimalkan pencarian.

Dalam pengembangan perangkat lunak, pattern matching sering digunakan untuk memvalidasi input data. Misalnya, pengembang bisa menggunakan regular expressions, sebuah teknik untuk pattern matching, untuk memverifikasi format data seperti alamat email, nomor telepon, atau kode pos. Ini membantu dalam menegakkan konsistensi data dan mencegah kesalahan input. Selain itu, pattern matching juga berperan dalam pemrograman fungsional, di mana pola tertentu dalam data dapat memicu fungsi tertentu, memungkinkan kode yang lebih modular dan mudah diuji.

2.2 Algoritma KMP

Algoritma Knuth-Morris-Pratt (KMP) merupakan salah satu algoritma yang digunakan dalam pencarian string atau pattern matching. Algoritma ini dirancang untuk mencari keberadaan sebuah string atau pola dalam teks yang lebih panjang dengan cara yang lebih cerdas dibandingkan dengan pendekatan brute force. Algoritma KMP melakukan prosesnya dengan menghindari pengulangan pencocokan yang tidak perlu dengan memanfaatkan informasi yang sudah didapatkan dari pencocokan sebelumnya. Hal ini dicapai melalui penggunaan **fungsi pinggiran**, yang juga dikenal sebagai fungsi kegagalan, yang menentukan seberapa jauh pola harus digeser saat terjadi ketidakcocokan.

Fungsi pinggiran (border function) merupakan inti dari algoritma KMP. Fungsi ini memungkinkan algoritma untuk mempercepat proses pencarian dengan mengidentifikasi bagian terpanjang dari prefix pola yang juga merupakan suffix. Dengan informasi ini, KMP dapat melanjutkan pencocokan dari titik yang paling mungkin menghasilkan kesesuaian tanpa harus memulai kembali dari awal pola. Misalnya, jika mismatch terjadi pada posisi j dari pola, algoritma dapat menggunakan fungsi pinggiran untuk menemukan nilai j baru sehingga meminimalisir bagian dari teks yang perlu dicocokkan kembali.

2.3 Algoritma BM

Algoritma Boyer-Moore adalah salah satu metode paling efisien untuk pencocokan string, yang terutama digunakan untuk mencari keberadaan suatu pola dalam teks. Diciptakan oleh Robert S. Boyer dan J Strother Moore, algoritma ini mendapatkan popularitas karena kecepatannya yang tinggi dalam aplikasi praktis dibandingkan dengan algoritma brute force dan KMP. Ide dasar dari algoritma Boyer-Moore (BM) sebetulnya hampir sama dengan algoritma brute-force. Algoritma ini merupakan improvisasi dari KMP (Knuth-Morris-Pratt), algoritma yang menghilangkan proses pengecekan ulang oleh algoritma brute-force dengan menggunakan bantuan tabel berisi langkah perpindahan pattern yang harus dilakukan ke kanan ketika menjumpai ketidakcocokan karakter. Pada algoritma BM, pencocokan dilakukan dengan mengubah arah pencocokan, yaitu mencocokkan karakter dimulai dari kanan (karakter terakhir pada pattern) ke kiri. Sementara itu pattern tetap digeser ke kanan. Jika menemui ketidakcocokan, pattern akan dipindahkan ke kanan dan jika mungkin, membuat kecocokan dengan karakter pada teks yang sedang dilakukan pengecekan. Boyer-Moore memanfaatkan dua teknik utama: teknik cermin (*looking-glass technique*) dan teknik lompat karakter (*character-jump technique*), yang keduanya berkontribusi untuk mengurangi jumlah perbandingan yang diperlukan dalam pencocokan string.

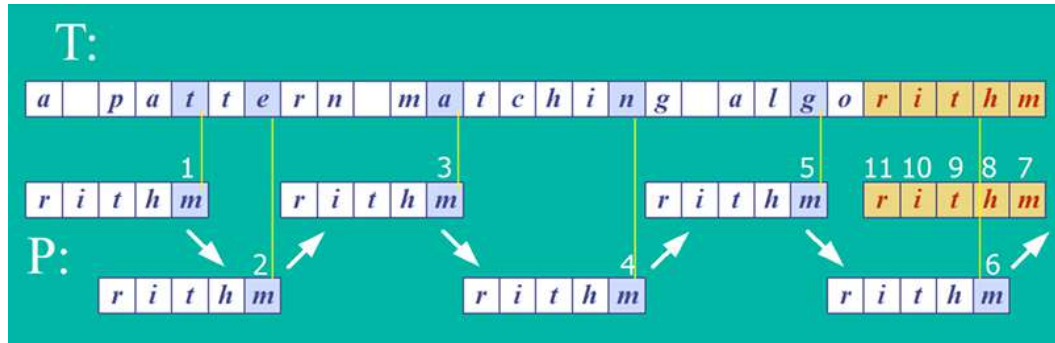
Teknik cermin dalam algoritma Boyer-Moore mengubah cara pencocokan karakter dalam pola dengan teks. Daripada memulai dari karakter pertama pola, algoritma ini memulai pencocokan dari karakter terakhir pola. Pendekatan ini memungkinkan deteksi ketidakcocokan lebih awal dalam proses pencocokan, sehingga dapat dengan cepat menggeser pola tanpa perlu mengevaluasi setiap karakter dari awal. Ini secara signifikan mengurangi waktu yang dibutuhkan untuk mencari pola dalam teks yang panjang.

Teknik lompat karakter mengambil keuntungan dari informasi tentang kegagalan pencocokan karakter. Ketika terjadi ketidakcocokan karakter, algoritma tidak sekadar menggeser pola satu karakter ke kanan seperti dalam brute force. Sebaliknya, Boyer-Moore menggunakan fungsi 'last occurrence' untuk menentukan seberapa jauh pola harus digeser. Fungsi ini memberikan posisi terakhir dari setiap karakter dalam pola, yang memungkinkan pola digeser lebih jauh, melewati bagian teks yang tidak mungkin cocok.

Ada tiga kasus yang dapat terjadi ketika terjadi ketidakcocokan, yang masing-masing menentukan strategi pergeseran pola:

1. Jika karakter yang tidak cocok di teks (x) terdapat di suatu tempat dalam pola, algoritma mencoba menggeser pola ke kanan untuk menyelaraskan kemunculan terakhir x dalam pola dengan karakter ini di teks.

2. Jika x terdapat dalam pola, tetapi pergeseran ke kemunculan terakhir tidak memungkinkan (karena akan melewati batas teks), maka pola digeser satu karakter ke kanan.
3. Jika kedua kasus sebelumnya tidak berlaku (x tidak ada dalam pola), maka pola digeser sepenuhnya untuk memulai pencocokan baru tepat setelah karakter x ini di teks.



Gambar 2.3 Ilustrasi Proses Algoritma BM

(Sumber: [Pencocokan-string-2021.pdf \(itb.ac.id\)](https://itb.ac.id/pencocokan-string-2021.pdf))

Boyer-Moore sangat efisien terutama ketika alfabet dari teks yang dicari cukup besar, seperti teks bahasa Inggris. Hal ini dikarenakan potensi pergeseran pola yang lebih besar saat ketidakcocokan terjadi, yang mengurangi jumlah perbandingan yang diperlukan. Namun, algoritma ini bisa kurang efektif jika alfabet kecil, seperti dalam kasus data biner, karena peluang untuk lompatan besar lebih sedikit.

Dalam analisis kasus terburuk, Boyer-Moore masih memiliki efisiensi yang signifikan dibandingkan brute force, dengan kompleksitas waktu berjalan $O(nm + A)$, di mana n adalah panjang teks dan m adalah panjang pola, dan A adalah ukuran alfabet. Meskipun dalam kasus tertentu bisa mendekati brute force, pada umumnya Boyer-Moore jauh lebih cepat untuk pencarian dalam teks bahasa Inggris dan aplikasi serupa. Ini menjadikan Boyer-Moore pilihan yang sangat baik untuk sistem yang memproses volume data teks yang besar atau untuk aplikasi yang memerlukan pencarian string yang efisien dan cepat.

2.4 Algoritma Regex

Regular Expression (Regex) merupakan notasi standar yang mendeskripsikan suatu pola (pattern) berupa urutan karakter atau string. Regex digunakan untuk pencocokan string (string matching) dengan efisien. Regex sudah menjadi standar yang tersebar di semua tools dan bahasa pemrograman sehingga penting untuk dipelajari. Regex memungkinkan untuk mencari pola dalam string dengan cara yang sangat efisien dan fleksibel. Berikut adalah beberapa konsep dasar dan penggunaan regex.

1. Metakarakter

Regex menggunakan sejumlah metakarakter yang memiliki fungsi khusus. Misalnya, karakter titik (.) akan cocok karakter apapun, kecuali baris baru (newline : \n). Jika diperlukan metakarakter (seperti .) sebagai bagian pola regex, gunakanlah backslash '\'. Karakter seperti *, +, dan ? digunakan untuk menentukan berapa kali suatu bagian dari pola harus muncul. Metakarakter tersebut membantu dalam menciptakan pola yang fleksibel dan dinamis.

2. Character Classes

Character classes memungkinkan kita untuk mencocokkan karakter dari set tertentu. Tabel berikut dapat memperlihatkan berbagai konstruksi regex untuk membentuk character class.

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

Berikut adalah penjelasan lebih lanjut mengenai pengertian setiap jenis class.

1. Simple Class

Bentuk paling sederhana dari character class adalah karakter yang berada dalam kurung siku (row pertama). Contohnya regex [abc]a akan cocok dengan "aa", "ba", dan "ca".

2. Negasi

Negasi dalam regex digunakan untuk mengecualikan karakter tertentu dari pencocokan. Dengan menambahkan simbol ^ di awal set karakter dalam tanda kurung siku, seperti [^bcr], regex akan mencocokkan setiap karakter yang bukan "b", "c", atau "r". Ini berarti, di konteks ini, kata seperti "hat" yang tidak mengandung karakter "b", "c", atau "r" akan cocok dengan pola ini.

3. Ranges

Ranges memungkinkan pencocokan karakter dalam rentang tertentu, yang ditentukan dengan menggunakan tanda hubung - antara dua

karakter. Misalnya, [a-e] akan mencocokkan karakter apa pun dari "a" sampai "e", dan [1-5] mencocokkan angka dari 1 sampai 5. Rentang ini sangat berguna untuk menentukan satu set karakter yang berurutan. Negasi juga dapat diterapkan ke dalam ranges. Dengan menggunakan simbol ^ dalam karakter kelas bisa mengecualikan rentang tertentu dalam pencocokan, sehingga memungkinkan pencocokan untuk semua karakter di luar rentang tersebut.

4. Unions

Unions memungkinkan kombinasi dari beberapa range atau karakter. Sebagai contoh, [a-e[v-z]] menandakan gabungan dari huruf "a" sampai "e" dengan "v" sampai "z". Ini memperluas jangkauan pencocokan ke grup karakter yang lebih besar dan lebih spesifik. Selain itu, penting untuk membedakan antara penggunaan kurung siku dan kurung biasa dalam regex. Kurung siku [] digunakan untuk membuat kelas karakter, di mana karakter di dalamnya bisa cocok dengan salah satu dari elemen yang disertakan. Sementara itu, kurung biasa () menangani grup sebagai unit dan digunakan untuk mengelompokkan karakter atau sub-pattern sebagai satu kesatuan yang harus cocok persis seperti yang tertulis. Sebagai contoh, [ade] akan cocok dengan "a", "d", atau "e", sedangkan (ade) hanya akan cocok dengan string "ade".

3. Predefined Character Classes

Regex juga memiliki predefined character classes yang sering digunakan untuk membuat regex lebih mudah dibaca dan mengurangi kesalahan.

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

4. Quantifiers

Quantifiers digunakan untuk menentukan berapa kali elemen dalam pola harus muncul. Terdapat beberapa quantifier dalam regex seperti pada tabel berikut.

Construct	Arti
X?	X muncul satu atau tidak sama sekali
X*	X muncul bol atau banyak
X+	X muncul satu atau banyak
x{n}	X muncul tepat n kali
x{n,}	X muncul setidaknya n kali
x{n,m}	X muncul antara n sampai m kali

Sebagai contoh, berikut regex yang dapat menangkap kata “rekan” atau “rekans” (s bisa ada atau tidak). s? artinya karakter s dapat muncul atau tidak.

5. Boundary Matchers

Boundary matchers digunakan untuk mencari pola yang muncul di suatu posisi tertentu.

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

Berikut adalah cheatsheet dari keseluruhan konsep regex di atas.

Cheat Sheet	
Character classes	
.	any character except newline
\w \d \s	word, digit, whitespace
\W \D \S	not word, digit, whitespace
[abc]	any of a, b, or c
[^abc]	not a, b, or c
[a-g]	character between a & g
Anchors	
^abc\$	start / end of the string
\b	word boundary
Escaped characters	
\. * \\	escaped special characters
\t \n \r	tab, linefeed, carriage return
\u00A9	unicode escaped ©
Groups & Lookaround	
(abc)	capture group
\1	backreference to group #1
(?:abc)	non-capturing group
(?=abc)	positive lookahead
(?!abc)	negative lookahead
Quantifiers & Alternation	
a* a+ a?	0 or more, 1 or more, 0 or 1
a{5} a{2,}	exactly five, two or more
a{1,3}	between one & three
a+? a{2,}?	match as few as possible
ab cd	match ab or cd

Gambar 2.4 Cheat Sheet Regex

(Sumber: [Modul Praktikum NLP : Regex - Google Dokumen](#))

2.5 Teknik Pengukuran Persentase Kemiripan

Teknik pengukuran persentase kemiripan pada tugas ini digunakan untuk menentukan seberapa mirip suatu sidik jari yang menjadi masukan dengan sidik jari yang terdapat dalam basis data. Persentase kemiripan ini adalah ukuran numerik yang menunjukkan tingkat kesamaan antara dua data gambar, yang diekspresikan dalam bentuk persentase. Dalam tugas ini, persentase kemiripan digunakan untuk memastikan keakuratan sistem dalam mengenali individu berdasarkan pola sidik jari mereka.

Metode pengukuran persentase kemiripan yang dapat digunakan dalam tugas besar ini diantaranya adalah algoritma Hamming Distance, Levenshtein Distance, ataupun Longest Common Subsequence (LCS).

Hamming Distance adalah sebuah algoritma yang digunakan untuk mengukur jarak antara dua string dengan panjang yang sama. Jarak Hamming didefinisikan sebagai jumlah posisi di mana simbol yang bersesuaian dalam dua string tersebut berbeda. Algoritma ini bekerja dengan membandingkan setiap pasangan karakter pada posisi yang sama dalam kedua string dan menghitung jumlah pasangan yang berbeda. Hamming Distance hanya dapat digunakan jika kedua string memiliki panjang yang sama.

Levenshtein Distance, juga dikenal sebagai Edit Distance, mengukur jumlah minimal operasi yang diperlukan untuk mengubah satu string menjadi string lainnya. Operasi yang diizinkan adalah penyisipan, penghapusan, atau substitusi satu karakter. Algoritma ini bekerja dengan membangun matriks yang memetakan setiap kemungkinan perubahan dari satu string ke string lainnya dan kemudian menghitung jalur dengan biaya terendah melalui matriks tersebut. Matriks ini diisi dengan cara dinamis, di mana setiap sel dalam matriks mengandung nilai Levenshtein Distance antara substring dari kedua string yang sedang dibandingkan.

