

Tugas Besar 3 IF2211 Strategi Algoritma

Semester II tahun 2023/2024

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



Disusun oleh:

Christian Justin Hendrawan 13522122

Farhan Raditya Aji 13522142

Mohammad Akmal Ramadhan 13522161

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

## DAFTAR ISI

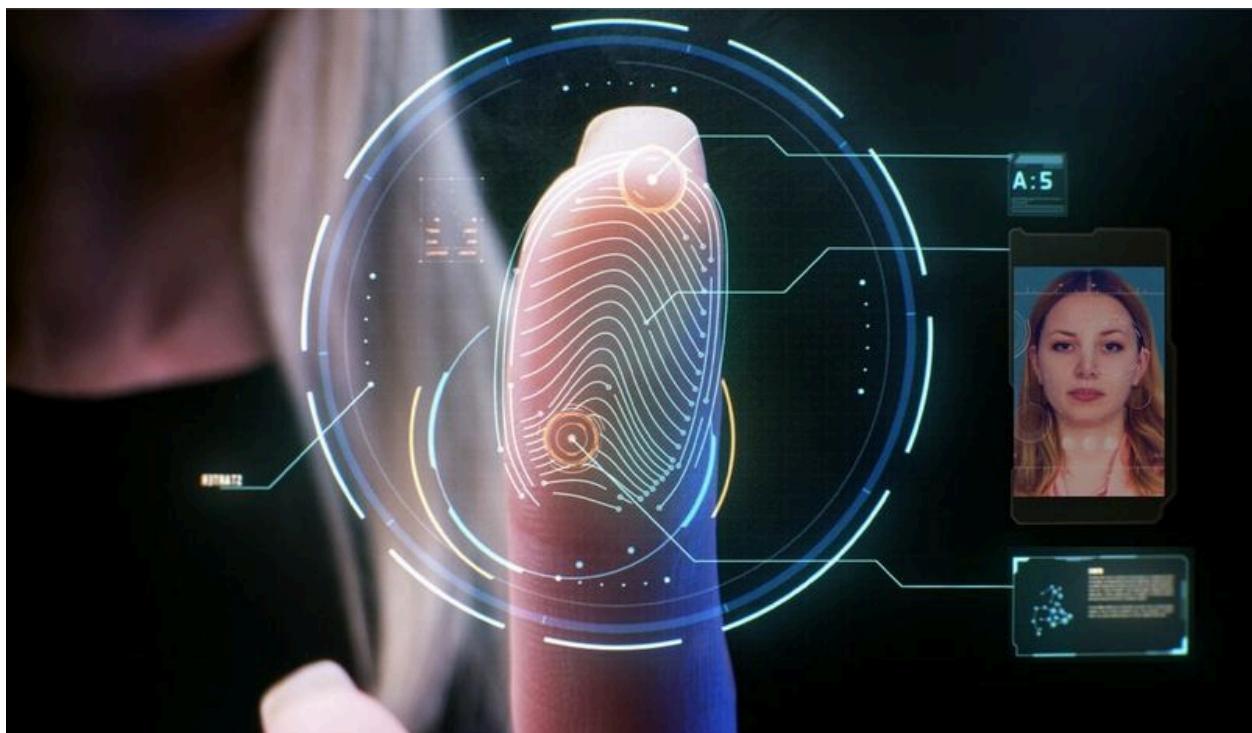
DAFTAR ISI.....	2
BAB 1	
DESKRIPSI TUGAS.....	4
BAB 2	
LANDASAN TEORI.....	6
2.1 Dasar Teori.....	6
2.1.1 Knuth-Morris-Pratt.....	6
2.1.2 Boyer-Moore.....	7
2.1.3 Regular Expression.....	8
2.1.4 Metode Pengukuran Tingkat Kemiripan.....	10
2.2 Aplikasi Desktop.....	11
2.2.1 Windows Forms (WinForms).....	11
BAB 3	
ANALISIS PEMECAHAN MASALAH.....	13
3.1 Langkah - Langkah Pemecahan Masalah.....	13
3.1.1 Fitur String Matching dengan Algoritma KMP.....	13
3.1.2 Fitur String Matching dengan Algoritma Boyer-Moore.....	15
3.1.3 Sistem Deteksi Melalui Citra Sidik Jari.....	16
3.2 Fitur Fungsional dan Arsitektur Aplikasi Desktop Yang Dibangun.....	18
3.2.1 Fitur Fungsional.....	18
3.2.2 Arsitektur Aplikasi.....	19
3.3 Ilustrasi Kasus.....	21
BAB 4	
IMPLEMENTASI DAN PENGUJIAN.....	23
4.1 Spesifikasi Teknis Program.....	23
4.1.1 KnuthMorrisPratt.cs.....	23
4.1.2 BoyerMoore.cs.....	23
4.1.3 Database.cs.....	24
4.1.4 AlayConverter.cs.....	25
4.1.5 Form1.cs.....	25
4.2 Tata Cara Penggunaan Program.....	26
4.3 Hasil Pengujian.....	27
4.3.1 Kasus 1.....	27
4.3.2 Kasus 2.....	28
4.3.3 Kasus 3.....	29
4.3.4 Kasus 4.....	29

4.4 Analisis Hasil.....	30
4.4.1 Analisis Keakuratan.....	30
4.4.2 Analisis Optimalitas.....	31
4.4.3 Hasil Analisis.....	32
BAB 5	
KESIMPULAN DAN SARAN.....	33
5.1 Kesimpulan.....	33
5.2 Saran.....	33
5.3 Refleksi.....	34
LAMPIRAN.....	36
DAFTAR PUSTAKA.....	37

## BAB 1

### DESKRIPSI TUGAS

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



Gambar 1. Ilustrasi fingerprint recognition pada deteksi berbasis biometrik.

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

Pattern matching merupakan teknik penting dalam sistem identifikasi sidik jari. Teknik ini digunakan untuk mencocokkan pola sidik jari yang ditangkap dengan pola sidik jari yang terdaftar di database. Algoritma pattern matching yang umum digunakan adalah Bozorth dan

Boyer-Moore. Algoritma ini memungkinkan sistem untuk mengenali sidik jari dengan cepat dan akurat, bahkan jika sidik jari yang ditangkap tidak sempurna.

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

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

## BAB 2

### LANDASAN TEORI

#### 2.1 Dasar Teori

##### 2.1.1 Knuth-Morris-Pratt

Algoritma Knuth-Morris-Pratt (KMP) adalah algoritma yang digunakan untuk mencari keberadaan suatu string (pattern) dalam string lainnya (text). Algoritma ini dinamakan berdasarkan penemunya, yaitu Donald Knuth, Vaughan Pratt, dan James H. Morris.

Algoritma ini memiliki keunggulan dibandingkan dengan algoritma pencarian string sederhana, yaitu efisiensi waktu. Algoritma pencarian string sederhana akan kembali ke awal pattern dan maju satu karakter di text ketika terjadi ketidakcocokan. Sedangkan algoritma KMP, ketika terjadi ketidakcocokan, akan memajukan pencarian di text tanpa mengulangi pengecekan pada karakter yang telah dicek.

Algoritma KMP menggunakan konsep prefix dan suffix dalam pattern. Prefix adalah substring yang diambil dari awal string, dan suffix adalah substring yang diambil dari akhir string. Algoritma ini memanfaatkan fakta bahwa ketika terjadi ketidakcocokan, kita tidak perlu memeriksa ulang karakter yang telah dicek. Kita hanya perlu memeriksa prefix dan suffix dalam pattern yang telah dicek.

Untuk mencapai efisiensi ini, algoritma KMP menggunakan struktur data tambahan berupa LPS array (Longest Proper Prefix which is also suffix). LPS array berisi panjang maksimum dari prefix yang juga merupakan suffix. Nilai  $LPS[i]$  adalah panjang prefix terpanjang yang juga merupakan suffix dari pattern hingga indeks  $i$ . Dengan menggunakan LPS array, algoritma KMP dapat memajukan pencarian di text tanpa mengulangi pengecekan pada karakter yang telah dicek. Langkah-langkah untuk mengimplementasikan algoritma KMP adalah sebagai berikut.