Longest Common Subsequence (LCS) adalah algoritma yang digunakan untuk menemukan urutan subsekuens terpanjang yang terdapat dalam dua string yang diberikan, di mana urutan karakter dalam subsekuens tersebut harus sama dengan urutan kemunculannya dalam string aslinya, tetapi tidak perlu berurutan secara langsung. Algoritma LCS bekerja dengan membangun matriks dinamis di mana setiap sel mewakili panjang LCS dari substring yang sedang dibandingkan. Dari nilai-nilai dalam matriks ini, kita dapat merekonstruksi LCS itu sendiri.

Ketiga algoritma ini memiliki pendekatan yang berbeda untuk mengukur kemiripan antara dua string. Hamming Distance lebih sederhana dan cepat tetapi hanya berlaku untuk string dengan panjang yang sama. Levenshtein Distance lebih fleksibel dan dapat menangani string dengan panjang berbeda, namun memiliki kompleksitas yang lebih tinggi. Longest Common Subsequence membantu menemukan urutan karakter yang sama dalam urutan yang benar, yang dapat berguna dalam analisis teks dan pemrosesan bahasa alami.

2.6 Aplikasi Dekstop

Pada Tugas Besar ini, aplikasi desktop dikembangkan menggunakan framework Avalonia dengan backend yang ditulis dalam bahasa C#. Aplikasi ini dirancang khusus untuk memudahkan pengguna dalam pencarian dan identifikasi sidik jari. Pengguna dapat memasukkan citra sidik jari melalui antarmuka yang intuitif, di mana aplikasi akan membandingkannya dengan data yang tersimpan dalam basis data SQL relasional. Selain itu, aplikasi menyediakan opsi untuk memilih algoritma pencarian sidik jari yang diinginkan, antara Knuth-Morris-Pratt (KMP) atau Boyer-Moore (BM), yang keduanya diintegrasikan untuk meningkatkan efektivitas pencarian.

Dalam penggunaannya, setelah sidik jari diunggah melalui pemilihan button “Pilih Citra”, pengguna dapat menentukan algoritma pencarian dan memulai proses pencarian dengan menekan tombol 'Search'. Aplikasi kemudian akan mencari sidik jari yang paling mirip yang tersimpan dalam basis data, menampilkan hasil yang relevan bersama dengan detail dari hasil pencarian seperti tingkat kemiripan dalam persentase dan waktu eksekusi pencarian serta list biodata.

BAB 3 ANALISIS PEMECAHAN MASALAH

3.1 Langkah-Langkah Pemecahan Masalah

Permasalahan utama yang harus diselesaikan dalam Tugas Besar 3 ini adalah menerima gambar sidik jari dan menampilkan data individu terkait melalui sistem atau program berbasis biometrik menggunakan sidik jari (dalam bentuk gambar) dimana program tersebut juga harus mampu menangani data korup berupa bahasa alay dengan kombinasi huruf besar-kecil, penggunaan angka, dan penyingkatan. Sistem ini harus diimplementasikan dengan mengintegrasikan algoritma Knuth-Morris-Pratt (KMP) dan Boyer-Moore (BM), dibantu oleh Regular Expression (Regex) untuk mencocokkan sidik jari serta memperbaiki nama yang rusak dalam biodata. Berikut adalah langkah-langkah yang digunakan untuk menyelesaikan permasalahan tersebut.

1. Pada `SolverPageView.axml.cs`, terdapat metode `InitializeComponent()`, yang berfungsi menghubungkan komponen frontend dengan logika backend. Di sini, kami menggunakan variabel seperti `_imageDisplay`, `_option1`, `_option2`, dsb. untuk menghubungkan komponen frontend dan backend menggunakan `FindControl`. Pada metode `ImageInputButton_Click`, kita akan mengambil data gambar dari file yang dipilih. Metode ini akan memeriksa semua file yang ada di komputer menggunakan perulangan `var files`. Hanya file dengan format tertentu seperti PNG, JPG, JPEG, dan BMP yang diperbolehkan sebagai input. Jika `files.Count > 0`, yang berarti pengguna telah memilih suatu file, program akan membuka file tersebut, mengubahnya menjadi bitmap, lalu menggunakan bitmap tersebut sebagai sumber (`Source`) untuk `ImageDisplay` agar gambar yang dipilih oleh User dapat ditampilkan pada aplikasi. Gambar yang dipilih tersebut akan melalui pemrosesan lebih lanjut karena tanpa pemrosesan tersebut gambar yang berasal dari input user bisa saja memiliki ukuran ataupun resolusi yang berbeda-beda sehingga memungkinkan perubahan gambar menjadi *string pattern* tidak sesuai dan sulit ditemukan kecocokannya dengan data yang terdapat dalam *database*.



Gambar 3.1.1 Ilustrasi Gambar awal yang belum mengalami pemrosesan

2. Untuk menangani perbedaan ukuran dan resolusi gambar sidik jari tersebut, kami melakukan pemrosesan gambar yang diawali dengan mengubah gambar menjadi citra grayscale. Dalam proses ini, kami menggunakan library OpenCV, dimulai

dengan memuat gambar menggunakan CvInvoke, kemudian mengubahnya menjadi grayscale dengan CvtColor. Pengubahan gambar menjadi citra grayscale didasari oleh pertimbangan beberapa keuntungan dengan menggunakan citra grayscale. Gambar grayscale hanya memiliki satu channel dibandingkan dengan gambar berwarna yang memiliki tiga channel (RGB). Dalam gambar berwarna, setiap piksel diwakili oleh tiga nilai yang masing-masing menunjukkan intensitas merah, hijau, dan biru. Sedangkan dalam gambar grayscale, setiap piksel hanya diwakili oleh satu nilai yang menunjukkan intensitas cahaya, membuat proses lebih sederhana dan efisien. Gambar grayscale memiliki ukuran piksel 1 byte, sedangkan gambar berwarna bisa mencapai 3 byte per piksel. Selain itu, komponen penting dalam pencocokan sidik jari adalah pola dari sidik jari itu sendiri, bukan warnanya. Warna menjadi hal yang tidak relevan untuk menganalisis sebuah sidik jari. Gambar dengan citra grayscale juga membutuhkan lebih sedikit ruang penyimpanan dibandingkan dengan gambar berwarna. Setelah diubah menjadi grayscale, gambar kemudian diubah ukurannya menjadi 96x103 piksel untuk memastikan konsistensi dalam perbandingan. Selanjutnya, gambar grayscale yang masih memiliki berbagai tingkat keabuan diubah menjadi gambar biner dimana gambar tersebut hanya akan merepresentasikan warna hitam dan putih. Dengan gambar yang telah diubah menjadi biner dan berukuran 96x103 piksel, kita dapat mempertahankan pola sidik jari dan memudahkan proses pencocokan string. Konversi ini memastikan bahwa hanya pola sidik jari yang dianalisis, tanpa terganggu oleh variasi warna atau tingkat keabuan.



Gambar 3.1.2 Ilustrasi Gambar yang sudah mengalami pemrosesan

3. Setelah melalui perubahan citra gambar, kami memanfaatkan fungsi `GetEncoding("iso-8859-1")` untuk mengubah gambar menjadi pola dalam bentuk karakter ASCII 256 karakter, sesuai dengan spesifikasi yang diberikan. Encoding ISO-8859-1, juga dikenal sebagai Latin-1, merupakan salah satu standar encoding yang mampu merepresentasikan setiap byte sebagai karakter unik dalam rentang 0-255. Ini berarti bahwa setiap byte dari data gambar yang telah diproses dapat secara langsung diterjemahkan menjadi satu karakter dalam encoding ini. Prinsip dari Encoding tersebut adalah dengan mengubah gambar menjadi karakter ASCII, di mana setiap karakter diwakili oleh 8 bit. Kemudian, kami juga menggunakan fungsi `GetString` untuk mendapatkan string dari data tersebut. Dengan demikian, gambar yang menjadi input user telah bertransformasi dari gambar menjadi

sebuah string. Setelah gambar berada dalam bentuk string, program melanjutkan tahap pencarian dengan mengolah informasi dari SearchButton. Pada tahap ini, gambar dalam bentuk string akan melalui tahap pencarian sidik jari yang cocok dengan data sidik jari yang terdapat dalam database dengan menggunakan algoritma KMP dan BM sesuai dengan masukan algoritma yang diinginkan oleh user. Gambar yang digunakan pada proses pattern matching kedua algoritma tersebut adalah gambar sidik jari penuh berukuran $m \times n$ pixel yang diambil sebesar 60 pixel setiap kali proses pencocokan data dengan pertimbangan agar variasi pattern lebih banyak dan pencocokan menjadi lebih akurat dan presisi.

4. Proses pencarian dengan pencocokan string menggunakan algoritma KMP, BM, serta penggunaan Regex untuk menyelesaikan permasalahan data yang korup akan dijelaskan lebih lanjut pada bagian selanjutnya yaitu bagian 3.2 Proses Penyelesaian Solusi dengan Algoritma KMP dan BM. Hasil dari proses penyelesaian solusi menggunakan algoritma KMP dan BM adalah passing informasi ke sebuah objek Result yang berisi informasi berupa Image (gambar sidik jari yang ditemukan dari database setelah proses pencarian menggunakan algoritma KMP dan BM yang sudah diubah dari bentuk string menjadi bentuk gambar kembali), Text (biodata dari pemilik sidik jari terkait), TimeDiff (lama waktu eksekusi program), dan Percentage (persentase kemiripan gambar dari input user dengan gambar yang telah dipilih dari database yang dihitung dengan menggunakan teknik *Hamming Distance*). Jika persentase kurang dari delapan puluh persen, maka pencarian dikatakan tidak ditemukan dengan pertimbangan angka di bawah delapan puluh lima persen sudah memiliki terlalu banyak perbedaan sehingga kadar kemiripan nya tidak relevan.
5. Informasi dari objek Result ini kemudian di-*passing* ke objek-objek UI terkait dan ditampilkan kepada user melalui ResultWindow. Dengan demikian, pengguna dapat melihat hasil pencarian berupa gambar sidik jari yang cocok, informasi pemilik sidik jari, waktu eksekusi, dan persentase kemiripan secara langsung di antarmuka aplikasi.

3.2 Proses Penyelesaian Solusi dengan Algoritma KMP dan BM

Setelah berada diubah ke dalam bentuk string, pola string tersebut akan diolah oleh program sesuai dengan algoritma yang dipilih user untuk menyelesaikan pencarian biodata yang sesuai dengan gambar sidik jari dari user. Berikut adalah penyelesaian persoalan menggunakan algoritma KMP dan BM yang di dalamnya sudah disertai penggunaan konsep Regex.

3.2.1 KMP

Berikut adalah langkah penyelesaian persoalan menggunakan algoritma KMP.

1. Dalam penerapan algoritma KMP pada persoalan ini, metode yang akan menjadi penerapan algoritma KMP dari pencarian biodata yang sesuai adalah metode match, yang menerima parameter berupa pola string hasil perubahan dari gambar dan pola string dari gambar yang terdapat pada database. Langkah pertama dalam algoritma KMP adalah menemukan border array atau prefix table untuk pola yang ingin dicari (misalnya border array dinotasikan dengan P). Border array ini membantu dalam menentukan seberapa jauh program dapat menggeser pola ke kanan saat terjadi ketidakcocokan. Selain itu, untuk memastikan pergeseran dilakukan dengan tepat, diperlukan KMP Border Function yang dinotasikan dengan $b(k)$, yang didefinisikan sebagai panjang dari prefix terbesar dari $P[0..k]$ yang juga merupakan suffix dari $P[1..k]$. Border Function diimplementasikan dengan menghitung jumlah pergeseran, yaitu pengurangan antara panjang border array dengan panjang prefix yang sama dengan suffix-nya. Pada program ini, perhitungan KMP Border Function diimplementasikan oleh metode `computeBorder`.
2. Tujuan utama dari langkah kedua ini adalah membangun border array, dimulai dengan menginisialisasi border array (misalnya b) untuk pola P . Nilai awal $b[0]$ diatur menjadi 0 karena tidak ada prefix yang juga suffix untuk satu karakter. Kemudian, program akan melakukan iterasi melalui pola dengan dua indeks. Indeks pertama (misalnya i) bergerak maju melalui pola, sementara indeks kedua (misalnya j) digunakan untuk melacak panjang dari prefix yang cocok. Jika karakter pada posisi i dalam pola cocok dengan karakter pada posisi j , maka $b[i]$ diatur menjadi $j + 1$, dan kedua indeks i dan j ditingkatkan (masing-masing indeks ditambah dengan angka 1). Jika terjadi ketidakcocokan dan j lebih besar dari 0, program akan menggunakan nilai $b[j-1]$ untuk memperbarui j . Jika ketidakcocokan terjadi dan j adalah 0, program akan mengatur $b[i]$ menjadi 0 dan melanjutkan pemeriksaan dengan indeks i .
3. Implementasi dalam metode match dimulai dengan menginisialisasi teks dan pola menjadi array karakter. Panjang teks (n) dan panjang pola (m) dihitung, dan border array ditentukan dengan memanggil metode `computeBorder`. Algoritma kemudian melakukan pencarian pola dengan menggunakan dua indeks, i untuk teks dan j untuk pola. Jika karakter dari pola dan teks cocok, kedua indeks ditingkatkan. Jika indeks j mencapai akhir pola ($m - 1$), pola ditemukan dalam teks dan metode mengembalikan

nilai true. Jika karakter tidak cocok dan j lebih besar dari 0, nilai j diperbarui menggunakan border array. Jika karakter tidak cocok dan j adalah 0, indeks i ditingkatkan untuk memeriksa karakter teks berikutnya.

4. Metode findMatch menggunakan algoritma KMP untuk mencari pola dalam teks dengan menginisialisasi waktu mulai dan mempersiapkan daftar sidik jari dari database. Pola yang dicari diproses dalam set ukuran tertentu dan hasil pencarian diperbarui berdasarkan kecocokan dengan pola saat ini. Jika ditemukan kecocokan dalam sidik jari yang terdapat dalam database, program akan mencoba mencocokkan nama yang ditemukan dengan biodata dalam database menggunakan regex. Jika ditemukan kecocokan, gambar hasil dikonversi kembali dari pola string ke dalam bentuk gambar, lalu informasi hasil diperbarui dengan melakukan passing parameter dari hasil nama yang ditemukan dan persentase kecocokan.

3.2.2 BM

Berikut adalah langkah penyelesaian persoalan menggunakan algoritma KMP.

1. Proses pencarian dimulai dengan metode findMatch, yang menginisialisasi waktu mulai dan mempersiapkan daftar sidik jari dari database. Pola yang dicari dipecah menjadi beberapa bagian dengan ukuran tertentu, dan hasil pencarian diperbarui berdasarkan kecocokan dengan pola saat ini. Pada setiap iterasi, sidik jari yang cocok dengan bagian pola disimpan, sementara yang tidak cocok diabaikan. Proses ini berlangsung hingga ditemukan kecocokan atau tidak ada lagi sidik jari yang tersisa untuk diperiksa. Jika ditemukan kecocokan dalam sidik jari yang ada dalam database, program mencoba mencocokkan nama yang ditemukan dengan biodata dalam database menggunakan regex. Jika ada kecocokan, gambar hasil dikonversi kembali dari pola string ke bentuk gambar, dan informasi hasil diperbarui dengan parameter nama yang ditemukan dan persentase kecocokan.
2. Algoritma Boyer-Moore diimplementasikan dalam method Bmmatch, dimulai dengan membangun tabel last occurrence, yang berisi posisi terakhir kemunculan setiap karakter dalam pola. Panjang teks dan pola dihitung, dan indeks awal i diatur pada posisi terakhir pola. Jika panjang pola lebih besar dari panjang teks, tidak ada kecocokan yang mungkin, dan metode mengembalikan nilai false. Pencarian dilakukan dengan dua indeks: i untuk teks dan j untuk pola. Jika karakter pada posisi i dalam teks

cocok dengan karakter pada posisi j dalam pola, kedua indeks akan berkurang. Jika indeks j mencapai 0, pola ditemukan dalam teks dan metode mengembalikan nilai true. Jika karakter tidak cocok, nilai i diperbarui berdasarkan tabel last, yaitu dengan menambahkan panjang pola dikurangi nilai minimum antara j dan 1 ditambah posisi terakhir kemunculan karakter teks pada indeks i . Proses ini berulang hingga indeks i melebihi panjang teks atau pola ditemukan.

3. Metode BuildLast membangun tabel last occurrence dengan menginisialisasi semua nilai menjadi -1 untuk setiap karakter dalam set karakter ASCII 8-bit. Setiap karakter dalam pola kemudian diperbarui dengan posisi terakhir kemunculannya. Tabel last occurrence ini digunakan dalam metode BmMatch untuk menentukan seberapa jauh pola harus digeser saat terjadi ketidakcocokan, sehingga memungkinkan algoritma untuk mengabaikan sebagian besar teks yang tidak relevan dan mempercepat proses pencarian pola. Implementasi lengkap dari algoritma Boyer-Moore dalam metode findMatch dan BmMatch memastikan bahwa pola dicocokkan dengan efisien, meminimalkan jumlah perbandingan yang diperlukan dengan menggunakan informasi dari tabel last untuk menggeser pola secara optimal. Dengan demikian, algoritma Boyer-Moore mampu mencari pola dalam teks dengan kecepatan dan efisiensi yang tinggi.

Masing-masing algoritma KMP dan BM dibantu oleh penggunaan konsep regex dalam pencarian nama yang sesuai. Metode match pada kelas MyRegex menerima dua parameter string, text1 dan text2, di mana text2 adalah nama yang dicari dan text1 adalah nama yang ada di database. Metode ini memulai dengan membuat objek Regex menggunakan pola yang diperoleh dari metode getPattern. Pola ini dirancang untuk mencocokkan nama dengan mempertimbangkan berbagai bentuk penulisan, termasuk penggunaan karakter alternatif dan variasi huruf besar/kecil.

Metode getPattern membangun pola regex dengan menambahkan berbagai kemungkinan karakter yang dapat menggantikan setiap karakter dalam text2. Setiap karakter dalam text2 diubah menjadi bagian dari pola yang mempertimbangkan huruf besar dan kecil, pengganti karakter "alay", dan spasi. Misalnya, karakter 'o' dalam text2 dapat dicocokkan dengan 'o', 'O', atau '0'. Selain itu, karakter vokal dalam text2 diikuti dengan tanda tanya dalam pola regex untuk memungkinkan kecocokan opsional. Hasil dari metode getPattern adalah sebuah pola regex yang mampu menangkap berbagai variasi penulisan nama.

Setelah pola regex dibentuk, metode match menggunakan pola ini untuk mencocokkan dengan text1 menggunakan metode Match dari kelas Regex. Jika kecocokan ditemukan, metode ini mengembalikan nilai true, menunjukkan bahwa text2 ditemukan dalam text1. Penggunaan regex dalam algoritma KMP dan BM memungkinkan pencarian yang lebih fleksibel dan akurat dengan mempertimbangkan variasi penulisan dan karakter alternatif, yang sangat penting dalam konteks pencarian nama yang mungkin memiliki banyak variasi.

3.3 Fitur Fungsional dan Arsitektur Aplikasi Dekstop

Fitur fungsional dan arsitektur aplikasi desktop yang kami buat menggunakan bahasa pemrograman C# dan framework Avalonia dengan arsitektur MVVM (Model-View-ViewModel). C# dipilih karena kemampuannya yang kuat dalam pengembangan aplikasi desktop, performa tinggi, serta dukungan yang baik dari komunitas dan berbagai alat pengembangan. Avalonia merupakan framework UI lintas platform yang unggul karena mendukung berbagai sistem operasi seperti Windows, macOS, dan Linux, serta memungkinkan pengembangan antarmuka yang modern dan responsif. Arsitektur MVVM digunakan dalam aplikasi ini untuk memisahkan logika backend dan tampilan, memudahkan pengelolaan kode, dan meningkatkan keterbacaan serta maintainability.

Dalam Avalonia, antarmuka pengguna (UI) didefinisikan menggunakan Avalonia XAML (AXML) yang memisahkan visualisasi dari logika aplikasi. Konsep pembuatannya terdiri dari tiga komponen utama sesuai dengan arsitektur MVVM: Model, View, dan ViewModel. Model merepresentasikan data dan logika aplikasi, View bertanggung jawab untuk visualisasi antarmuka pengguna yang disimpan dalam AXML, dan ViewModel berfungsi sebagai penghubung antara Model dan View, mengelola data yang ditampilkan dan menerima input pengguna.

Pada aplikasi ini, main window menggunakan AXML dengan elemen SplitView yang membagi area kerja menjadi bagian navbar dan konten utama seperti halaman beranda (homepage), halaman about, dan solver page. Navbar menyediakan navigasi antar halaman, sementara solver page memungkinkan pengguna memilih algoritma pencarian sidik jari dan mengunggah gambar sidik jari yang akan dicari. Hasil pencarian ditampilkan dalam bentuk pop-up window (result) yang memberikan informasi detail mengenai hasil pencarian.

Fitur lain yang disediakan termasuk efek transparan pada background untuk tampilan yang lebih modern dan estetis. Selain itu, homepage window memberikan

gambaran umum tentang aplikasi dan fitur-fiturnya, sedangkan solver page menawarkan pilihan algoritma yang dapat digunakan untuk pemrosesan sidik jari. Dengan menggunakan kombinasi C#, Avalonia, dan arsitektur MVVM, aplikasi ini menawarkan performa yang tinggi, fleksibilitas, dan kemudahan dalam pengelolaan serta pengembangan lebih lanjut.

3.4 Contoh Ilustrasi Kasus

Misalnya, didapatkan bentuk string hasil konversi dari gambar sidik jari yang ingin dicari yaitu ABCDCDCD. Kemudian, di dalam database terdapat 5 sidik jari dengan pola string yaitu CDCDCD, ABABAB, OKOKOK, JKJKJK, dan ABCDEF. Pengecekan akan dimulai dengan mencocokkan string hasil konversi dengan string dari sidik jari pertama yang berada di dalam list sidik jari. Jika tidak terdapat pola yang sesuai antara string hasil konversi dengan string sidik jari yang ada pada database, maka sidik jari yang ingin dicari tidak berada dalam database. Berikut adalah ilustrasi pencarian dengan menggunakan algoritma KMP dan BM dengan menjadikan sidik jari pertama sebagai pattern.

3.4.1 KMP

1. Pembentukan array border dengan menggunakan border function (method computeborder)

j	0	1	2	3	4	5
P[j]	C	D	C	D	C	D
k	0	1	2	3	4	-
b(k)	0	0	1	2	1	-

k berhenti pada indeks keempat karena jika tidak mungkin terjadi ketidakcocokan (tidak ditemukan) pola pada indeks yang sama dengan panjang pattern dikurang satu. Array border yang akan terbentuk adalah [0,0,1,2,1]

2. Ilustrasi Pencocokan

Teks : ABCDCDCD

Langkah :

1

1. CDCDCD ($A \neq C$ sehingga pattern akan bergeser 1 kali)

2

2. CDCDCD ($B \neq C$ sehingga pattern akan bergeser 1 kali)

3 4 5 6 7 8

3. CDCDCD (Pola CDCDCD telah ditemukan)

Pola CDCDCD telah ditemukan dalam teks ABCDCDCD dengan jumlah perbandingan karakter sebanyak 8 kali.

3.4.2 BM

1. Pembentukan tabel last occurrence (method Buildlast)

j	0	1	2	3	4	5	other
P[j]	C	D	C	D	C	D	
L(j)	4	5	4	5	4	5	-1

2. Ilustrasi Pencocokan

Teks : ABCDCDCD

Langkah :

5 4 3 2 1

1. CDCDCD ($B \neq D$ dan tidak ada dalam pattern sehingga pattern akan bergeser 1 kali)

111098 7 6

2. CDCDCD (Pola CDCDCD telah ditemukan)

Pola CDCDCD telah ditemukan dalam teks ABCDCDCD dengan jumlah perbandingan karakter sebanyak 11 kali.

BAB 4 IMPLEMENTASI DAN PENGUJIAN

4.1 Spesifikasi Teknis Program

1. Struktur data

Struktur data dalam program ini terdiri dari beberapa kelas yang memiliki peran penting dalam pengelolaan objek dan logika aplikasi. Kelas `People` dalam `People.cs` bertanggung jawab untuk membuat objek yang mewakili orang dengan atribut seperti NIK, nama, tempat lahir, tanggal lahir, jenis kelamin, golongan darah, alamat, agama, status perkawinan, pekerjaan, dan kewarganegaraan. Kelas ini juga menyediakan metode untuk mencetak informasi orang tersebut dan memungkinkan perubahan beberapa atributnya. Kelas `SidikJari` dalam `SidikJari.cs` adalah sebuah struct yang digunakan untuk menyimpan informasi berkas citra sidik jari dan nama terkait. Selanjutnya, kelas `Result` dalam `Result.cs` menghasilkan objek hasil yang berisi atribut seperti gambar, teks, dan objek `People`. Objek `Result` ini bisa diisi dengan nilai yang diperoleh dari hasil pencarian dengan cara passing parameter melalui konstruktor atau metode set. Kelas ini juga memiliki metode statis untuk mengatur atribut `People`, mengubah nama, mengatur persentase hasil, dan mencetak informasi `People`. Selain itu, ada kelas lain yang berperan sebagai logika aplikasi dan tidak bertanggung jawab dalam pembuatan objek. Dalam kelas-kelas tersebut kami juga menggunakan penerapan data array dan tipe data lainnya yang membantu pemrosesan. Struktur ini memungkinkan pemisahan yang jelas antara data dan logika, memastikan bahwa setiap kelas memiliki tanggung jawab yang spesifik dalam pengelolaan data dan operasional aplikasi.

2. Fungsi dan Prosedur

Fungsi dan prosedur yang dijelaskan dalam tabel ini hanya fungsi dan prosedur utama dari proses pencarian solusi sehingga fungsi maupun prosedur seperti frontend dan beberapa fungsi integrasi antara frontend dan backend tidak ada dalam tabel berikut.

Class People.cs
Kelas People.cs bertanggung jawab untuk merepresentasikan data pribadi seseorang dengan berbagai atribut seperti NIK, nama, tempat lahir, tanggal lahir, jenis kelamin, golongan darah, alamat, agama, status perkawinan, pekerjaan, dan kewarganegaraan, serta menyediakan metode untuk mengakses, mengubah, dan menampilkan informasi tersebut.
Atribut

```

1 private string _nik;
2 private string _nama;
3 private string _tempat_lahir;
4 private string _tanggal_lahir;
5 private string _jenis_kelamin;
6 private string _golongan_darah;
7 private string _alamat;
8 private string _agama;
9 private string _status_perkawinan;
10 private string _pekerjaan;
11 private string _kewarganegaraan;

```

Konstruktor



Kode di samping merupakan konstruktor dari objek People.

Fungsi/Prosedur

Penjelasan

```

public override string
ToString()

```

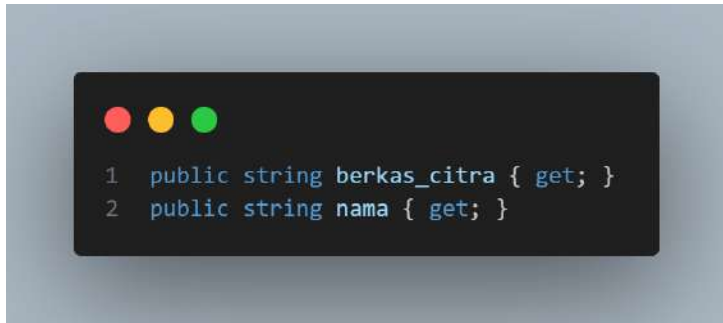
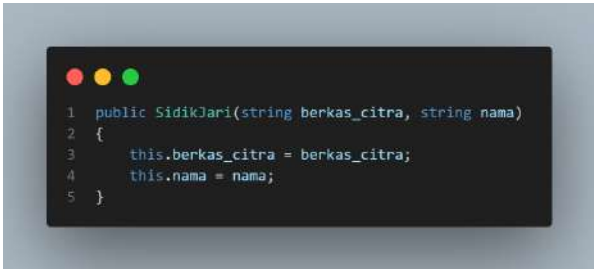
Metode ToString bertanggung jawab untuk menghasilkan representasi string dari objek People dengan menampilkan semua atributnya dalam format yang rapi dan terbaca.

```

public void print ()

```

Metode print bertanggung jawab untuk mencetak setiap atribut dari objek People ke konsol dalam format yang mudah dibaca,

	dengan setiap atribut ditampilkan di baris yang terpisah.
Class SidikJari.java	
Kelas `SidikJari.cs` bertanggung jawab untuk merepresentasikan data sidik jari dengan atribut berkas citra dan nama dalam bentuk struct.	
Atribut	
 <pre> 1 public string berkas_citra { get; } 2 public string nama { get; } </pre>	
Konstruktor	
 <pre> 1 public SidikJari(string berkas_citra, string nama) 2 { 3 this.berkas_citra = berkas_citra; 4 this.nama = nama; 5 } </pre>	Kode di samping merupakan konstruktor dari objek SidikJari di mana sebuah SidikJari akan menyimpan informasi berupa berkas citra dan nama.
Class Result.cs	

Kelas Result.cs bertanggung jawab untuk menyimpan dan mengelola informasi terkait hasil identifikasi, termasuk objek People, gambar, teks, perbedaan waktu, dan persentase kecocokan.

Atribut

```
1 public static Bitmap _image;
2
3 public static TimeSpan timeDiff;
4
5 public static int percentage;
6 public Bitmap Image { get; set; }
7 public string Text { get; set; }
8 public People People { get; set; }
```

Konstruktor



Kode di samping merupakan konstruktor dari objek Result.