1. Buat LPS array dari pattern. LPS array berisi panjang maksimum dari prefix yang juga merupakan suffix. Nilai  $LPS[i]$  adalah panjang prefix terpanjang yang juga merupakan suffix dari pattern hingga indeks  $i$ .

2. Mulai mencocokkan pattern dan text. Jika karakter pattern dan text sama, maju ke karakter berikutnya pada pattern dan text.
3. Jika karakter pattern dan text tidak sama, lakukan hal berikut:
  - a. Jika indeks pattern tidak nol, kembali ke karakter sebelumnya pada pattern menggunakan LPS array. Ini tidak mengubah posisi karakter pada text.
  - b. Jika indeks pattern nol, maju ke karakter berikutnya pada text.

### 2.1.2 Boyer-Moore

Algoritma Boyer-Moore adalah algoritma pencarian string yang efisien, yang dirancang untuk mencari suatu string (pattern) dalam string lainnya (text). Algoritma ini dinamakan berdasarkan penemunya, yaitu Robert S. Boyer dan J Strother Moore. Keunggulan utama dari algoritma Boyer-Moore adalah efisiensi waktu. Algoritma ini mencapai efisiensi ini dengan melakukan pengecekan dari akhir pattern dan sering kali "melompat" lebih jauh dalam text dibandingkan algoritma pencarian string lainnya.

Algoritma Boyer-Moore menggunakan dua aturan heuristik, yaitu Bad Character Heuristic dan Good Suffix Heuristic. Bad Character Heuristic digunakan ketika karakter yang sedang dicek tidak cocok. Algoritma ini akan melihat tabel Bad Character Heuristic dan melompat sejauh jarak dari karakter terakhir pattern ke lokasi karakter tersebut terakhir kali muncul dalam pattern.

Good Suffix Heuristic digunakan ketika ada suffix yang cocok dalam pattern. Algoritma ini akan melihat tabel Good Suffix Heuristic dan melompat sejauh jarak yang ditentukan dalam tabel tersebut. Dengan menggunakan kedua aturan heuristik ini, algoritma Boyer-Moore dapat melompat lebih jauh dalam text dan menghindari pengecekan ulang pada karakter yang telah dicek, sehingga mencapai efisiensi waktu yang lebih baik.

Langkah-langkah dalam menerapkan algoritma Boyer-Moore adalah sebagai berikut.

1. Buat tabel Bad Character Heuristic. Tabel ini berisi jarak dari karakter terakhir pattern ke lokasi karakter tersebut terakhir kali muncul dalam pattern.

2. Buat tabel Good Suffix Heuristic. Tabel ini berisi jarak lompatan jika ada suffix yang cocok dalam pattern.
3. Mulai mencocokkan pattern dan text dari karakter terakhir pattern. Jika karakter pattern dan text sama, maju mundur ke karakter sebelumnya pada pattern dan text.
4. Jika karakter pattern dan text tidak sama, atau telah mencapai awal pattern dan semua karakter cocok, lakukan hal berikut:
  - a. Hitung jarak lompatan berdasarkan tabel Bad Character Heuristic dan Good Suffix Heuristic.
  - b. Pilih jarak lompatan terjauh dan maju sejauh itu dalam text.

Algoritma Boyer-Moore sangat efisien, terutama pada teks yang panjang dan pattern yang pendek, karena sering kali dapat melompat lebih jauh dibandingkan algoritma pencarian string lainnya

### 2.1.3 Regular Expression

Regular Expression (Regex) adalah notasi yang digunakan untuk mencocokkan pola dalam teks. Regex digunakan dalam operasi pencarian dan penggantian string, serta manipulasi string. Regex sangat populer dan tersedia di hampir semua bahasa pemrograman modern seperti JavaScript, Python, Java, dan lainnya. Berikut adalah beberapa komponen utama dalam Regex

- a. Konstanta: Dalam konteks RegEx, konstanta adalah karakter literal atau kelas karakter. Karakter literal mencocokkan dirinya sendiri, sedangkan kelas karakter mencocokkan setiap karakter dalam kelas tersebut. Misalnya, regex a akan mencocokkan semua 'a' dalam teks, dan [abc] akan mencocokkan karakter 'a', 'b', atau 'c'.
- b. Operator: Operator digunakan untuk mengubah cara kerja konstanta. Beberapa operator utama dalam Regex adalah:
  - Concatenation: Menggabungkan dua konstanta bersama-sama. Misalnya, ab akan mencocokkan 'a' diikuti oleh 'b'.

- Alternation (|): Mencocokkan konstanta ini atau itu. Misalnya,  $a|b$  akan mencocokkan 'a' atau 'b'.
- Quantifiers (\*, +, ?, {n}, {n,}, {n,m}): Menentukan seberapa sering konstanta dapat muncul. Misalnya,  $a^*$  akan mencocokkan 'a' nol atau lebih kali.
- Grouping (): Mengelompokkan konstanta dan operator bersama-sama. Misalnya,  $(ab)^*$  akan mencocokkan 'ab' nol atau lebih kali.
- Anchors (^, \$): Menentukan posisi konstanta dalam string. Misalnya,  $^a$  akan mencocokkan karakter 'a' di awal string, dan  $a$$  akan mencocokkan 'a' di akhir string.

Langkah-langkah dalam menggunakan Regex :

- a. Tentukan Konstanta: Pertama, tentukan konstanta yang ingin Anda cari. Ini bisa berupa karakter literal atau kelas karakter. Misalnya, jika Anda mencari huruf 'a', maka konstanta Anda adalah  $a$ . Jika Anda mencari angka, Anda bisa menggunakan kelas karakter  $\d$ .
- b. Gunakan Operator: Selanjutnya, gunakan operator untuk mengubah cara kerja konstanta. Misalnya, jika Anda mencari huruf 'a' di awal string, Anda bisa menggunakan anchor  $^$  untuk menciptakan regex  $^a$ . Jika Anda mencari satu atau lebih 'a', Anda bisa menggunakan quantifier  $+$  untuk menciptakan regex  $a^+$ .
- c. Gabungkan Konstanta dan Operator: Anda bisa menggabungkan beberapa konstanta dan operator untuk menciptakan pola pencarian yang lebih kompleks. Misalnya, jika Anda mencari string yang dimulai dengan satu atau lebih 'a' diikuti oleh satu atau lebih 'b', Anda bisa menciptakan regex  $^a+b^+$ .
- d. Gunakan Grouping jika Diperlukan: Jika Anda ingin mengelompokkan beberapa konstanta dan operator bersama-sama, Anda bisa menggunakan tanda kurung  $()$ . Misalnya, jika Anda mencari string yang dimulai dengan 'ab' diulang satu atau lebih kali, Anda bisa menciptakan regex  $^((ab)+)$ .
- e. Terapkan Regex: Terakhir, terapkan regex Anda ke teks yang ingin Anda cari. Cara ini akan berbeda-beda tergantung pada bahasa pemrograman yang Anda

gunakan, tetapi biasanya melibatkan pemanggilan fungsi seperti `match()`, `search()`, atau `findall()`.

Regex yang diimplementasikan dalam program ini melakukan handle untuk angka dan huruf kapital, berikut ketentuannya mengurut sesuai prioritas:

1. Jika huruf pada suatu kata kapital semua, jangan ubah menjadi nonkapital
2. Apabila pada satu kata hanya ada huruf, jangan ubah menjadi kata
3. Apabila ada kapital pada awal kata, jangan ubah menjadi nonkapital
4. Ubah semua kapital menjadi nonkapital
5. ubah semua angka menjadi huruf

#### 2.1.4 Metode Pengukuran Tingkat Kemiripan

Metode perhitungan yang digunakan untuk mencari tingkat kemiripan dari 2 buah fingerprint adalah dengan menggunakan Levenshtein Distance. Levenshtein Distance, juga dikenal sebagai jarak edit, adalah teknik untuk mengukur seberapa mirip dua string. Ini dihitung sebagai jumlah minimum operasi yang diperlukan untuk mengubah satu string menjadi string lainnya. Operasi yang dihitung dalam Levenshtein Distance adalah penambahan, penghapusan, atau penggantian karakter.

Misalnya, jika kita memiliki dua string: "kucing" dan "anjing", Jarak Levenshtein antara keduanya adalah 3, karena kita perlu mengganti karakter pertama "k" dengan "a" dan karakter kedua "u" dengan "j" dan karakter ketiga "c" menjadi "j" untuk mengubah "kucing" menjadi "anjing". Untuk menghitung persentase kemiripan antara dua string menggunakan Jarak Levenshtein, kita bisa menggunakan rumus berikut:

Persentase Kemiripan :

$$(1 - (\text{Jarak Levenshtein} / \text{Panjang Maksimum dari Dua String})) * 100\%$$

Dengan menggunakan rumus ini, semakin tinggi Jarak Levenshtein, semakin rendah persentase kemiripannya, dan sebaliknya. Jadi, jika dua string identik, Jarak Levenshtein akan 0, dan persentase kemiripannya akan 100%. Jika dua string tidak

memiliki karakter yang sama, Jarak Levenshtein akan sama dengan panjang string, dan persentase kemiripannya akan 0%.

Nilai batas yang diambil untuk mencari sidik jari paling mirip adalah di atas 60%. Karena 60% seringkali dianggap cukup baik untuk banyak kasus, termasuk pencocokan sidik jari. Dalam proses pengambilan sidik jari, bisa terjadi kesalahan atau variasi. Misalnya, posisi jari yang sedikit berbeda atau tekanan yang tidak konsisten saat menempatkan jari dapat menghasilkan gambar sidik jari yang sedikit berbeda. Dengan ambang batas 60% ke atas, kita masih dapat mengidentifikasi sidik jari yang sama meskipun ada variasi kecil dalam pengukuran. Ambang 60% juga memastikan bahwa hanya sidik jari yang sangat mirip (dan oleh karena itu kemungkinan besar dari jari yang sama) yang dianggap cocok.

## 2.2 Aplikasi Desktop

Aplikasi desktop adalah perangkat lunak yang biasanya diinstal secara lokal pada komputer desktop atau laptop dan memiliki antarmuka pengguna yang memungkinkan pengguna berinteraksi dengan program melalui elemen grafis seperti menu, jendela, dan tombol. Aplikasi desktop ini dibuat dengan menggunakan Windows Forms dengan bahasa pemrograman C#.

### 2.2.1 Windows Forms (WinForms)

WinForms adalah pustaka antarmuka pengguna (UI) yang disediakan oleh Framework.NET dan digunakan untuk membuat aplikasi desktop di sistem operasi Windows. WinForms memiliki berbagai kontrol dan komponen yang memungkinkan pengembang membuat antarmuka pengguna yang kaya dan responsif. Beberapa komponen utama WinForms adalah sebagai berikut:

- **Form**

Form adalah dasar dari aplikasi WinForms yang mewakili jendela dalam aplikasi. Setiap form dapat memiliki elemen UI lainnya seperti tombol, kotak teks, dan label. Form ini juga berfungsi sebagai wadah utama bagi komponen dan kontrol lainnya dalam aplikasi. Form memiliki berbagai properti seperti ukuran, posisi, dan warna latar belakang yang dapat dikustomisasi sesuai kebutuhan aplikasi.

- **Controls**

WinForms menyediakan berbagai kontrol seperti Button, TextBox, Label, PictureBox, dan lain-lain yang dapat digunakan untuk membangun antarmuka pengguna. Kontrol ini memungkinkan pengguna untuk berinteraksi dengan aplikasi, seperti memasukkan teks, menekan tombol, dan menampilkan informasi. Kontrol ini juga dapat dikustomisasi dan diatur melalui properti dan metode yang disediakan oleh WinForms.

- **Events**

WinForms menggunakan model berbasis event di mana tindakan pengguna seperti klik tombol atau input teks dapat ditangani melalui event handler. Event handler adalah metode yang akan dijalankan ketika suatu event terjadi. Misalnya, ketika pengguna mengklik sebuah tombol, event handler untuk event klik tombol tersebut akan dipanggil. Dengan menggunakan event, aplikasi dapat merespons interaksi pengguna dengan cara yang dinamis dan fleksibel.

WinForms juga menyediakan fitur-fitur tambahan seperti dukungan untuk drag-and-drop, tata letak otomatis, dan integrasi dengan database. Fitur-fitur ini memudahkan pengembangan aplikasi desktop yang kompleks dan memungkinkan pengembang untuk fokus pada logika bisnis dan fungsi utama dari aplikasi.

Selain itu, WinForms mendukung penggunaan kontrol kustom dan pustaka pihak ketiga, yang memungkinkan pengembang untuk memperluas fungsionalitas aplikasi mereka di luar kontrol dan komponen standar yang disediakan oleh WinForms. Dengan demikian, WinForms adalah alat yang kuat dan fleksibel untuk membangun aplikasi desktop di lingkungan Windows.

## BAB 3

### ANALISIS PEMECAHAN MASALAH

#### 3.1 Langkah - Langkah Pemecahan Masalah

##### 3.1.1 Fitur String Matching dengan Algoritma KMP

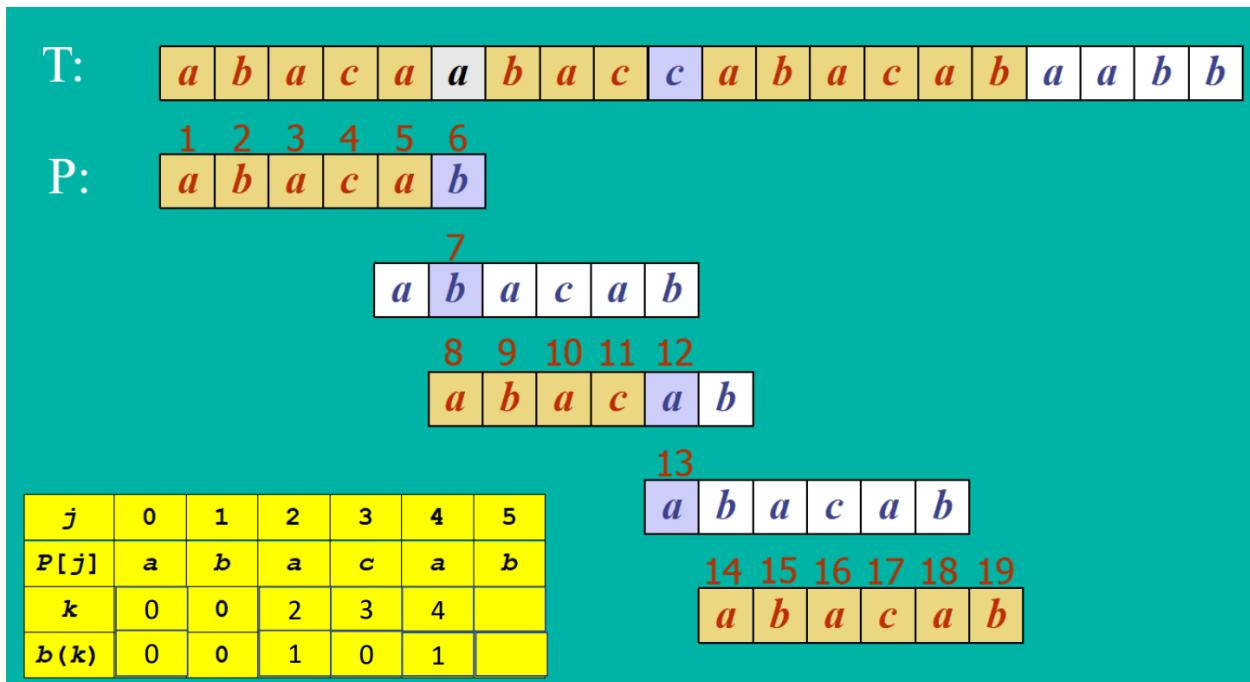
Berikut adalah langkah-langkah dalam mengimplementasi string matching dengan algoritma KMP.

- **Pendefinisian String:** Pertama, kita perlu mendefinisikan string pencarian dan string target. String pencarian adalah string yang ingin kita cari dalam string target.
- **Pembuatan Tabel Lompatan (Failure Function):** Sebelum memulai proses pencocokan, algoritma KMP memerlukan tabel lompatan. Tabel ini digunakan untuk menentukan berapa banyak karakter yang dapat dilewati saat mencocokkan string jika terjadi kegagalan pencocokan. Tabel ini dibuat dengan membandingkan prefix dan suffix dari string pencarian.
- **Pengisian Tabel Lompatan:** Tabel lompatan diisi dengan membandingkan prefix dan suffix dari string pencarian. Jika prefix dan suffix sama, nilai dalam tabel pada posisi tersebut akan ditingkatkan. Berikut adalah ilustrasi tabel lompatan.