Fungsi/Prosedur

Penjelasan

```
public static void
createNewPeople (People people)
```

Metode createNewPeople bertanggung jawab untuk mengganti objek People statis _people dengan objek People baru yang diterima sebagai parameter. Ketika metode ini dipanggil dengan objek People baru, objek _people statis akan diganti dengan objek baru

	tersebut, sehingga setiap referensi ke <code>_people</code> setelahnya akan mengacu pada objek <code>People</code> yang baru.
<pre>public static void setName(String nama)</pre>	Metode <code>setName</code> bertanggung jawab untuk mengubah nama dari objek <code>People</code> statis <code>_people</code> dengan nilai yang diterima sebagai parameter. Ketika metode ini dipanggil dengan string <code>nama</code> , properti <code>Nama</code> dari objek <code>_people</code> statis akan diperbarui dengan nilai baru tersebut.
<pre>public static void setPercentage(int percentage)</pre>	Metode <code>setPercentage</code> bertanggung jawab untuk mengubah nilai persentase statis <code>percentage</code> dengan nilai yang diterima sebagai parameter. Ketika metode ini dipanggil dengan integer <code>percentage</code> , variabel statis <code>percentage</code> di kelas <code>Result</code> akan diperbarui dengan nilai baru tersebut.
<pre>public static void print()</pre>	Metode <code>print</code> bertanggung jawab untuk mencetak ke konsol informasi tertentu dari objek <code>People</code> statis <code>_people</code> , yaitu <code>nama</code> , <code>tempat lahir</code> , dan <code>tanggal lahir</code> . Ketika metode ini dipanggil, nilai dari properti <code>Nama</code> , <code>TempatLahir</code> , dan <code>TanggalLahir</code> dari objek

	_people statis akan dicetak ke konsol, masing-masing di baris yang terpisah.
Class Utils.cs	
Kelas `Utils.cs` bertanggung jawab untuk menyediakan metode utilitas yang mengonversi file menjadi array byte dan mengonversi array byte menjadi objek `Bitmap`.	
Fungsi/Prosedur	Penjelasan
<pre>public static byte[] ConvertToBinary(string filePath)</pre>	Metode ConvertToBinary bertanggung jawab untuk mengonversi file yang terletak di filePath menjadi array byte dengan membaca seluruh konten file. Ketika metode ini dipanggil dengan parameter filePath, jalur file tersebut akan dimodifikasi untuk memastikan format yang benar, kemudian file akan dibaca seluruhnya ke dalam array byte yang kemudian dikembalikan.
<pre>public static Bitmap ConvertToBitmap(byte[] data)</pre>	Metode ConvertToBitmap bertanggung jawab untuk mengonversi array byte menjadi objek Bitmap dengan menggunakan MemoryStream. Ketika metode ini dipanggil dengan parameter data, array byte tersebut akan dimasukkan ke dalam MemoryStream, kemudian MemoryStream tersebut digunakan untuk membuat objek Bitmap yang dikembalikan.
Class Database.cs	

Kelas Utils.cs bertanggung jawab untuk menyediakan metode utilitas yang mengonversi file menjadi array byte dan mengonversi array byte menjadi objek `Bitmap`.

Atribut Statik

```
1 public static List<People> BIODATA = new List<People>();  
2 public static List<SidikJari> SIDIK_JARI = new List<SidikJari>();
```

Fungsi/Prosedur	Penjelasan
<pre>public static void Load()</pre>	<p>Metode Load bertanggung jawab untuk membuka koneksi ke database MySQL dan memanggil metode LoadBiodata dan LoadSidikJari untuk memuat data ke dalam daftar BIODATA dan SIDIK_JARI. Ketika metode ini dipanggil, ia membuat koneksi ke database dengan string koneksi yang diberikan, membuka koneksi tersebut, kemudian memanggil LoadBiodata dan LoadSidikJari untuk memuat data.</p>

<pre>private static void LoadBiodata(MySqlConnection cn)</pre>	<p>Metode LoadBiodata bertanggung jawab untuk memuat data dari tabel biodata di database MySQL ke dalam daftar BIODATA. Ketika metode ini dipanggil, ia membersihkan daftar BIODATA, menjalankan query untuk mengambil semua data dari tabel biodata, membaca setiap baris hasil query, membuat objek People dari data tersebut, dan menambahkannya ke daftar BIODATA.</p>
<pre>private static void LoadSidikJari(MySqlConnection cn)</pre>	<p>Metode LoadSidikJari bertanggung jawab untuk memuat data dari tabel sidik_jari di database MySQL ke dalam daftar SIDIK_JARI. Ketika metode ini dipanggil, ia membersihkan daftar SIDIK_JARI, menjalankan query untuk mengambil semua data dari tabel sidik_jari, membaca setiap baris hasil query, membuat objek SidikJari dari data tersebut, dan menambahkannya ke daftar SIDIK_JARI.</p>
<p align="center">Class ImageConverter.cs</p>	
<p>Kelas ImageConverter.cs bertanggung jawab untuk melakukan konversi dan pra-pemrosesan gambar, termasuk mengubah gambar menjadi array byte dan string biner.</p>	
<p align="center">Fungsi/Prosedur</p>	<p align="center">Penjelasan</p>

<pre>public static byte[] PreprocessImage (string imagePath)</pre>	<p>Metode PreprocessImage bertanggung jawab untuk melakukan pra-pemrosesan pada gambar yang diberikan, termasuk mengubahnya menjadi citra biner dan menyimpannya dalam format file biner. Ketika metode ini dipanggil dengan parameter imagePath, gambar dibaca menggunakan EmguCV, kemudian diubah menjadi citra grayscale, diresize ke ukuran yang diinginkan, dikonversi menjadi citra biner, kemudian disimpan sebagai file biner dan seluruh kontennya dibaca sebagai array byte.</p>
<pre>public static string ImgPathToString (string imagePath)</pre>	<p>Metode ImgPathToString bertanggung jawab untuk mengubah gambar yang diberikan menjadi string biner. Ketika metode ini dipanggil dengan parameter imagePath, seluruh konten dari gambar dibaca sebagai array byte, kemudian array byte tersebut diubah menjadi string menggunakan pengkodean ISO-8859-1.</p>
<p style="text-align: center;">Class MyRegex.cs</p>	
<p>Kelas MyRegex.cs bertanggung jawab untuk menyediakan metode untuk melakukan pencocokan pola teks dengan pola yang dihasilkan berdasarkan teks masukan.</p>	

Atribut

```

1 private static List<char> vocals = new List<char> {'a', 'i', 'u', 'e', 'o'};
2
3 private static Dictionary<char, char> alay = new Dictionary<char, char>()
4 {
5     { 'o', '0' },
6     { 'i', '1' },
7     { 'z', '2' },
8     { 'e', '3' },
9     { 'a', '4' },
10    { 's', '5' },
11    { 'g', '6' },
12 };

```

Fungsi/Prosedur	Penjelasan
<pre> public static bool match(string text1, string text2) </pre>	<p>Metode match bertanggung jawab untuk memeriksa apakah pola teks pertama cocok dengan pola teks kedua. Ketika metode ini dipanggil dengan parameter text1 dan text2, ia membuat pola regex menggunakan getPattern, kemudian menggunakan regex tersebut untuk mencocokkan teks pertama dengan teks kedua dan mengembalikan hasil pencocokan.</p>
<pre> public static string getPattern(string text) </pre>	<p>Metode getPattern bertanggung jawab untuk menghasilkan pola regex dari teks masukan. Ketika metode ini dipanggil dengan parameter text, ia membuat pola regex dengan mengecek</p>

	<p>setiap karakter dalam teks dan menggantinya dengan karakter yang sesuai berdasarkan aturan tertentu, seperti mengubah huruf menjadi huruf besar dan kecil, dan menggantikan karakter alay dengan karakter yang sesuai. Kemudian, ia mengembalikan pola regex yang dihasilkan.</p>
<p>Class KMP.cs</p>	
<p>Kelas KMP.cs bertanggung jawab untuk mencari pola yang cocok dalam daftar sidik jari dan data biodata menggunakan algoritma Knuth-Morris-Pratt (KMP).</p>	
<p>Fungsi/Prosedur</p>	<p>Penjelasan</p>
<pre>public static bool findMatch(string pattern)</pre>	<p>Metode findMatch bertanggung jawab untuk menemukan pola yang cocok dalam daftar sidik jari menggunakan algoritma KMP, dan jika cocok, mengupdate objek Result dengan hasil yang sesuai. Metode ini pertama-tama mendapatkan daftar sidik jari dari database. Kemudian, melakukan pencarian iteratif terhadap bagian-bagian dari pola yang akan dicocokkan, mengurangi ukuran daftar sidik jari yang cocok setiap kali. Jika ada kecocokan ditemukan, data biodata yang cocok diambil dan disimpan di objek Result, bersama dengan gambar yang sesuai.</p>

	Kemudian, metode mengembalikan hasil kebenaran pencocokan.
<pre>public static bool match(string t, string p)</pre>	Metode match bertanggung jawab untuk melakukan pencocokan pola menggunakan algoritma KMP. Metode ini menerapkan algoritma KMP untuk mencocokkan pola dengan teks yang diberikan, dan mengembalikan hasil kebenaran pencocokan.
<pre>private static int[] computeBorder(char[] pattern)</pre>	Metode computeBorder bertanggung jawab untuk menghitung border array untuk pola yang diberikan. Metode ini menghitung dan mengembalikan array border untuk pola yang diberikan, yang digunakan dalam algoritma KMP untuk mempercepat pencocokan pola.
Class BoyerMoore.cs	
Kelas BoyerMoore.cs bertanggung jawab untuk mencari pola yang cocok dalam daftar sidik jari dan data biodata menggunakan algoritma Boyer-Moore.	
Fungsi/Prosedur	Penjelasan
<pre>public static bool findMatch(string pattern)</pre>	Metode findMatch bertanggung jawab untuk menemukan pola yang cocok dalam daftar sidik jari menggunakan algoritma

	<p>Boyer-Moore, dan jika cocok, mengupdate objek Result dengan hasil yang sesuai. Metode ini pertama-tama mendapatkan daftar sidik jari dari database. Kemudian, melakukan pencarian iteratif terhadap bagian-bagian dari pola yang akan dicocokkan, mengurangi ukuran daftar sidik jari yang cocok setiap kali. Jika ada kecocokan ditemukan, data biodata yang cocok diambil dan disimpan di objek Result, bersama dengan gambar yang sesuai. Kemudian, metode mengembalikan hasil kebenaran pencocokan.</p>
<pre>public static bool BmMatch(string text, string pattern)</pre>	<p>Metode BmMatch bertanggung jawab untuk melakukan pencocokan pola menggunakan algoritma Boyer-Moore. Metode ini menerapkan algoritma Boyer-Moore untuk mencocokkan pola dengan teks yang diberikan, dan mengembalikan hasil kebenaran pencocokan.</p>
<pre>public static int[] BuildLast(string pattern)</pre>	<p>Metode BuildLast bertanggung jawab untuk membangun tabel "last" yang digunakan dalam algoritma Boyer-Moore. Metode ini membangun tabel "last" yang menyimpan indeks terakhir dari setiap karakter dalam pola yang dicari, yang digunakan dalam pencocokan pola Boyer-Moore.</p>

Class HammingDist.cs	
Kelas HammingDist.cs bertanggung jawab untuk menemukan pola yang cocok dalam daftar sidik jari menggunakan metode Hamming Distance.	
Fungsi/Prosedur	Penjelasan
<pre>public static bool findMatch(string pattern)</pre>	<p>Metode findMatch bertanggung jawab untuk mencari pola yang cocok dalam daftar sidik jari menggunakan metode Hamming Distance, dan jika cocok, mengupdate objek Result dengan hasil yang sesuai. Metode ini pertama-tama mendapatkan daftar sidik jari dari database. Kemudian, menghitung Hamming Distance antara pola yang dicari dan setiap sidik jari dalam daftar. Mencari sidik jari dengan Hamming Distance terkecil, kemudian mencocokkan nama sidik jari yang cocok dengan data biodata menggunakan regex. Jika cocok, data biodata yang sesuai disimpan di objek Result bersama dengan gambar yang sesuai, dan waktu yang dibutuhkan untuk pencarian.</p>
<pre>public static int hamming(string a, string b, int m, int n)</pre>	<p>Metode hamming bertanggung jawab untuk menghitung Hamming Distance antara dua string. Metode ini mengiterasi melalui</p>

	<p>karakter-karakter dari dua string hingga panjang yang lebih kecil di antara keduanya, dan menghitung jumlah karakter yang berbeda di posisi yang sama. Jumlah karakter yang berbeda tersebut adalah Hamming Distance.</p>
--	--