$j$	0	1	2	3	4	5
$P[j]$	a	b	a	c	a	b
$k$	0	0	2	3	4	
$b(k)$	0	0	1	0	1	

Gambar 2. Ilustrasi Tabel Lompatan  
(Sumber: [PowerPoint Presentation \(itb.ac.id\)](#) )

- **Pencocokan String dengan Algoritma KMP:** Setelah tabel lompatan dibuat, kita dapat menggunakan algoritma KMP untuk mencari kecocokan antara string pencarian dan string target.
- **Proses Pencocokan:** Proses pencocokan dilakukan dengan membandingkan setiap karakter dari string target dengan string pencarian. Jika karakter cocok, kita lanjutkan ke karakter berikutnya. Jika tidak, kita melihat tabel lompatan dan melompat ke indeks yang ditentukan oleh tabel, bukan ke indeks berikutnya.
- **Penanganan Kegagalan Pencocokan:** Jika terjadi kegagalan pencocokan, algoritma akan melompat ke indeks yang ditentukan oleh tabel lompatan, bukan ke indeks berikutnya. Ini memungkinkan algoritma untuk tidak memeriksa karakter yang sudah diketahui tidak cocok.



Gambar 3. Ilustrasi Pencarian KMP

(Sumber: [PowerPoint Presentation \(itb.ac.id\)](http://PowerPoint Presentation (itb.ac.id)) )

- **Penanganan Hasil:** Setelah proses pencocokan selesai, kita perlu menangani hasilnya. Jika kecocokan ditemukan, algoritma akan mengembalikan indeks di mana kecocokan ditemukan. Jika tidak ada kecocokan, algoritma akan mengembalikan nilai yang menunjukkan bahwa tidak ada kecocokan.

### 3.1.2 Fitur String Matching dengan Algoritma Boyer-Moore

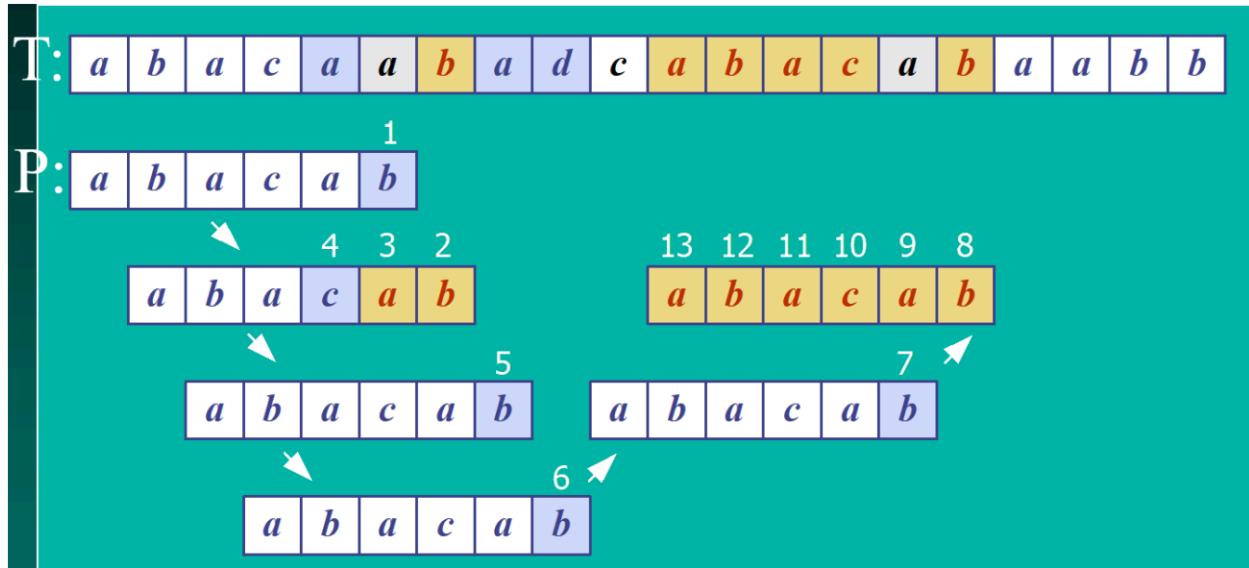
Berikut adalah langkah-langkah dalam mengimplementasi string matching dengan algoritma Boyer-Moore.

1. **Pendefinisian String:** Pertama, kita perlu mendefinisikan string pencarian dan string target. String pencarian adalah string yang ingin kita cari dalam string target.
2. **Pembuatan Tabel Bad Character:** Algoritma Boyer-Moore memerlukan tabel Bad Character. Tabel ini digunakan untuk menentukan berapa banyak karakter yang dapat dilewati saat mencocokkan string jika terjadi kegagalan pencocokan. Tabel ini dibuat dengan memetakan setiap karakter dalam string pencarian ke posisi terakhir kemunculannya dalam string tersebut.

<i>x</i>	<i>a</i>	<i>b</i>	<i>c</i>	<i>d</i>
<i>L(x)</i>	4	5	3	-1

Gambar 4. Ilustrasi Tabel Bad Character  
(Sumber:[PowerPoint Presentation \(itb.ac.id\)](#) )

3. **Pembuatan Tabel Good Suffix:** Selain tabel Bad Character, algoritma Boyer-Moore juga menggunakan tabel Good Suffix. Tabel ini digunakan untuk menentukan berapa banyak karakter yang dapat dilewati saat mencocokkan string jika terjadi kegagalan pencocokan dan ada suffix yang cocok antara string pencarian dan bagian dari string target yang telah diperiksa.
4. **Pencocokan String dengan Algoritma Boyer-Moore:** Setelah tabel Bad Character dan Good Suffix dibuat, kita dapat menggunakan algoritma Boyer-Moore untuk mencari kecocokan antara string pencarian dan string target.
5. **Proses Pencocokan:** Proses pencocokan dilakukan dengan membandingkan setiap karakter dari string target dengan string pencarian dari kanan ke kiri. Jika karakter cocok, kita lanjutkan ke karakter berikutnya. Jika tidak, kita melihat tabel Bad Character dan Good Suffix dan melompat ke indeks yang ditentukan oleh tabel, bukan ke indeks berikutnya.



Gambar 5. Ilustrasi Pencarian BM

(Sumber:[PowerPoint Presentation \(itb.ac.id\)](#) )

6. **Penanganan Kegagalan Pencocokan:** Jika terjadi kegagalan pencocokan, algoritma akan melompat ke indeks yang ditentukan oleh tabel Bad Character atau Good Suffix, bukan ke indeks berikutnya. Ini memungkinkan algoritma untuk tidak memeriksa karakter yang sudah diketahui tidak cocok.
7. **Penanganan Hasil:** Setelah proses pencocokan selesai, kita perlu menangani hasilnya. Jika kecocokan ditemukan, algoritma akan mengembalikan indeks di mana kecocokan ditemukan. Jika tidak ada kecocokan, algoritma akan mengembalikan nilai yang menunjukkan bahwa tidak ada kecocokan.

### 3.1.3 Sistem Deteksi Melalui Citra Sidik Jari

Berikut adalah langkah-langkah pemecahan masalah sistem deteksi melalui citra sidik jari.

1. **Pendefinisian citra yang akan dianalisis:** Pertama, kita perlu mendefinisikan citra sidik jari yang akan dicocokkan dan citra sidik jari target. Citra sidik jari yang akan dicocokkan adalah sidik jari yang didapat dari input user.
2. **Pengambilan Area Pencocokkan :** Alih-alih memproses seluruh citra sidik jari, yang bisa bervariasi dalam ukuran dan posisi jari, sistem mengekstrak area 8x8 piksel. Area ini

diambil dari bagian tengah citra sidik jari yang berukuran m x n piksel. Pemilihan area tengah ini didasarkan pada beberapa pertimbangan kritis.

- Pertama, **bagian tengah cenderung memiliki konsentrasi detail sidik jari yang tinggi** - ridge, loops, dan whorls yang membentuk pola unik tiap individu.
- Kedua, dan sama pentingnya, **fokus pada bagian tengah mengurangi risiko termasuknya area kosong** atau warna putih yang sering muncul di tepi citra sidik jari. Area kosong ini, yang mungkin dihasilkan dari bagian jari yang tidak menyentuh pemindai atau efek pemrosesan citra, tidak membawa informasi biometrik. Jika area ini dimasukkan dalam analisis, bisa menyebabkan kecocokan palsu yang tinggi antara sidik jari yang berbeda, karena algoritma akan menganggap area putih yang mirip sebagai kecocokan.
- Ketiga, **pemilihan ukuran 8x8 piksel menyediakan keseimbangan optimal** antara kekayaan detail dan efisiensi komputasi. Area yang lebih kecil, katakanlah 10x10 piksel, mungkin tidak menangkap variasi ridge dan loops yang cukup untuk membedakan sidik jari. Di sisi lain, area yang jauh lebih besar, seperti 100x100 piksel, akan meningkatkan kompleksitas komputasi secara signifikan - memperlambat proses pencocokan tanpa memberikan peningkatan akurasi yang sebanding. Ukuran 8x8 menyediakan cukup detail untuk identifikasi unik namun cukup kecil untuk pemrosesan cepat.

Dengan membatasi analisis pada area tengah 8x8 piksel, sistem tidak hanya meningkatkan efisiensi komputasi tetapi juga secara signifikan mengurangi variabilitas yang disebabkan oleh perbedaan dalam pengambilan citra, seperti posisi atau rotasi jari. Lebih penting lagi, pendekatan ini secara cerdas menghindari kecocokan palsu yang disebabkan oleh area non-informatif, meningkatkan keandalan sistem secara keseluruhan.

3. **Pengubahan Citra Sidik Jari menjadi String Base64:** Setelah memperoleh bagian tengah citra sidik jari, langkah berikutnya adalah mengubahnya ke format yang cocok untuk algoritma pencarian string. Ini dilakukan dalam dua tahap. Pertama, citra diubah menjadi data biner, memastikan setiap detail sidik jari tetap utuh. Kedua, data biner ini dikonversi menjadi string Base64. Penggunaan Base64 adalah pilihan strategis. Format ini menggunakan hanya 64 karakter ASCII yang aman, membuatnya ideal untuk penyimpanan dalam basis data teks dan transmisi melalui berbagai sistem. Yang

terpenting, Base64 memungkinkan rekonstruksi data citra asli tanpa kehilangan informasi, kritis untuk integritas sidik jari. Dengan pendekatan ini, setiap pola unik ridge dan loop sidik jari direpresentasikan sebagai string unik, siap untuk diproses oleh algoritma KMP atau BM.

4. **Pencarian dalam Basis Data:** Setelah mendapatkan String Base64 untuk dicocokkan, langkah berikutnya adalah mencari kecocokan antara string tersebut dengan setiap string target dalam basis data menggunakan algoritma Knuth-Morris-Pratt atau Boyer-Moore.
5. **Penanganan Hasil:** Setelah proses pencocokan selesai, kita perlu menangani hasilnya. Jika algoritma KMP maupun BM dapat menemukan citra sidik jari dalam basis data yang *exact match* dengan citra sidik jari target, maka akan dikembalikan tingkat kemiripan sebesar 100%. Namun jika tidak ditemukan string *match*, maka akan digunakan algoritma Levenshtein Distance untuk mencari tingkat kemiripan yang paling mendekati antara string sidik jari yang akan dicocokkan dengan string-string sidik jari target pada basis data. Untuk menjaga integritas dan keandalan sistem, telah ditetapkan ambang batas kemiripan minimum sebesar 60% ketika menggunakan Jarak Levenshtein. Angka 60% ini dipilih setelah pengujian berulang. Kami menemukan bahwa di bawah 60%, perbedaan sidik jari terlalu besar—mungkin jari yang berbeda atau sidik jari yang sangat terdistorsi. Di atas 60%, perbedaannya cenderung kecil, seperti yang disebabkan oleh variasi dalam pemindaian. Jadi, 60% menjadi titik keseimbangan: cukup tinggi untuk menghindari kecocokan yang salah, namun cukup rendah untuk menangani variasi normal dalam pemindaian sidik jari.

## 3.2 Fitur Fungsional dan Arsitektur Aplikasi Desktop Yang Dibangun

### 3.2.1 Fitur Fungsional

#### 1. Pemilihan Citra Sidik Jari

Pengguna dapat memilih citra sidik jari dari perangkat pengguna. Fitur ini diimplementasikan dengan komponen OpenFileDialog yang memungkinkan pengguna untuk memuat file gambar dengan format .jpg, .jpeg, .png, atau .bmp. Gambar yang dipilih kemudian ditampilkan di PictureBox yang telah disediakan.

## **2. Pilihan Algoritma Pencarian**

Aplikasi ini menyediakan dua algoritma pencarian pola untuk proses deteksi: Boyer-Moore (BM) dan Knuth-Morris-Pratt (KMP). Pengguna dapat memilih salah satu dari algoritma ini melalui komponen RadioButton.

## **3. Pencarian dan Kecocokan Sidik Jari**

Setelah citra dipilih dan algoritma ditentukan, aplikasi melakukan proses pencarian untuk menemukan kecocokan antara citra sidik jari yang diunggah dengan data sidik jari yang sudah ada dalam database. Hasil dari proses ini mencakup persentase kecocokan dan waktu pencarian yang ditampilkan di label lblPersentaseKecocokan dan lblWaktuPencarian.

## **4. Tampilan Hasil Pencarian**

Jika ditemukan kecocokan, aplikasi menampilkan citra sidik jari yang cocok dari database di picBoxMatched dan citra hasil pencocokan di picBoxResult.

### **3.2.2 Arsitektur Aplikasi**

Arsitektur aplikasi ini didesain dengan pendekatan berorientasi objek dan memanfaatkan berbagai komponen dari framework .NET untuk pengembangan aplikasi Windows Forms. Berikut adalah penjelasan detail dari arsitektur yang digunakan:

#### **1. Struktur Umum**

##### **a. Form1**

Kelas utama yang mengatur antarmuka pengguna dan logika aplikasi. Form1 berisi elemen-elemen UI seperti label, tombol, dan picture box serta event handler yang mengatur respon terhadap tindakan pengguna.

##### **b. Algorithms**

Namespace yang berisi implementasi algoritma Boyer-Moore dan KMP yang digunakan untuk pencarian pola pada data sidik jari.

##### **c. Data**

Namespace yang berisi utility untuk mengakses dan memproses data dari database, termasuk konversi gambar ke binary dan sebaliknya.

## 2. Komponen Utama dan Fungsi

### a. Form1\_Load

Event handler yang memuat data dari database saat form pertama kali dimuat. Memanfaatkan metode LoadDataFromDB untuk mendapatkan data.

### b. btnPilihCitra\_Click

Event handler yang mengatur proses pemilihan gambar sidik jari. Menggunakan OpenFileDialog untuk memungkinkan pengguna memilih file gambar dari komputer mereka dan kemudian mengkonversi gambar tersebut ke format binary untuk diproses lebih lanjut.

### c. button2\_Click

Event handler yang mengatur proses pencarian citra sidik jari yang dipilih dengan data yang ada di database. Memanggil metode SearchFingerprint untuk melakukan pencarian dengan algoritma yang dipilih.

### d. SearchFingerprint

Metode asinkron yang melakukan pencarian citra sidik jari dengan menggunakan algoritma yang dipilih (Boyer-Moore atau KMP). Metode ini menghitung kesamaan antara citra yang diunggah dengan data dalam database dan menampilkan hasilnya di UI.

## 3. Data dan Algoritma

### a. BoyerMoore dan KMP

Kelas yang mengimplementasikan algoritma Boyer-Moore dan Knuth-Morris-Pratt untuk pencarian pola. Kedua kelas ini digunakan untuk melakukan pencocokan antara sidik jari yang diunggah dengan data yang ada.