4.2 Source Code

1. KMP.cs

```

1 public class BM
2 {
3     public static void findMatch(String pattern)
4     {
5         long startTime = System.nanoTime();
6
7         List<Integer> sids = new ArrayList<>();
8
9         int patternLength = pattern.length();
10        int sidsLen = 0;
11        Console.WriteLine("Pattern length: " + patternLength);
12
13        int previous = sidsLen;
14        int current = 0;
15
16        int loop = 0;
17        while (sidsLen < patternLength)
18        {
19            string currentStr = pattern.Substring(patternLength/2-loop, loop);
20            previous = sidsLen;
21
22            sidsLen = sidsLen + sidsLen + match(pattern, currentStr, sidsLen, loop);
23
24            sids.Add(sidsLen);
25            patternLength = sidsLen;
26            Console.WriteLine(sidsLen);
27        }
28
29        Console.WriteLine(sidsLen);
30
31        return sidsLen;
32    }
33
34    private static void findMatch(String pattern, int sidsLen)
35    {
36        try {
37            Result obj = new Result();
38
39            obj.sidsLen = sidsLen;
40
41            obj.startTime = System.nanoTime();
42            obj.endTime = System.nanoTime();
43
44            obj.result = match(pattern, currentStr, sidsLen, loop);
45
46            obj.sidsLen = sidsLen;
47
48            Console.WriteLine("Match found using BM");
49
50            return obj;
51        }
52        catch {
53            return null;
54        }
55    }
56
57    private static void findMatch(String pattern, int sidsLen)
58    {
59        try {
60            Result obj = new Result();
61
62            obj.sidsLen = sidsLen;
63
64            obj.startTime = System.nanoTime();
65            obj.endTime = System.nanoTime();
66
67            obj.result = match(pattern, currentStr, sidsLen, loop);
68
69            obj.sidsLen = sidsLen;
70
71            Console.WriteLine("Match found using BM");
72
73            return obj;
74        }
75        catch {
76            return null;
77        }
78    }
79
80    private static void findMatch(String pattern, int sidsLen)
81    {
82        try {
83            Result obj = new Result();
84
85            obj.sidsLen = sidsLen;
86
87            obj.startTime = System.nanoTime();
88            obj.endTime = System.nanoTime();
89
90            obj.result = match(pattern, currentStr, sidsLen, loop);
91
92            obj.sidsLen = sidsLen;
93
94            Console.WriteLine("Match found using BM");
95
96            return obj;
97        }
98        catch {
99            return null;
100        }
101    }
102
103    private static void findMatch(String pattern, int sidsLen)
104    {
105        try {
106            Result obj = new Result();
107
108            obj.sidsLen = sidsLen;
109
110            obj.startTime = System.nanoTime();
111            obj.endTime = System.nanoTime();
112
113            obj.result = match(pattern, currentStr, sidsLen, loop);
114
115            obj.sidsLen = sidsLen;
116
117            Console.WriteLine("Match found using BM");
118
119            return obj;
120        }
121        catch {
122            return null;
123        }
124    }
125
126    private static void findMatch(String pattern, int sidsLen)
127    {
128        try {
129            Result obj = new Result();
130
131            obj.sidsLen = sidsLen;
132
133            obj.startTime = System.nanoTime();
134            obj.endTime = System.nanoTime();
135
136            obj.result = match(pattern, currentStr, sidsLen, loop);
137
138            obj.sidsLen = sidsLen;
139
140            Console.WriteLine("Match found using BM");
141
142            return obj;
143        }
144        catch {
145            return null;
146        }
147    }
148
149    private static void findMatch(String pattern, int sidsLen)
150    {
151        try {
152            Result obj = new Result();
153
154            obj.sidsLen = sidsLen;
155
156            obj.startTime = System.nanoTime();
157            obj.endTime = System.nanoTime();
158
159            obj.result = match(pattern, currentStr, sidsLen, loop);
160
161            obj.sidsLen = sidsLen;
162
163            Console.WriteLine("Match found using BM");
164
165            return obj;
166        }
167        catch {
168            return null;
169        }
170    }
171
172    private static void findMatch(String pattern, int sidsLen)
173    {
174        try {
175            Result obj = new Result();
176
177            obj.sidsLen = sidsLen;
178
179            obj.startTime = System.nanoTime();
180            obj.endTime = System.nanoTime();
181
182            obj.result = match(pattern, currentStr, sidsLen, loop);
183
184            obj.sidsLen = sidsLen;
185
186            Console.WriteLine("Match found using BM");
187
188            return obj;
189        }
190        catch {
191            return null;
192        }
193    }
194
195    private static void findMatch(String pattern, int sidsLen)
196    {
197        try {
198            Result obj = new Result();
199
200            obj.sidsLen = sidsLen;
201
202            obj.startTime = System.nanoTime();
203            obj.endTime = System.nanoTime();
204
205            obj.result = match(pattern, currentStr, sidsLen, loop);
206
207            obj.sidsLen = sidsLen;
208
209            Console.WriteLine("Match found using BM");
210
211            return obj;
212        }
213        catch {
214            return null;
215        }
216    }
217
218    private static void findMatch(String pattern, int sidsLen)
219    {
220        try {
221            Result obj = new Result();
222
223            obj.sidsLen = sidsLen;
224
225            obj.startTime = System.nanoTime();
226            obj.endTime = System.nanoTime();
227
228            obj.result = match(pattern, currentStr, sidsLen, loop);
229
230            obj.sidsLen = sidsLen;
231
232            Console.WriteLine("Match found using BM");
233
234            return obj;
235        }
236        catch {
237            return null;
238        }
239    }
240
241    private static void findMatch(String pattern, int sidsLen)
242    {
243        try {
244            Result obj = new Result();
245
246            obj.sidsLen = sidsLen;
247
248            obj.startTime = System.nanoTime();
249            obj.endTime = System.nanoTime();
250
251            obj.result = match(pattern, currentStr, sidsLen, loop);
252
253            obj.sidsLen = sidsLen;
254
255            Console.WriteLine("Match found using BM");
256
257            return obj;
258        }
259        catch {
260            return null;
261        }
262    }
263
264    private static void findMatch(String pattern, int sidsLen)
265    {
266        try {
267            Result obj = new Result();
268
269            obj.sidsLen = sidsLen;
270
271            obj.startTime = System.nanoTime();
272            obj.endTime = System.nanoTime();
273
274            obj.result = match(pattern, currentStr, sidsLen, loop);
275
276            obj.sidsLen = sidsLen;
277
278            Console.WriteLine("Match found using BM");
279
280            return obj;
281        }
282        catch {
283            return null;
284        }
285    }
286
287    private static void findMatch(String pattern, int sidsLen)
288    {
289        try {
290            Result obj = new Result();
291
292            obj.sidsLen = sidsLen;
293
294            obj.startTime = System.nanoTime();
295            obj.endTime = System.nanoTime();
296
297            obj.result = match(pattern, currentStr, sidsLen, loop);
298
299            obj.sidsLen = sidsLen;
300
301            Console.WriteLine("Match found using BM");
302
303            return obj;
304        }
305        catch {
306            return null;
307        }
308    }
309
310    private static void findMatch(String pattern, int sidsLen)
311    {
312        try {
313            Result obj = new Result();
314
315            obj.sidsLen = sidsLen;
316
317            obj.startTime = System.nanoTime();
318            obj.endTime = System.nanoTime();
319
320            obj.result = match(pattern, currentStr, sidsLen, loop);
321
322            obj.sidsLen = sidsLen;
323
324            Console.WriteLine("Match found using BM");
325
326            return obj;
327        }
328        catch {
329            return null;
330        }
331    }
332
333    private static void findMatch(String pattern, int sidsLen)
334    {
335        try {
336            Result obj = new Result();
337
338            obj.sidsLen = sidsLen;
339
340            obj.startTime = System.nanoTime();
341            obj.endTime = System.nanoTime();
342
343            obj.result = match(pattern, currentStr, sidsLen, loop);
344
345            obj.sidsLen = sidsLen;
346
347            Console.WriteLine("Match found using BM");
348
349            return obj;
350        }
351        catch {
352            return null;
353        }
354    }
355
356    private static void findMatch(String pattern, int sidsLen)
357    {
358        try {
359            Result obj = new Result();
360
361            obj.sidsLen = sidsLen;
362
363            obj.startTime = System.nanoTime();
364            obj.endTime = System.nanoTime();
365
366            obj.result = match(pattern, currentStr, sidsLen, loop);
367
368            obj.sidsLen = sidsLen;
369
370            Console.WriteLine("Match found using BM");
371
372            return obj;
373        }
374        catch {
375            return null;
376        }
377    }
378
379    private static void findMatch(String pattern, int sidsLen)
380    {
381        try {
382            Result obj = new Result();
383
384            obj.sidsLen = sidsLen;
385
386            obj.startTime = System.nanoTime();
387            obj.endTime = System.nanoTime();
388
389            obj.result = match(pattern, currentStr, sidsLen, loop);
390
391            obj.sidsLen = sidsLen;
392
393            Console.WriteLine("Match found using BM");
394
395            return obj;
396        }
397        catch {
398            return null;
399        }
400    }
401
402    private static void findMatch(String pattern, int sidsLen)
403    {
404        try {
405            Result obj = new Result();
406
407            obj.sidsLen = sidsLen;
408
409            obj.startTime = System.nanoTime();
410            obj.endTime = System.nanoTime();
411
412            obj.result = match(pattern, currentStr, sidsLen, loop);
413
414            obj.sidsLen = sidsLen;
415
416            Console.WriteLine("Match found using BM");
417
418            return obj;
419        }
420        catch {
421            return null;
422        }
423    }
424
425    private static void findMatch(String pattern, int sidsLen)
426    {
427        try {
428            Result obj = new Result();
429
430            obj.sidsLen = sidsLen;
431
432            obj.startTime = System.nanoTime();
433            obj.endTime = System.nanoTime();
434
435            obj.result = match(pattern, currentStr, sidsLen, loop);
436
437            obj.sidsLen = sidsLen;
438
439            Console.WriteLine("Match found using BM");
440
441            return obj;
442        }
443        catch {
444            return null;
445        }
446    }
447
448    private static void findMatch(String pattern, int sidsLen)
449    {
450        try {
451            Result obj = new Result();
452
453            obj.sidsLen = sidsLen;
454
455            obj.startTime = System.nanoTime();
456            obj.endTime = System.nanoTime();
457
458            obj.result = match(pattern, currentStr, sidsLen, loop);
459
460            obj.sidsLen = sidsLen;
461
462            Console.WriteLine("Match found using BM");
463
464            return obj;
465        }
466        catch {
467            return null;
468        }
469    }
470
471    private static void findMatch(String pattern, int sidsLen)
472    {
473        try {
474            Result obj = new Result();
475
476            obj.sidsLen = sidsLen;
477
478            obj.startTime = System.nanoTime();
479            obj.endTime = System.nanoTime();
480
481            obj.result = match(pattern, currentStr, sidsLen, loop);
482
483            obj.sidsLen = sidsLen;
484
485            Console.WriteLine("Match found using BM");
486
487            return obj;
488        }
489        catch {
490            return null;
491        }
492    }
493
494    private static void findMatch(String pattern, int sidsLen)
495    {
496        try {
497            Result obj = new Result();
498
499            obj.sidsLen = sidsLen;
500
501            obj.startTime = System.nanoTime();
502            obj.endTime = System.nanoTime();
503
504            obj.result = match(pattern, currentStr, sidsLen, loop);
505
506            obj.sidsLen = sidsLen;
507
508            Console.WriteLine("Match found using BM");
509
510            return obj;
511        }
512        catch {
513            return null;
514        }
515    }
516
517    private static void findMatch(String pattern, int sidsLen)
518    {
519        try {
520            Result obj = new Result();
521
522            obj.sidsLen = sidsLen;
523
524            obj.startTime = System.nanoTime();
525            obj.endTime = System.nanoTime();
526
527            obj.result = match(pattern, currentStr, sidsLen, loop);
528
529            obj.sidsLen = sidsLen;
530
531            Console.WriteLine("Match found using BM");
532
533            return obj;
534        }
535        catch {
536            return null;
537        }
538    }
539
540    private static void findMatch(String pattern, int sidsLen)
541    {
542        try {
543            Result obj = new Result();
544
545            obj.sidsLen = sidsLen;
546
547            obj.startTime = System.nanoTime();
548            obj.endTime = System.nanoTime();
549
550            obj.result = match(pattern, currentStr, sidsLen, loop);
551
552            obj.sidsLen = sidsLen;
553
554            Console.WriteLine("Match found using BM");
555
556            return obj;
557        }
558        catch {
559            return null;
560        }
561    }
562
563    private static void findMatch(String pattern, int sidsLen)
564    {
565        try {
566            Result obj = new Result();
567
568            obj.sidsLen = sidsLen;
569
570            obj.startTime = System.nanoTime();
571            obj.endTime = System.nanoTime();
572
573            obj.result = match(pattern, currentStr, sidsLen, loop);
574
575            obj.sidsLen = sidsLen;
576
577            Console.WriteLine("Match found using BM");
578
579            return obj;
580        }
581        catch {
582            return null;
583        }
584    }
585
586    private static void findMatch(String pattern, int sidsLen)
587    {
588        try {
589            Result obj = new Result();
590
591            obj.sidsLen = sidsLen;
592
593            obj.startTime = System.nanoTime();
594            obj.endTime = System.nanoTime();
595
596            obj.result = match(pattern, currentStr, sidsLen, loop);
597
598            obj.sidsLen = sidsLen;
599
600            Console.WriteLine("Match found using BM");
601
602            return obj;
603        }
604        catch {
605            return null;
606        }
607    }
608
609    private static void findMatch(String pattern, int sidsLen)
610    {
611        try {
612            Result obj = new Result();
613
614            obj.sidsLen = sidsLen;
615
616            obj.startTime = System.nanoTime();
617            obj.endTime = System.nanoTime();
618
619            obj.result = match(pattern, currentStr, sidsLen, loop);
620
621            obj.sidsLen = sidsLen;
622
623            Console.WriteLine("Match found using BM");
624
625            return obj;
626        }
627        catch {
628            return null;
629        }
630    }
631
632    private static void findMatch(String pattern, int sidsLen)
633    {
634        try {
635            Result obj = new Result();
636
637            obj.sidsLen = sidsLen;
638
639            obj.startTime = System.nanoTime();
640            obj.endTime = System.nanoTime();
641
642            obj.result = match(pattern, currentStr, sidsLen, loop);
643
644            obj.sidsLen = sidsLen;
645
646            Console.WriteLine("Match found using BM");
647
648            return obj;
649        }
650        catch {
651            return null;
652        }
653    }
654
655    private static void findMatch(String pattern, int sidsLen)
656    {
657        try {
658            Result obj = new Result();
659
660            obj.sidsLen = sidsLen;
661
662            obj.startTime = System.nanoTime();
663            obj.endTime = System.nanoTime();
664
665            obj.result = match(pattern, currentStr, sidsLen, loop);
666
667            obj.sidsLen = sidsLen;
668
669            Console.WriteLine("Match found using BM");
670
671            return obj;
672        }
673        catch {
674            return null;
675        }
676    }
677
678    private static void findMatch(String pattern, int sidsLen)
679    {
680        try {
681            Result obj = new Result();
682
683            obj.sidsLen = sidsLen;
684
685            obj.startTime = System.nanoTime();
686            obj.endTime = System.nanoTime();
687
688            obj.result = match(pattern, currentStr, sidsLen, loop);
689
690            obj.sidsLen = sidsLen;
691
692            Console.WriteLine("Match found using BM");
693
694            return obj;
695        }
696        catch {
697            return null;
698        }
699    }
700
701    private static void findMatch(String pattern, int sidsLen)
702    {
703        try {
704            Result obj = new Result();
705
706            obj.sidsLen = sidsLen;
707
708            obj.startTime = System.nanoTime();
709            obj.endTime = System.nanoTime();
710
711            obj.result = match(pattern, currentStr, sidsLen, loop);
712
713            obj.sidsLen = sidsLen;
714
715            Console.WriteLine("Match found using BM");
716
717            return obj;
718        }
719        catch {
720            return null;
721        }
722    }
723
724    private static void findMatch(String pattern, int sidsLen)
725    {
726        try {
727            Result obj = new Result();
728
729            obj.sidsLen = sidsLen;
730
731            obj.startTime = System.nanoTime();
732            obj.endTime = System.nanoTime();
733
734            obj.result = match(pattern, currentStr, sidsLen, loop);
735
736            obj.sidsLen = sidsLen;
737
738            Console.WriteLine("Match found using BM");
739
740            return obj;
741        }
742        catch {
743            return null;
744        }
745    }
746
747    private static void findMatch(String pattern, int sidsLen)
748    {
749        try {
750            Result obj = new Result();
751
752            obj.sidsLen = sidsLen;
753
754            obj.startTime = System.nanoTime();
755            obj.endTime = System.nanoTime();
756
757            obj.result = match(pattern, currentStr, sidsLen, loop);
758
759            obj.sidsLen = sidsLen;
760
761            Console.WriteLine("Match found using BM");
762
763            return obj;
764        }
765        catch {
766            return null;
767        }
768    }
769
770    private static void findMatch(String pattern, int sidsLen)
771    {
772        try {
773            Result obj = new Result();
774
775            obj.sidsLen = sidsLen;
776
777            obj.startTime = System.nanoTime();
778            obj.endTime = System.nanoTime();
779
780            obj.result = match(pattern, currentStr, sidsLen, loop);
781
782            obj.sidsLen = sidsLen;