### b. DBUtilities

Kelas utility yang berisi metode untuk memproses dan mengakses data dari database. Termasuk metode GetDataFromDB untuk mengambil data dan ConvertImageToBinary untuk mengkonversi gambar ke format binary.

## 3.3 Ilustrasi Kasus

Seorang pengguna ingin mengidentifikasi individu berdasarkan citra sidik jari yang diunggah. Aplikasi akan mencocokkan citra sidik jari yang diunggah dengan data sidik jari yang ada di database menggunakan algoritma Boyer-Moore atau KMP. Berikut adalah langkah-langkah atau step by stepnya.

### Langkah 1: Memilih Citra Sidik Jari

- **Tujuan:** Memungkinkan pengguna untuk memilih citra sidik jari dari perangkat mereka.
- **Proses:**
  1. Pengguna membuka dialog untuk memilih file gambar sidik jari dari komputer mereka.
  2. Sistem menampilkan gambar yang dipilih di antarmuka aplikasi.
  3. Sistem mengkonversi gambar yang dipilih ke format binary dan base64 untuk diproses lebih lanjut.

### Langkah 2: Memilih Algoritma Pencarian

- **Tujuan:** Memungkinkan pengguna untuk memilih algoritma pencarian pola yang akan digunakan (Boyer-Moore atau KMP).
- **Proses:**
  1. Pengguna memilih algoritma yang diinginkan melalui antarmuka aplikasi.

### Langkah 3: Melakukan Pencarian Sidik Jari

- **Tujuan:** Mencocokkan citra sidik jari yang diunggah dengan data sidik jari dalam database menggunakan algoritma yang dipilih.

- **Proses:**
  1. Pengguna memulai proses pencarian dengan menekan tombol pencarian.
  2. Sistem memvalidasi apakah gambar dan algoritma telah dipilih.
  3. Sistem memulai proses pencarian dengan algoritma yang dipilih (Boyer-Moore atau KMP).
- 4. **Detail Proses Pencarian:**
  - **Boyer-Moore:**
    - Menghasilkan tabel bad character untuk mempercepat pencarian.
    - Memindai teks dari kanan ke kiri dan menggunakan tabel bad character untuk menggeser pola sesuai kebutuhan.
  - **KMP:**
    - Menghasilkan tabel longest prefix suffix (LPS) untuk mencocokkan pola dengan teks.
    - Memindai teks dari kiri ke kanan dan menggunakan tabel LPS untuk menghindari pencocokan ulang.
- 5. Sistem menghitung jarak Levenshtein untuk mengukur kemiripan antara pola dan teks jika pola tidak ditemukan.

#### **Langkah 4: Menampilkan Hasil Pencarian**

- **Tujuan:** Menampilkan hasil pencarian kepada pengguna.
- **Proses:**
  1. Sistem menghitung persentase kecocokan berdasarkan hasil pencarian dan jarak Levenshtein.
  2. Sistem menampilkan persentase kecocokan dan waktu yang dibutuhkan untuk pencarian di antarmuka aplikasi.
  3. Jika ditemukan kecocokan, sistem menampilkan citra sidik jari yang cocok dari database di antarmuka aplikasi.
  4. Jika tidak ditemukan kecocokan, sistem menampilkan pesan bahwa tidak ada kecocokan yang ditemukan.

## BAB 4

### IMPLEMENTASI DAN PENGUJIAN

#### 4.1 Spesifikasi Teknis Program

##### 4.1.1 KnuthMorrisPratt.cs

Kelas KMP digunakan untuk mengimplementasikan algoritma Knuth-Morris-Pratt (KMP) untuk pencarian pola dalam teks, serta menghitung jarak Levenshtein antara dua string. List<int> result digunakan untuk menyimpan indeks kemunculan pola dalam teks. int[] lps digunakan untuk menyimpan nilai Longest Prefix Suffix (LPS) dalam algoritma KMP. int[,] distance digunakan untuk menyimpan matriks jarak Levenshtein dalam perhitungan jarak Levenshtein. KMPSearch(string pat, string txt) digunakan untuk mencari semua kemunculan pola pat dalam teks txt menggunakan algoritma KMP, dan mengembalikan daftar indeks kemunculan. ComputeLPSArray(string pat, int M, int[] lps) digunakan untuk menghitung nilai Longest Prefix Suffix (LPS) untuk pola yang diberikan dalam algoritma KMP. CalculateLevenshteinDistance(string text, string pattern) digunakan untuk menghitung jarak Levenshtein (jumlah operasi pengeditan minimum) antara dua string text dan pattern.

##### 4.1.2 BoyerMoore.cs

Kelas BoyerMoore digunakan untuk mengimplementasikan algoritma pencarian string Boyer-Moore dan menghitung jarak Levenshtein antara dua string. List<int> occurrences digunakan untuk menyimpan indeks awal dari kemunculan pola dalam teks. int[] badMatchTable digunakan untuk menyimpan tabel heuristik karakter buruk dalam algoritma Boyer-Moore. int[,] distance digunakan untuk menyimpan matriks jarak Levenshtein dalam perhitungan jarak Levenshtein. GenerateBadMatchTable(string pattern) digunakan untuk menghasilkan tabel heuristik karakter buruk untuk pola yang diberikan dalam algoritma Boyer-Moore.

`Search(string text, string pattern)` digunakan untuk mencari semua kemunculan pola pattern dalam teks text menggunakan algoritma Boyer-Moore, dan mengembalikan daftar indeks awal kemunculan. `CalculateLevenshteinDistance(string text, string pattern)` digunakan untuk menghitung jarak Levenshtein (jumlah operasi pengeditan minimum) antara dua string text dan pattern.

#### 4.1.3 Database.cs

Kelas DBUtilities berisi kumpulan metode statis yang digunakan untuk berinteraksi dengan database, seperti inisialisasi database, mengonversi gambar menjadi biner, mengambil data dari database, dan mengupdate data dalam database. `List<string>` digunakan untuk menyimpan daftar data seperti daftar berkas citra dan daftar nama dari database. `InitializeDBAsync(IConfiguration configuration)` digunakan untuk menginisialisasi database dengan berkas citra BMP dari direktori uji.

`ConvertImageToBinary(Bitmap image)` digunakan untuk mengonversi objek `Bitmap` menjadi representasi biner. `GetDataFromDB(IConfiguration configuration)` digunakan untuk mengambil daftar berkas citra dari tabel sidik\_jari dalam database. `GetNamesByCitraFromDB(IConfiguration configuration, string citra)` digunakan untuk mengambil nama-nama yang terkait dengan berkas citra tertentu dari tabel sidik\_jari dalam database. `GetNamesFromBiodata(IConfiguration configuration)` digunakan untuk mengambil daftar nama dari tabel biodata dalam database.

`GetBiodataByNameFromDB(IConfiguration configuration, string name)` digunakan untuk mengambil data biodata seseorang berdasarkan namanya dari tabel biodata dalam database. `UpdateNameInSidikJariAsync(IConfiguration configuration, string oldName, string newName)` digunakan untuk mengupdate nama dalam tabel sidik\_jari dalam database. Kelas ini menggunakan Dapper, library pemetaan objek-relasional (ORM) untuk .NET, dan Microsoft.Data.Sqlite untuk berinteraksi dengan database SQLite. Kelas ini juga menggunakan kelas `SqliteConnection` dan `IConfiguration` dari .NET untuk konfigurasi dan koneksi ke database.

#### **4.1.4 AlayConverter.cs**

Kelas AlayConverter digunakan untuk mengonversi teks dalam format alay (gaya bahasa gaul dengan mengganti huruf tertentu dengan angka) menjadi teks biasa. private static readonly Dictionary<char, char> digitToLetterMap merupakan sebuah dictionary yang memetakan angka ke huruf dalam format alay. private static string ConvertDigitsToLetters(string input) adalah fungsi untuk mengonversi angka dalam sebuah string menjadi huruf sesuai pemetaan dalam digitToLetterMap. private static string NormalizeCase(string input) adalah fungsi untuk menyeragamkan penggunaan huruf kapital dan non-kapital dalam sebuah string. public static string ConvertAlayToOriginal(string alayText) adalah fungsi utama yang dipanggil untuk mengonversi teks dalam format alay menjadi teks biasa. Memanggil fungsi NormalizeCase dan ConvertDigitsToLetters secara berurutan.