783
784            Console.WriteLine("Match found using BM");
785
786            return obj;
787        }
788        catch {
789            return null;
790        }
791    }
792
793    private static void findMatch(String pattern, int sidsLen)
794    {
795        try {
796            Result obj = new Result();
797
798            obj.sidsLen = sidsLen;
799
800            obj.startTime = System.nanoTime();
801            obj.endTime = System.nanoTime();
802
803            obj.result = match(pattern, currentStr, sidsLen, loop);
804
805            obj.sidsLen = sidsLen;
806
807            Console.WriteLine("Match found using BM");
808
809            return obj;
810        }
811        catch {
812            return null;
813        }
814    }
815
816    private static void findMatch(String pattern, int sidsLen)
817    {
818        try {
819            Result obj = new Result();
820
821            obj.sidsLen = sidsLen;
822
823            obj.startTime = System.nanoTime();
824            obj.endTime = System.nanoTime();
825
826            obj.result = match(pattern, currentStr, sidsLen, loop);
827
828            obj.sidsLen = sidsLen;
829
830            Console.WriteLine("Match found using BM");
831
832            return obj;
833        }
834        catch {
835            return null;
836        }
837    }
838
839    private static void findMatch(String pattern, int sidsLen)
840    {
841        try {
842            Result obj = new Result();
843
844            obj.sidsLen = sidsLen;
845
846            obj.startTime = System.nanoTime();
847            obj.endTime = System.nanoTime();
848
849            obj.result = match(pattern, currentStr, sidsLen, loop);
850
851            obj.sidsLen = sidsLen;
852
853            Console.WriteLine("Match found using BM");
854
855            return obj;
856        }
857        catch {
858            return null;
859        }
860    }
861
862    private static void findMatch(String pattern, int sidsLen)
863    {
864        try {
865            Result obj = new Result();
866
867            obj.sidsLen = sidsLen;
868
869            obj.startTime = System.nanoTime();
870            obj.endTime = System.nanoTime();
871
872            obj.result = match(pattern, currentStr, sidsLen, loop);
873
874            obj.sidsLen = sidsLen;
875
876            Console.WriteLine("Match found using BM");
877
878            return obj;
879        }
880        catch {
881            return null;
882        }
883    }
884
885    private static void findMatch(String pattern, int sidsLen)
886    {
887        try {
888            Result obj = new Result();
889
890            obj.sidsLen = sidsLen;
891
892            obj.startTime = System.nanoTime();
893            obj.endTime = System.nanoTime();
894
895            obj.result = match(pattern, currentStr, sidsLen, loop);
896
897            obj.sidsLen = sidsLen;
898
899            Console.WriteLine("Match found using BM");
900
901            return obj;
902        }
903        catch {
904            return null;
905        }
906    }
907
908    private static void findMatch(String pattern, int sidsLen)
909    {
910        try {
911            Result obj = new Result();
912
913            obj.sidsLen = sidsLen;
914
915            obj.startTime = System.nanoTime();
916            obj.endTime = System.nanoTime();
917
918            obj.result = match(pattern, currentStr, sidsLen, loop);
919
920            obj.sidsLen = sidsLen;
921
922            Console.WriteLine("Match found using BM");
923
924            return obj;
925        }
926        catch {
927            return null;
928        }
929    }
930
931    private static void findMatch(String pattern, int sidsLen)
932    {
933        try {
934            Result obj = new Result();
935
936            obj.sidsLen = sidsLen;
937
938            obj.startTime = System.nanoTime();
939            obj.endTime = System.nanoTime();
940
941            obj.result = match(pattern, currentStr, sidsLen, loop);
942
943            obj.sidsLen = sidsLen;
944
945            Console.WriteLine("Match found using BM");
946
947            return obj;
948        }
949        catch {
950            return null;
951        }
952    }
953
954    private static void findMatch(String pattern, int sidsLen)
955    {
956        try {
957            Result obj = new Result();
958
959            obj.sidsLen = sidsLen;
960
961            obj.startTime = System.nanoTime();
962            obj.endTime = System.nanoTime();
963
964            obj.result = match(pattern, currentStr, sidsLen, loop);
965
966            obj.sidsLen = sidsLen;
967
968            Console.WriteLine("Match found using BM");
969
970            return obj;
971        }
972        catch {
973            return null;
974        }
975    }
976
977    private static void findMatch(String pattern, int sidsLen)
978    {
979        try {
980            Result obj = new Result();
981
982            obj.sidsLen = sidsLen;
983
984            obj.startTime = System.nanoTime();
985            obj.endTime = System.nanoTime();
986
987            obj.result = match(pattern, currentStr, sidsLen, loop);
988
989            obj.sidsLen = sidsLen;
990
991            Console.WriteLine("Match found using BM");
992
993            return obj;
994        }
995        catch {
996            return null;
997        }
998    }
999
1000    private static void findMatch(String pattern, int sidsLen)
1001    {
1002        try {
1003            Result obj = new Result();
1004
1005            obj.sidsLen = sidsLen;
1006
1007            obj.startTime = System.nanoTime();
1008            obj.endTime = System.nanoTime();
1009
1010            obj.result = match(pattern, currentStr, sidsLen, loop);
1011
1012            obj.sidsLen = sidsLen;
1013
1014            Console.WriteLine("Match found using BM");
1015
1016            return obj;
1017        }
1018        catch {
1019            return null;
1020        }
1021    }
1022
1023    private static void findMatch(String pattern, int sidsLen)
1024    {
1025        try {
1026            Result obj = new Result();
1027
1028            obj.sidsLen = sidsLen;
1029
1030            obj.startTime = System.nanoTime();
1031            obj.endTime = System.nanoTime();
1032
1033            obj.result = match(pattern, currentStr, sidsLen, loop);
1034
1035            obj.sidsLen = sidsLen;
1036
1037            Console.WriteLine("Match found using BM");
1038
1039            return obj;
1040        }
1041        catch {
1042            return null;
1043        }
1044    }
1045
1046    private static void findMatch(String pattern, int sidsLen)
1047    {
1048        try {
1049            Result obj = new Result();
1050
1051            obj.sidsLen = sidsLen;
1052
1053            obj.startTime = System.nanoTime();
1054            obj.endTime = System.nanoTime();
1055
1056            obj.result = match(pattern, currentStr, sidsLen, loop);
1057
1058            obj.sidsLen = sidsLen;
1059
1060            Console.WriteLine("Match found using BM");
1061
1062            return obj;
1063        }
1064        catch {
1065            return null;
1066        }
1067    }
1068
1069    private static void findMatch(String pattern, int sidsLen)
1070    {
1071        try {
1072            Result obj = new Result();
1073
1074            obj.sidsLen = sidsLen;
1075
1076            obj.startTime = System.nanoTime();
1077            obj.endTime = System.nanoTime();
1078
1079            obj.result = match(pattern, currentStr, sidsLen, loop);
1080
1081            obj.sidsLen = sidsLen;
1082
1083            Console.WriteLine("Match found using BM");
1084
1085            return obj;
1086        }
1087        catch {
1088            return null;
1089        }
1090    }
1091
1092    private static void findMatch(String pattern, int sidsLen)
1093    {
1094        try {
1095            Result obj = new Result();
1096
1097            obj.sidsLen = sidsLen;
1098
1099            obj.startTime = System.nanoTime();
1100            obj.endTime = System.nanoTime();
1101
1102            obj.result = match(pattern, currentStr, sidsLen, loop);
1103
1104            obj.sidsLen = sidsLen;
1105
1106            Console.WriteLine("Match found using BM");
1107
1108            return obj;
1109        }
1110        catch {
1111            return null;
1112        }
1113    }
1114
1115    private static void findMatch(String pattern, int sidsLen)
1116    {
1117        try {
1118            Result obj = new Result();
1119
1120            obj.sidsLen = sidsLen;
1121
1122            obj.startTime = System.nanoTime();
1123            obj.endTime = System.nanoTime();
1124
1125            obj.result = match(pattern, currentStr, sidsLen, loop);
1126
1127            obj.sidsLen = sidsLen;
1128
1129            Console.WriteLine("Match found using BM");
1130
1131            return obj;
1132        }
1133        catch {
1134            return null;
1135        }
1136    }
1137
1138    private static void findMatch(String pattern, int sidsLen)
1139    {
1140        try {
1141            Result obj = new Result();
1142
1143            obj.sidsLen = sidsLen;
1144
1145            obj.startTime = System.nanoTime();
1146            obj.endTime = System.nanoTime();
1147
1148            obj.result = match(pattern, currentStr, sidsLen, loop);
1149
1150            obj.sidsLen = sidsLen;
1151
1152            Console.WriteLine("Match found using BM");
1153
1154            return obj;
1155        }
1156        catch {
1157            return null;
1158        }
1159    }
1160
1161    private static void findMatch(String pattern, int sidsLen)
1162    {
1163        try {
1164            Result obj = new Result();
1165
1166            obj.sidsLen = sidsLen;
1167
1168            obj.startTime = System.nanoTime();
1169            obj.endTime = System.nanoTime();
1170
1171            obj.result = match(pattern, currentStr, sidsLen, loop);
1172
1173            obj.sidsLen = sidsLen;
1174
1175            Console.WriteLine("Match found using BM");
1176
1177            return obj;
1178        }
1179        catch {
1180            return null;
1181        }
1182    }
1183
1184    private static void findMatch(String pattern, int sidsLen)
1185    {
1186        try {
1187            Result obj = new Result();
1188
1189            obj.sidsLen = sidsLen;
1190
1191            obj.startTime = System.nanoTime();
1192            obj.endTime = System.nanoTime();
1193
1194            obj.result = match(pattern, currentStr, sidsLen, loop);
1195
1196            obj.sidsLen = sidsLen;
1197
1198            Console.WriteLine("Match found using BM");
1199
1200            return obj;
1201        }
1202        catch {
1203            return null;
1204        }
1205    }
1206
1207    private static void findMatch(String pattern, int sidsLen)
1208    {
1209        try {
1210            Result obj = new Result();
1211
1212            obj.sidsLen = sidsLen;
1213
1214            obj.startTime = System.nanoTime();
1215            obj.endTime = System.nanoTime();
1216
1217            obj.result = match(pattern, currentStr, sidsLen, loop);
1218
1219            obj.sidsLen = sidsLen;
1220
1221            Console.WriteLine("Match found using BM");
1222
1223            return obj;
1224        }
1225        catch {
1226            return null;
1227        }
1228    }
1229
1230    private static void findMatch(String pattern, int sidsLen)
1231    {
1232        try {
1233            Result obj = new Result();
1234
1235            obj.sidsLen = sidsLen;
1236
1237            obj.startTime = System.nanoTime();
1238            obj.endTime = System.nanoTime();
1239
1240            obj.result = match(pattern, currentStr, sidsLen, loop);
1241
1242            obj.sidsLen = sidsLen;
1243
1244            Console.WriteLine("Match found using BM");
1245
1246            return obj;
1247        }
1248        catch {
1249            return null;
1250        }
1251    }
1252
1253    private static void findMatch(String pattern, int sidsLen)
1254    {
1255        try {
1256            Result obj = new Result();
1257
1258            obj.sidsLen = sidsLen;
1259
1260            obj.startTime = System.nanoTime();
1261            obj.endTime = System.nanoTime();
1262
1263            obj.result = match(pattern, currentStr, sidsLen, loop);
1264
1265            obj.sidsLen = sidsLen;
1266
1267            Console.WriteLine("Match found using BM");
1268
1269            return obj;
1270        }
1271        catch {
1272            return null;
1273        }
1274    }
1275
1276    private static void findMatch(String pattern, int sidsLen)
1277    {
1278        try {
1279            Result obj = new Result();
1280
1281            obj.sidsLen = sidsLen;
1282
1283            obj.startTime = System.nanoTime();
1284            obj.endTime = System.nanoTime();
1285
1286            obj.result = match(pattern, currentStr, sidsLen, loop);
1287
1288            obj.sidsLen = sidsLen;
1289
1290            Console.WriteLine("Match found using BM");
1291
1292            return obj;
1293        }
1294        catch {
1295            return null;
1296        }
1297    }
1298
1299    private static void findMatch(String pattern, int sidsLen)
1300    {
1301        try {
1302            Result obj = new Result();
1303
1304            obj.sidsLen = sidsLen;
1305
1306            obj.startTime = System.nanoTime();
1307            obj.endTime = System.nanoTime();
1308
1309            obj.result = match(pattern, currentStr, sidsLen, loop);
1310
1311            obj.sidsLen = sidsLen;
1312
1313            Console.WriteLine("Match found using BM");
1314
1315            return obj;
1316        }
1317        catch {
1318            return null;
1319        }
1320    }
1321
1322    private static void findMatch(String pattern, int sidsLen)
1323    {
1324        try {
1325            Result obj = new Result();
1326
1327            obj.sidsLen = sidsLen;
1328
1329            obj.startTime = System.nanoTime();
1330            obj.endTime = System.nanoTime();
1331
1332            obj.result = match(pattern, currentStr, sidsLen, loop);
1333
1334            obj.sidsLen = sidsLen;
1335
1336            Console.WriteLine("Match found using BM");
1337
1338            return obj;
1339        }
1340        catch {
1341            return null;
1342        }
1343    }
1344
1345    private static void findMatch(String pattern, int sidsLen)
1346    {
1347        try {
1348            Result obj = new Result();
1349
1350            obj.sidsLen = sidsLen;
1351
1352            obj.startTime = System.nanoTime();
1353            obj.endTime = System.nanoTime();
1354
1355            obj.result = match(pattern, currentStr, sidsLen, loop);
1356
1357            obj.sidsLen = sidsLen;
1358
1359            Console.WriteLine("Match found using BM");
1360
1361            return obj;
1362        }
1363        catch {
1364            return null;
1365        }
1366    }
1367
1368    private static void findMatch(String pattern, int sidsLen)
1369    {
1370        try {
1371            Result obj = new Result();
1372
1373            obj.sidsLen = sidsLen;
1374
1375            obj.startTime = System.nanoTime();
1376            obj.endTime = System.nanoTime();
1377
1378            obj.result = match(pattern, currentStr, sidsLen, loop);
1379
1380            obj.sidsLen = sidsLen;
1381
1382            Console.WriteLine("Match found using BM");
1383
1384            return obj;
1385        }
1386        catch {
1387            return null;
1388        }
1389    }
1390
1391    private static void findMatch(String pattern, int sidsLen)
1392    {
1393        try {
1394            Result obj = new Result();
1395
1396            obj.sidsLen = sidsLen;
1397
1398            obj.startTime = System.nanoTime();
1399            obj.endTime = System.nanoTime();
1400
1401            obj.result = match(pattern, currentStr, sidsLen, loop);
1402
1403            obj.sidsLen = sidsLen;
1404
1405            Console.WriteLine("Match found using BM");
1406
1407            return obj;
1408        }
1409        catch {
1410            return null;
1411        }
1412    }
1413
1414    private static void findMatch(String pattern, int sidsLen)
1415    {
1416        try {
1417            Result obj = new Result();
1418
1419            obj.sidsLen = sidsLen;
1420
1421            obj.startTime = System.nanoTime();
14
```



```

1 public class BoyerKore
2 {
3     public static bool findMatch(string pattern)
4     {
5         DateTime startTime = DateTime.Now;
6
7         List<sidikJar> sidikJarList = new List<sidikJar>(Database.SIDIK_JARI);
8
9         int patternLength = pattern.Length;
10        int setSize = 64;
11        Console.WriteLine("Pattern Length: " + patternLength);
12
13        int prevLen = sidikJarList.Count;
14        int newLen = 0;
15
16        int loop = 0;
17
18        while (sidikJarList.Count != 1 && prevLen != newLen && loop*setSize < patternLength/2)
19        {
20            string currentSet = pattern.Substring(patternLength/2-loop*setSize, setSize);
21            prevLen = sidikJarList.Count;
22
23            sidikJarList = sidikJarList.Where(sidikJar => BMATCH(ImageConverter.TagPathToString(sidikJar.berkas_citra), currentSet)).ToList();
24
25            newLen = sidikJarList.Count;
26            patternLength = newLen;
27            loop++;
28            Console.WriteLine(sidikJarList.Count);
29        }
30
31        Console.WriteLine(sidikJarList[0].nama);
32
33        foreach (Utils.People biodata in Database.SIDIKDATA)
34        {
35            try {
36                if (MyImagev.match(biodata.Nama, sidikJarList[0].nama)) {
37
38                    Result_Image = Utils.Utils.ConvertToBitmap(Encoding.GetEncoding("iso-8859-1").GetBytes(ImageConverter.TagPathToString(sidikJarList[0].berkas_citra)));
39
40                    DateTime endTime = DateTime.Now;
41                    TimeSpan timeDiff = endTime - startTime;
42
43                    Result.timeDiff = timeDiff;
44                    Result.percentage = 0;
45
46                    Result.createNewPeople(biodata);
47                    Result.setName(sidikJarList[0].nama);
48                    Result.percentage = 100;
49
50                    Console.WriteLine("Match found using BM");
51                    return true;
52                }
53            } catch (Exception e) {
54                Console.WriteLine(e.Message);
55            }
56        }
57
58        return false;
59    }
60
61    public static bool BMATCH(string text, string pattern)
62    {
63        int[] last = BuildLast(pattern);
64        int n = text.Length;
65        int m = pattern.Length;
66        int i = m - 1;
67
68        if (i > n - 1)
69            return false;
70
71        int j = m - 1;
72        do
73        {
74            if (pattern[j] == text[i])
75            {
76                if (j == 0)
77                    return true;
78                else
79                {
80                    i--;
81                    j--;
82                }
83            }
84            else
85            {
86                int lo = last[i]; //last occ
87                i = i + m - Math.Min(j, i + lo);
88                j = m - 1;
89            }
90        } while (i <= n - 1);
91
92        return false;
93    }
94
95    public static int[] BuildLast(string pattern)
96    {
97        int[] last = new int[256]; // ASCII 8-bit char set
98        for (int i = 0; i < 256; i++)
99            last[i] = -1; // initialize array
100        for (int i = 0; i < pattern.Length; i++)
101            last[pattern[i]] = i;
102        return last;
103    }
104 }
105