#### **4.1.5 Form1.cs**

Form1 adalah kelas utama yang merepresentasikan antarmuka pengguna (UI) dan logika aplikasi. BoyerMoore dan KMP adalah kelas-kelas yang mengimplementasikan algoritma pencarian string Boyer-Moore dan Knuth-Morris-Pratt. DBUtilities adalah kelas utilitas untuk berinteraksi dengan database SQLite menggunakan Dapper ORM. AlayConverter adalah kelas untuk mengonversi nama "alay" menjadi nama asli. Biodata adalah kelas model untuk menyimpan informasi biodata seseorang. List<string> data digunakan untuk menyimpan representasi Base64 dari bagian tengah citra sidik jari. List<string> files digunakan untuk menyimpan nama-nama file citra sidik jari. List<string> names\_alay digunakan untuk menyimpan nama-nama dalam format "alay" dari tabel biodata. List<string> names\_ori digunakan untuk menyimpan nama-nama asli yang dikonversi dari nama "alay".

LoadCustomFont() digunakan untuk memuat font kustom dari file dan menerapkannya pada elemen-elemen UI. LoadDataFromDB() digunakan untuk memuat data sidik jari dan nama dari database, mengonversi gambar menjadi data biner. btnPilihCitra\_Click() digunakan untuk memungkinkan pengguna memilih gambar sidik

jari dan menyimpannya sebagai data biner. button2\_Click() dan SearchFingerprint() digunakan untuk mencari sidik jari yang cocok, menghitung persentase kecocokan, dan menampilkan hasil. InitializeDBAsync(), ConvertImageToBinary(), GetDataFromDB(), GetNamesByCitraFromDB(), GetNamesFromBiodata(), GetBiodataByNameFromDB(), dan UpdateNameInSidikJariAsync() digunakan untuk Fungsi-fungsi dalam DBUtilities untuk berinteraksi dengan database.

## 4.2 Tata Cara Penggunaan Program

Untuk menjalankan program yang telah dibuat, perlu dilakukan langkah - langkah sebagai berikut.

1. Clone Repository GitHub pada Visual Studio
2. Pilih file .sln
3. Jalankan dengan klik tombol run/play atau tekan F5

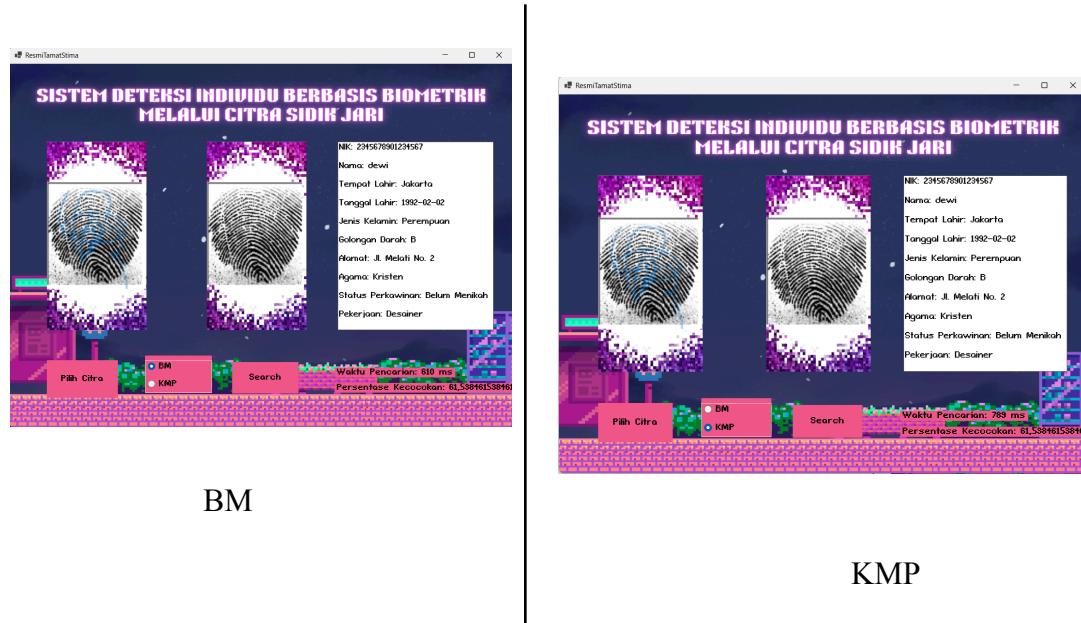
## 4.3 Hasil Pengujian

### 4.3.1 Kasus 1



Pada kasus testing yang pertama sidik jari yang dicoba adalah sidik jari yang memang terdaftar dan sama persis dengan inputan. Maka didapatkan hasilnya kedua algoritma mendapatkan tingkat kecocokannya 100%. Algoritma BM mengeksekusi selama 537 ms dan KMP selama 806ms .

#### 4.3.2 Kasus 2

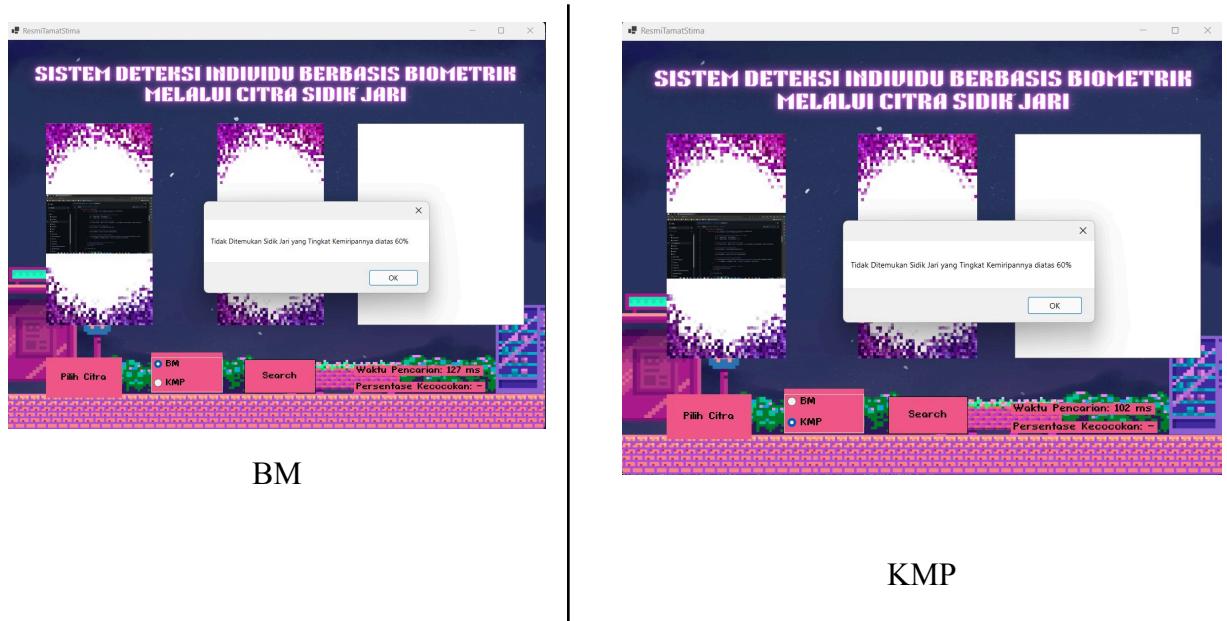


BM

KMP

Pada kasus testing yang kedua ,sidik jari yang dicoba adalah sidik jari yang kita sedikit coret-coret dari sidik jari aslinya untuk mengecek apakah algoritmanya dapat menemukan hasilnya. Maka didapatkan hasilnya kedua algoritma mendapatkan tingkat kecocokannya 61,5%. Algoritma BM mengeksekusi selama 610ms dan KMP selama 789ms.

### 4.3.3 Kasus 3



Pada kasus testing yang ketiga , dicoba menggunakan suatu gambar hasil screenshot random untuk mengetes hasilnya ketika dibandingkan dengan sidik jari dari database. Maka didapatkan hasilnya kedua algoritma mendapatkan tingkat kecocokannya dibawah 60% sehingga hasilnya tidak akan di tampilkan karena batas yang kami tentukan jika cocok adalah diatas 60%. Algoritma BM mengeksekusi selama 127ms dan KMP selama 102ms.