```

3. HammingDist.cs

```

1 public class HammingDist
2 {
3     public static void FindMatch(string pattern)
4     {
5         Console.WriteLine("Hamming Distance Find Match");
6         DateTime startTime = DateTime.Now;
7
8         List<Siddiklar> siddiklarList = new List<Siddiklar>(Database.SIDIK_1001);
9         List<int> hammingDistanceList = new List<int>();
10
11         int loop = 0;
12         int match100 = 0;
13         foreach (Siddiklar siddiklar in siddiklarList)
14         {
15             int hammingDistance = hamming(ImageConverter.TagPathToString(siddiklar.berkas_citra), pattern, ImageConverter.TagPathToString(siddiklar.berkas_citra).length, pattern.length);
16             hammingDistanceList.Add(hammingDistance);
17
18             int percentage = (int) ((1 - (double)hammingDistance / pattern.length) * 100);
19             if (percentage > 50)
20             {
21                 if (percentage == 100)
22                 {
23                     match100++;
24                 }
25             }
26         }
27
28         int minIndex = 0;
29         for (int i = 0; i < hammingDistanceList.Count; i++)
30         {
31             if (hammingDistanceList[i] < hammingDistanceList[minIndex])
32             {
33                 minIndex = i;
34             }
35         }
36         int percentage = (int) ((1 - (double)hammingDistanceList[minIndex] / pattern.length) * 100);
37
38         foreach (Utils.People biodata in Database.SIDOGALA)
39         {
40             try {
41                 if (MyRegex.Match(biodata.nama, siddiklarList[minIndex].nama)) {
42                     Result_image = Utils.Utils.ConvertToBmp(Encoding.GetEncoding("iso-8859-1").GetBytes(ImageConverter.TagPathToString(siddiklarList[minIndex].berkas_citra)));
43
44                     DateTime endTime = DateTime.Now;
45                     TimeSpan timeDiff = endTime - startTime;
46
47                     Result.timeDiff = timeDiff;
48                     Result.percentage = percentage;
49
50                     Result.createNewPeople(biodata);
51                     Result.setNew(siddiklarList[minIndex].nama);
52
53                     break;
54                 }
55             } catch (Exception e) {
56                 Console.WriteLine(e.Message);
57             }
58         }
59     }
60 }
61
62 public static int hamming(string a, string b, int m, int n)
63 {
64     int countDiff = 0;
65     for (int i = 0; i < Math.Min(m, n); i++)
66     {
67         if (a[i] != b[i])
68         {
69             countDiff++;
70         }
71     }
72
73     return countDiff;
74 }
75 }

```

4. People.cs

```

1 public class People
2 {
3     private string _nik;
4     private string _nama;
5     private string _tempat_lahir;
6     private string _tanggal_lahir;
7     private string _jenis_kelamin;
8     private string _golongan_darah;
9     private string _alamat;
10    private string _agama;
11    private string _status_perkawinan;
12    private string _pekerjaan;
13    private string _kewarganegaraan;
14
15    public People(string nik, string nama, string tempat_lahir,
16        string tanggal_lahir, string jenis_kelamin, string golongan_darah, string alamat, string agama,
17        string status_perkawinan, string pekerjaan, string kewarganegaraan)
18    {
19        _nik = nik;
20        _nama = nama;
21        _tempat_lahir = tempat_lahir;
22        _tanggal_lahir = tanggal_lahir;
23        _jenis_kelamin = jenis_kelamin;
24        _golongan_darah = golongan_darah;
25        _alamat = alamat;
26        _agama = agama;
27        _status_perkawinan = status_perkawinan;
28        _pekerjaan = pekerjaan;
29        _kewarganegaraan = kewarganegaraan;
30    }
31
32    public string Nama
33    {
34        get => _nama;
35        set => _nama = value;
36    }
37
38    public string TempatLahir
39    {
40        get => _tempat_lahir;
41        set => _tempat_lahir = value;
42    }
43
44    public string TanggalLahir
45    {
46        get => _tanggal_lahir;
47        set => _tanggal_lahir = value;
48    }
49
50    public string JenisKelamin
51    {
52        get => _jenis_kelamin;
53        set => _jenis_kelamin = value;
54    }
55
56    public string GolonganDarah
57    {
58        get => _golongan_darah;
59        set => _golongan_darah = value;
60    }
61
62    public string Alamat
63    {
64        get => _alamat;
65        set => _alamat = value;
66    }
67
68    public string Agama
69    {
70        get => _agama;
71        set => _agama = value;
72    }
73
74    public string StatusPerkawinan
75    {
76        get => _status_perkawinan;
77        set => _status_perkawinan = value;
78    }
79
80    public string Pekerjaan
81    {
82        get => _pekerjaan;
83        set => _pekerjaan = value;
84    }
85
86    public string Kewarganegaraan
87    {
88        get => _kewarganegaraan;
89        set => _kewarganegaraan = value;
90    }
91
92    public string Nik => _nik;
93
94    public override string ToString()
95    {
96        return $"NIK: {_nik}\n" +
97            $"Nama: {Nama}\n" +
98            $"Tempat Lahir: {TempatLahir}\n" +
99            $"Tanggal Lahir: {TanggalLahir}\n" +
100            $"Jenis Kelamin: {JenisKelamin}\n" +
101            $"Golongan Darah: {GolonganDarah}\n" +
102            $"Alamat: {Alamat}\n" +
103            $"Agama: {Agama}\n" +
104            $"Status Perkawinan: {StatusPerkawinan}\n" +
105            $"Pekerjaan: {Pekerjaan}\n" +
106            $"Kewarganegaraan: {Kewarganegaraan}";
107    }
108
109    public void print ()
110    {
111        Console.WriteLine($"NIK : " + _nik);
112        Console.WriteLine($"Nama : " + _nama);
113        Console.WriteLine($"Tempat Lahir : " + _tempat_lahir);
114        Console.WriteLine($"Tanggal Lahir : " + _tanggal_lahir);
115        Console.WriteLine($"Jenis Kelamin : " + _jenis_kelamin);
116        Console.WriteLine($"Golongan darah : " + _golongan_darah);
117        Console.WriteLine($"Alamat : " + _alamat);
118        Console.WriteLine($"Agama : " + _agama);
119        Console.WriteLine($"Status Perkawinan : " + _status_perkawinan);
120        Console.WriteLine($"Pekerjaan : " + _pekerjaan);
121        Console.WriteLine($"Kewarganegaraan : " + _kewarganegaraan);
122    }
123 }

```

5. SidikJari.cs

```

1 public struct SidikJari
2 {
3     public string berkas_citra { get; }
4     public string nama { get; }
5
6     public SidikJari(string berkas_citra, string nama)
7     {
8         this.berkas_citra = berkas_citra;
9         this.nama = nama;
10    }
11 }

```

6. Result.cs

```

1 public class Result
2 {
3     public static People _people = new People(
4         nik: "1234567890",
5         nama: "John Doe", tempat_lahir: "Jakarta", tanggal_lahir: "1 Januari 2000",
6         jenis_kelamin: "Laki-laki", golongan_darah: "A", alamat: "Jalan Jalan",
7         agama: "Islam", status_perkawinan: "Belum Kawin", pekerjaan: "PNS",
8         kewarganegaraan: "Indonesia");
9
10    public static Bitmap _image;
11
12    public static TimeSpan timeDiff;
13
14    public static int percentage;
15    public Bitmap Image { get; set; }
16    public string Text { get; set; }
17    public People People { get; set; }
18
19    public Result(Bitmap image, string text, People people)
20    {
21        Image = image;
22        Text = text;
23        People = people;
24    }
25
26    public Result(Bitmap image, string text)
27    {
28        Image = image;
29        Text = text;
30    }
31
32    public Result(string text)
33    {
34        Text = text;
35    }
36
37    public static void createNewPeople(People people)
38    {
39        _people = people;
40    }
41
42    public static void setName(String nama)
43    {
44        _people.Nama = nama;
45    }
46
47    public static void setPercentage(int percentage)
48    {
49        Result.percentage = percentage;
50    }
51
52    public static void print()
53    {
54        Console.WriteLine("Nama: " + _people.Nama);
55        Console.WriteLine("Tempat Lahir: " + _people.Tempatlahir);
56        Console.WriteLine("Tanggal Lahir: " + _people.Tanggalahir);
57    }
58 }

```

7. Database.cs

```

1 public class Database
2 {
3     public static List<People> BIODATA = new List<People>();
4     public static List<SidikJari> SIDIK_JARI = new List<SidikJari>();
5
6     public static void Load()
7     {
8         string connStr = "Server=localhost;Database=tubes3;User=root;Password=password";
9         using var cn = new MySqlConnection(connStr);
10        cn.Open();
11
12        LoadBiodata(cn);
13        LoadSidikJari(cn);
14    }
15
16    private static void LoadBiodata(MySqlConnection cn)
17    {
18        Console.WriteLine("Loading Biodata...");
19        BIODATA.Clear();
20
21        string query = "SELECT * FROM biodata";
22        using var cmd = new MySqlCommand(query, cn);
23        using var reader = cmd.ExecuteReader();
24
25        while (reader.Read())
26        {
27            People biodata = new People(reader["NIK"].ToString(), reader["nama"].ToString(),
28            reader["tempat_lahir"].ToString(), reader["tanggal_lahir"].ToString(), reader["jenis_kelamin"].ToString(),
29            reader["golongan_darah"].ToString(), reader["alamat"].ToString(), reader["agama"].ToString(),
30            reader["status_perkawinan"].ToString(), reader["pekerjaan"].ToString(), reader["kewarganegaraan"].ToString());
31            BIODATA.Add(biodata);
32        }
33
34        Console.WriteLine("Biodata loaded! (" + BIODATA.Count + ")");
35    }
36
37    private static void LoadSidikJari(MySqlConnection cn)
38    {
39        Console.WriteLine("Loading SidikJari...");
40        SIDIK_JARI.Clear();
41
42        string query = "SELECT * FROM sidik_jari";
43        using var cmd = new MySqlCommand(query, cn);
44        using var reader = cmd.ExecuteReader();
45
46        while (reader.Read())
47        {
48            SidikJari sidikJari = new SidikJari(reader["berkas_citra"].ToString(), reader["nama"].ToString());
49            SIDIK_JARI.Add(sidikJari);
50        }
51        Console.WriteLine("SidikJari loaded! (" + SIDIK_JARI.Count + ")");
52    }
53 }

```

8. ImageConverter.cs

```

1 public class ImageConverter
2 {
3     public static byte[] PreprocessImage(string imagePath)
4     {
5         // Load image
6         // Console.WriteLine(imagePath);
7         // console current directory
8         imagePath = imagePath.Replace("file://", "");
9         Mat image = CvInvoke.Imread(imagePath);
10
11         // Convert image to grayscale
12         Mat gray = new Mat();
13         CvInvoke.CvtColor(image, gray, ColorConversion.Bgr2Gray);
14
15         // Resize to 96x103
16         Mat resized = new Mat();
17         CvInvoke.Resize(gray, resized, new Size(96, 103));
18
19         // Convert to binary
20         Mat binary = new Mat();
21         CvInvoke.Threshold(resized, binary, 127, 255, ThresholdType.Binary);
22
23         // Save binary image
24         CvInvoke.Imwrite("binary.BMP", binary);
25
26         // Read all bytes from binary image
27         byte[] binaryData = File.ReadAllBytes("binary.BMP");
28
29
30         return binaryData;
31     }
32
33     // black and white img path to binary string
34     public static string ImgPathToString(string imagePath)
35     {
36
37         byte[] binaryData = File.ReadAllBytes(imagePath);
38         String encoding = Encoding.GetEncoding("iso-8859-1").GetString(binaryData);
39
40         return encoding;
41     }
42 }

```

9. MyRegex.cs

```

1 public class MyRegex
2 {
3     private static List<char> vocals = new List<char> {'a', 'i', 'u', 'e', 'o'};
4
5     private static Dictionary<char, char> alay = new Dictionary<char, char>()
6     {
7         { 'o', '0' },
8         { 'i', '1' },
9         { 'z', '2' },
10        { 'e', '3' },
11        { 'a', '4' },
12        { 's', '5' },
13        { 'g', '6' },
14    };
15
16
17    public static bool match(string text1, string text2)
18    {
19        // Console.WriteLine(text1);
20        // Console.WriteLine(text2);
21
22        Regex nameRegex = new Regex(getPattern(text2));
23        Match match = nameRegex.Match(text1);
24
25        return match.Success;
26    }
27
28    public static string getPattern(string text)
29    {
30        string pattern = @"\b";
31
32        foreach (char c in text)
33        {
34            pattern += "[";
35            if (char.IsLetter(c))
36            {
37                pattern += char.ToLower(c);
38                pattern += char.ToUpper(c);
39            }
40
41            if (alay.ContainsKey(char.ToLower(c)))
42            {
43                pattern += alay[char.ToLower(c)];
44            }
45
46            if (c == ' ')
47            {
48                pattern += @"\s";
49            }
50
51            pattern += "]";
52
53            if (vocals.Contains(char.ToLower(c)))
54            {
55                pattern += "?";
56            }
57        }
58
59        pattern += @"\b";
60
61        // Console.WriteLine(text);
62        // Console.WriteLine(pattern);
63        return @pattern;
64    }
65 }

```

```

1  public class Utils
2  {
3      public static byte[] ConvertToBinary(string filePath)
4      {
5          string path = filePath.Replace("file:///", "");
6
7          // if (path[0] != '/')
8          // {
9          //     path = "/" + path;
10         // }
11
12         byte[] b = File.ReadAllBytes(path);
13
14         return b;
15     }
16
17     public static Bitmap ConvertToBitmap(byte[] data)
18     {
19         using (MemoryStream ms = new MemoryStream(data))
20         {
21             return new Bitmap(ms);
22         }
23     }
24 }

```

4.3 Penjelasan Tata Cara Penggunaan Program

Berikut adalah tata cara penggunaan aplikasi yang telah kami buat :

1. **Memulai Program** : Setelah program dijalankan, akan muncul window yang menampilkan halaman utama (home page).

2. **Navigasi dengan Navbar** :

- Navbar terletak di pojok kiri atas window, terdapat navbar dengan 4 tombol berurutan ke bawah:
- **Button Paling Atas** : Menunjukkan nama-nama tiap tombol pada navbar jika ditekan
- **Button Kedua** : Mengarah ke halaman utama (home page).
- **Button Ketiga** : Mengarah ke halaman solver.
- **Button Paling Bawah** : Mengarah ke halaman tentang kami (about us).

3. **Mencari Biodata dari Gambar** :

- **Mengakses Solver Page** : Untuk mencari biodata dari gambar sidik jari yang dimiliki pengguna, klik button ketiga di navbar.
- **Solver Page** : Setelah button ketiga ditekan, window akan menampilkan solver page. Halaman ini terdiri dari:
 - **Kotak** yang memiliki efek transparan untuk menampilkan gambar yang ingin dicari.
 - **Button** untuk memilih algoritma yang ingin digunakan.
 - **Tombol `Search`** untuk memulai pencarian.

4. **Memilih Gambar** :

Pengguna dapat melakukannya dengan menekan kotak transparan di layar. Ini akan mengarahkan pengguna untuk memilih file gambar sidik jari (format yang didukung: jpg, jpeg, bitmap, dan png).

5. **Memilih Algoritma** :

Pengguna dapat melakukannya dengan menekan button lingkaran di sebelah tulisan KMP atau BM untuk memilih algoritma yang ingin digunakan.

6. Melakukan Pencarian :

Pengguna dapat menekan tombol 'Search' untuk memulai pencarian. Kemudian, nantinya program akan menampilkan pop-up hasil pencarian kepada pengguna.

7. Melihat dan Menutup Hasil :

Pengguna dapat melihat hasil yang ditampilkan pada pop-up. Pengguna dapat menekan tombol 'X' pada pop-up setelah selesai melihat hasil, yang akan mengembalikan pengguna ke halaman solver.

8. Mengulangi Pencarian :

Untuk mengulangi pencarian dengan gambar yang berbeda, pengguna dapat mengulangi langkah 5 hingga 7.

9. Melihat About Us Page :

Pengguna dapat melihat halaman tentang kami dengan menekan button keempat (paling bawah) di navbar yang terletak pada pojok kiri atas window. Window akan menampilkan about us page.

10. Mengakhiri Program :

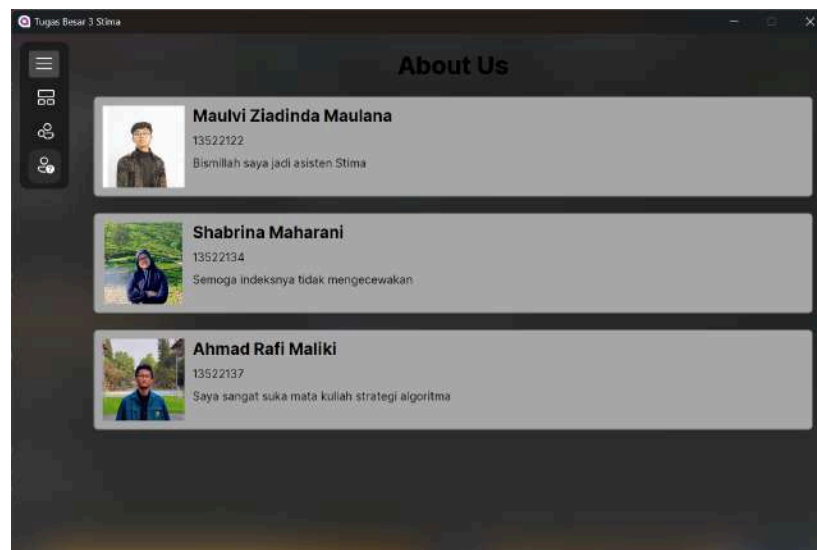
Setelah selesai menggunakan program, pengguna dapat menekan tombol 'X' di pojok kanan atas untuk menutup program

4.4 Hasil Pengujian

4.2.1 Home Page



4.2.2 About Us Page



4.2.3 Pengujian KMP

4.2.3.1 Real

Real merupakan gambar sidik jari yang sama persis dengan gambar sidik jari yang terdapat pada basis data.

Test Case 1



Test Case 2



4.2.3.2 Altered-Easy

Altered-Easy merupakan gambar sidik jari yang terdapat pada basis data. namun sedikit dirusak.

Test Case 1



Test Case 2



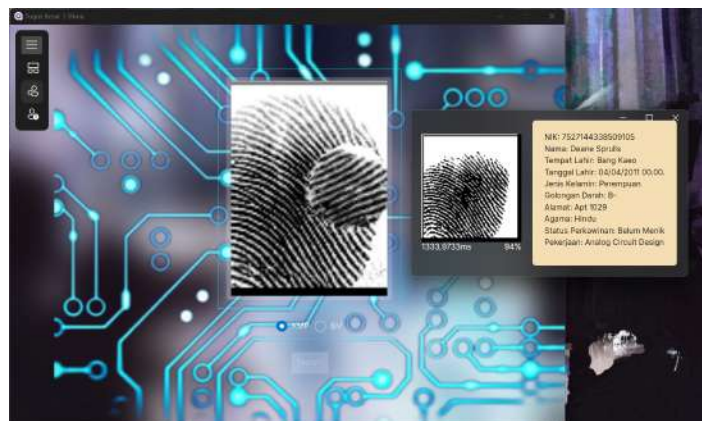
4.2.3.3 Altered-Medium

Altered-Medium merupakan gambar sidik jari yang terdapat pada basis data namun dirusak.

Test Case 1



Test Case 2



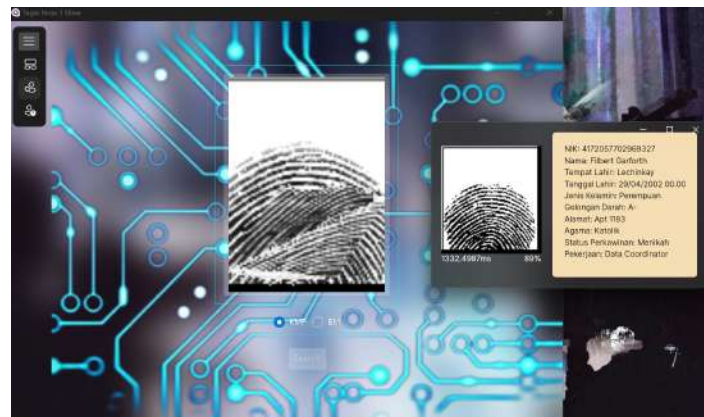
4.2.3.4 Altered-Hard

Altered-Hard merupakan gambar sidik jari yang terdapat pada basis data namun sangat dirusak.

Test Case 1



Test Case 2



4.2.4 Pengujian Boyer Moore

4.2.4.1 Real

Real merupakan gambar sidik jari yang sama persis dengan gambar sidik jari yang terdapat pada basis data.

Test Case 1



Test Case 2



4.2.4.2 Altered-Easy

Altered-Easy merupakan gambar sidik jari yang terdapat pada basis data. namun sedikit dirusak.

Test Case 1



Test Case 2



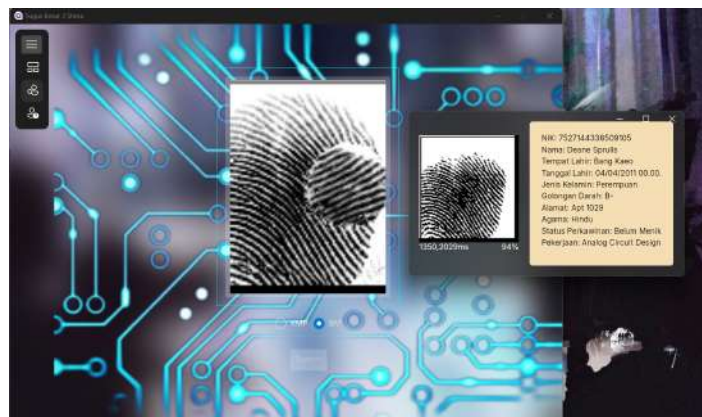
4.2.4.3 Altered-Medium

Altered-Medium merupakan gambar sidik jari yang terdapat pada basis data namun dirusak.

Test Case 1



Test Case 2



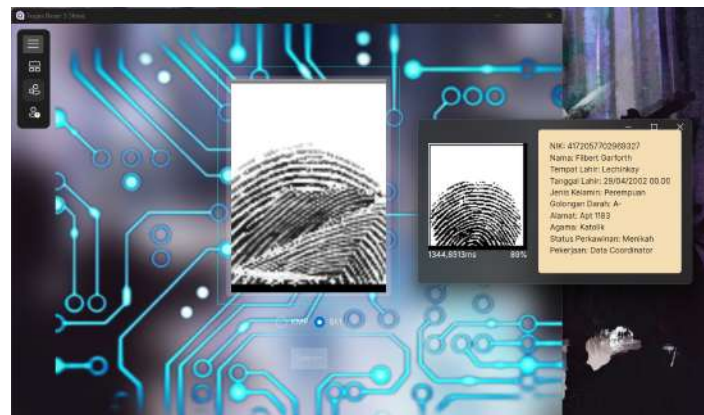
4.2.4.4 Altered-Hard

Altered-Hard merupakan gambar sidik jari yang terdapat pada basis data namun sangat dirusak.

Test Case 1



Test Case 2



4.5 Analisis Hasil Pengujian

Berdasarkan hasil pengujian yang telah dilakukan didapatkan data berupa persentase kemiripan gambar sidik jari yang dicari dengan gambar sidik jari yang terdapat dalam database dan waktu eksekusi untuk mendapatkan solusi sebagai berikut.

Kategori Pengujian	TC	KMP				BM			
		Persentase (%)	Rata-rata Persentase	Waktu Eksekusi (ms)	Rata-rata Waktu Eksekusi (ms)	Persentase (%)	Rata-rata Persentase	Waktu Eksekusi (ms)	Rata-rata Waktu Eksekusi (ms)
Real	TC	100	100	918.8165	946.71435	100	100	1000.2283	1110.71735

	1								
	TC 2	100		974.6122		100		1221.2064	
Altered-Easy	TC 1	96	96.5	873.9327	963.24915	96	96.5	940.7189	945.6379
	TC 2	97		1052.5656		97		950.5569	
Altered-Medium	TC 1	91	92.5	1360.2853	1347.1293	91	92.5	1371.0894	1360.64615
	TC 2	94		1333.9733		94		1350.2029	
Altered-Hard	TC 1	90	89.5	1285.3541	1308.9264	90	89.5	1318.0194	1331.43535
	TC 2	89		1332.4987		89		1344.8513	
Rata-Rata	Persentase		94.625	Waktu Eksekusi	1141.504575	Persentase	94.625	Waktu Eksekusi	1187.1092

Dari tabel di atas, terlihat bahwa algoritma Boyer-Moore membutuhkan rata-rata waktu sedikit lebih lama sekitar 40 ms untuk mendapatkan solusi dari pencarian gambar sidik jari. Hal ini tidak terlalu memiliki perbedaan yang signifikan antara algoritma KMP dan BM dalam hal waktu eksekusi. Perbedaan ini dapat terjadi karena faktor kinerja device ataupun faktor lainnya di luar algoritma. Namun, kedua algoritma menunjukkan persentase keberhasilan yang sangat tinggi, yaitu 94.625% untuk berbagai kategori pengujian.

Algoritma Boyer-Moore menunjukkan keunggulannya dalam kasus ketika alfabet yang digunakan besar, yang mana sering terjadi dalam teks bahasa Inggris, namun algoritma ini menjadi lebih lambat ketika alfabet kecil, seperti dalam teks biner. Kompleksitas waktu terburuk untuk Boyer-Moore adalah $O(nm + A)$, dimana A adalah ukuran alfabet. Di sisi lain, algoritma KMP memiliki kompleksitas waktu $O(m + n)$, dimana m adalah panjang pola dan n adalah panjang teks. KMP lebih stabil dalam hal performa waktu karena tidak terpengaruh oleh ukuran alfabet.

Dari analisis ini, dapat disimpulkan bahwa meskipun Boyer-Moore secara teoritis bisa lebih cepat dalam situasi tertentu, dalam konteks pengujian citra sidik jari, KMP menawarkan kecepatan yang lebih konsisten dan sedikit lebih cepat dalam kebanyakan kasus. Kedua algoritma tetap efektif dan efisien untuk pencocokan pola sidik jari, dan pemilihan antara keduanya dapat bergantung pada spesifikasinya dan konteks penggunaan yang lebih luas.

BAB 5 KESIMPULAN DAN SARAN

5.1 Kesimpulan

Dalam proyek ini, kami berhasil mengimplementasikan aplikasi desktop berbasis biometrik yang menggunakan algoritma pattern matching untuk mengidentifikasi sidik jari. Dengan menggunakan bahasa pemrograman C#, framework Avalonia, dan arsitektur MVVM (Model-View-ViewModel), aplikasi ini menawarkan performa yang baik, fleksibilitas tinggi, dan kemudahan dalam pengelolaan serta pengembangan. Penggunaan algoritma Boyer-Moore dan Knuth-Morris-Pratt (KMP) mendukung fungsi pencocokan pola yang efisien. Hasil pengujian menunjukkan bahwa kedua algoritma ini mampu melakukan pencocokan pola dengan tingkat akurasi yang tinggi, meskipun terdapat beberapa kasus di mana optimisasi lebih lanjut diperlukan. Aplikasi ini berhasil

memenuhi tujuan utamanya yaitu menyediakan solusi identifikasi sidik jari yang cepat dan akurat, yang dapat diintegrasikan dalam berbagai sistem keamanan.

5.2 Saran

Untuk pengembangan lebih lanjut, kami menyarankan beberapa peningkatan seperti fitur antarmuka pengguna (UI) dapat ditingkatkan dengan menambahkan elemen interaktif yang lebih responsif dan intuitif untuk meningkatkan pengalaman pengguna. Selain itu, eksplorasi lebih lanjut terhadap algoritma lain yang lebih modern dan adaptif dapat membantu dalam meningkatkan efisiensi dan akurasi sistem.

5.3 Refleksi

Selama proses pengembangan aplikasi ini, kami belajar banyak tentang pentingnya desain arsitektur yang baik dan penggunaan framework yang tepat. Penggunaan Avalonia dan arsitektur MVVM memberikan kami wawasan tentang bagaimana menjaga pemisahan yang jelas antara logika bisnis dan tampilan, yang pada akhirnya mempermudah proses pengembangan dan pemeliharaan aplikasi. Kami juga memahami pentingnya optimisasi algoritma dalam konteks pemrosesan data yang besar dan kompleks. Meskipun kami menghadapi beberapa tantangan, seperti penanganan data sidik jari yang bervariasi, melalui kolaborasi dan eksplorasi, kami berhasil mengatasinya dan mencapai hasil yang memuaskan. Pengalaman ini memperkaya pemahaman kami tentang pengembangan aplikasi berbasis biometrik dan memberikan dasar yang kuat untuk proyek-proyek di masa mendatang .

LAMPIRAN

1. **Link Repository Github** : https://github.com/rafimaliki/Tubes3_mamama
2. **Link Video** : https://youtu.be/Zr-F7fUz-_U

DAFTAR PUSTAKA

Rinaldi Munir. 2021. "Pencocokan String (String/Pattern Matching)"

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

Rinaldi Munir. 2019. "String Matching dengan Regular Expression"

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

Rinaldi Munir. 2020. "Modul Praktikum Kuliah : Pengantar Regular Expression "

<https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2019-2020/Modul-Praktikum-NLP-Regex.pdf>

AvaloniaUI OÜ . 2024. "Avalonia UI Documentation"

<https://docs.avaloniaui.net/docs/welcome>

Munir, Rinaldi dan Amir Muntaha. 2010. "Pengenal Sidik Jari dengan Menggunakan Algoritma Pencocokan String Boyer-Moore ".

<https://informatika.stei.itb.ac.id/~rinaldi.munir/Penelitian/Makalah-KNIF-2010.pdf>

GeeksForGeeks . 2023. "Basic Database Operations Using C#"

<https://www.geeksforgeeks.org/basic-database-operations-using-c-sharp/>

“Bahasa Alay Generator”

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