### 4.3.4 Kasus 4



Pada kasus testing yang keempat, dilakukan pencarian menggunakan sidik jari random, maka didapatkan sidik jari yang cocok , akan tetapi pada pencarian biodata di database berdasarkan nama di sidik jari tidak terdapat persentase tingkat kecocokan diatas 60% sehingga hasilnya tidak dapat dianggap sebagai biodatanya jika dibawah 60%. Algoritma BM mengeksekusi selama 509ms dan KMP selama 678ms.

## 4.4 Analisis Hasil

Berdasarkan hasil pengujian yang telah dilakukan, berikut adalah analisis keakuratan dan optimalitas dari algoritma Boyer-Moore (BM) dan Knuth-Morris-Pratt (KMP):

### 4.4.1 Analisis Keakuratan

#### 1. Kasus 1: Sidik Jari Identik

- **BM:** 100% akurasi, waktu eksekusi 537 ms.
- **KMP:** 100% akurasi, waktu eksekusi 806 ms.
- Pada kasus ini, kedua algoritma mencapai akurasi sempurna ketika sidik jari yang diuji identik dengan yang ada di database. Hal ini menunjukkan

bahwa kedua algoritma mampu mengidentifikasi sidik jari yang identik dengan sangat baik.

## 2. Kasus 2: Sidik Jari Sedikit Berubah

- **BM:** 61.5% akurasi, waktu eksekusi 610 ms.
- **KMP:** 61.5% akurasi, waktu eksekusi 789 ms.
- Akurasi kedua algoritma turun secara signifikan menjadi 61.5% ketika sidik jari mengalami sedikit perubahan. Ini menunjukkan bahwa meskipun terjadi perubahan kecil pada sidik jari, kedua algoritma masih bisa menemukan kecocokan tetapi dengan tingkat kepercayaan yang lebih rendah.

## 3. Kasus 3: Tangkapan Layar Acak

- **BM:** Akurasi kurang dari 60%, waktu eksekusi 127 ms.
- **KMP:** Akurasi kurang dari 60%, waktu eksekusi 102 ms.
- Pada kasus ini, kedua algoritma tidak menemukan kecocokan yang relevan untuk gambar yang sepenuhnya tidak terkait, seperti yang diharapkan. Waktu eksekusi juga jauh lebih rendah dibandingkan kasus lainnya karena tidak adanya pola yang cocok.

## 4. Kasus 4: Sidik Jari Acak

- **BM:** Waktu eksekusi 509 ms.
- **KMP:** Waktu eksekusi 678 ms.
- Hasil menunjukkan bahwa meskipun ditemukan kecocokan sidik jari, tidak ada biodata yang tingkat kemiripannya di atas 60% ditemukan di database. Waktu eksekusi berada di antara kasus 1 dan 2 serta lebih rendah dibandingkan kasus 3.

### 4.4.2 Analisis Optimalitas

#### ● Waktu Eksekusi:

- BM secara konsisten menunjukkan waktu eksekusi yang lebih cepat dibandingkan KMP di semua kasus uji.
- Kasus 1: BM (537 ms) lebih cepat daripada KMP (806 ms).
- Kasus 2: BM (610 ms) lebih cepat daripada KMP (789 ms).

- Kasus 3: BM (127 ms) lebih cepat daripada KMP (102 ms).
- Kasus 4: BM (509 ms) lebih cepat daripada KMP (678 ms).

- **Konsistensi Performa:**

- BM menunjukkan performa yang lebih optimal dalam hal kecepatan eksekusi di semua skenario yang diuji.
- Perbedaan waktu eksekusi sangat mencolok terutama pada kasus di mana ditemukan kecocokan yang dekat (Kasus 1 dan 2).

#### **4.4.3 Hasil Analisis**

- **Keakuratan:** Kedua algoritma, BM dan KMP, menunjukkan tingkat akurasi yang sangat baik untuk sidik jari yang identik dan menurun secara signifikan untuk sidik jari yang sedikit berubah. Keduanya tidak menemukan kecocokan yang relevan untuk gambar yang sepenuhnya tidak terkait, yang merupakan perilaku yang diharapkan.
- **Optimalitas:** BM lebih optimal dalam hal kecepatan eksekusi dibandingkan KMP. Ini menjadikan BM pilihan yang lebih baik untuk skenario di mana kecepatan adalah faktor yang kritis.

Dari analisis ini, dapat disimpulkan bahwa meskipun kedua algoritma efektif untuk pencocokan sidik jari, algoritma Boyer-Moore menawarkan kinerja yang lebih baik dalam hal waktu eksekusi tanpa mengorbankan akurasi.

## **BAB 5**

### **KESIMPULAN DAN SARAN**

#### **5.1 Kesimpulan**

Dari tugas besar ini, kami telah berhasil mengembangkan sebuah program yang dapat digunakan untuk mendeteksi individu berdasarkan citra sidik jari dengan menggunakan algoritma pattern matching Boyer-Moore (BM) dan Knuth-Morris-Pratt (KMP). Program ini mengimplementasikan kedua algoritma tersebut untuk melakukan pencocokan pola antara citra sidik jari yang diinputkan dengan basis data citra sidik jari yang tersedia.

Algoritma BM dan KMP terbukti efektif dalam menyelesaikan permasalahan pencocokan pola citra sidik jari dengan cepat dan efisien. Pemilihan antara kedua algoritma ini dapat bergantung pada preferensi pengguna atau karakteristik data yang digunakan. Jika pengguna ingin mendapatkan hasil pencocokan yang lebih cepat, maka algoritma BM lebih disarankan karena kemampuannya dalam melompati karakter yang tidak cocok secara lebih efisien. Namun, jika pengguna ingin mendapatkan hasil yang lebih akurat dan dapat menangani pola yang kompleks, algoritma KMP lebih sesuai karena kemampuannya dalam menangani prefiks dan sufiks yang berulang.

Dengan menggunakan algoritma BM dan KMP dalam program ini, pendekripsi individu berbasis biometrik melalui citra sidik jari dapat dilakukan dengan lebih efisien dan akurat. Hal ini berkontribusi dalam pengembangan sistem keamanan dan identifikasi yang lebih andal dan terpercaya.

#### **5.2 Saran**

Saran pengembangan untuk tugas besar ini adalah :

1. Meskipun algoritma BM dan KMP sudah terbukti efisien, masih terdapat ruang untuk peningkatan akurasi pendekripsi. Penelitian lebih lanjut dapat dilakukan untuk mengeksplorasi teknik-teknik pra-pemrosesan citra dan peningkatan algoritma pencocokan pola, sehingga dapat menangani variasi yang lebih besar dalam citra sidik jari.

2. Program saat ini mungkin kurang efektif dalam menangani citra sidik jari yang rusak atau tidak lengkap. Pengembangan lebih lanjut dapat dilakukan untuk meningkatkan kemampuan program dalam menangani citra yang tidak sempurna, seperti dengan menggunakan teknik rekonstruksi atau interpolasi citra.
3. Untuk meningkatkan pengalaman pengguna dan memudahkan penggunaan program, pengembangan antarmuka pengguna yang lebih intuitif dan ramah pengguna dapat dilakukan. Hal ini akan mempermudah interaksi pengguna dengan program dan memungkinkan penggunaan yang lebih luas.

### 5.3 Refleksi

Dalam pengembangan program deteksi individu berbasis biometrik melalui citra sidik jari, kami mengimplementasikan dua algoritma pencocokan pola, yaitu Boyer-Moore (BM) dan Knuth-Morris-Pratt (KMP). Selama pengembangan, kami menghadapi beberapa tantangan, terutama dalam mengolah dan memproses citra sidik jari secara efisien dan akurat.

Salah satu tantangan utama yang kami hadapi adalah menangani variasi dalam citra sidik jari, seperti resolusi, kontras, dan kualitas citra yang berbeda-beda. Hal ini mempengaruhi akurasi pencocokan pola dan membutuhkan pra-pemrosesan citra yang optimal. Kami menyadari bahwa pra-pemrosesan citra merupakan aspek penting yang harus diperhatikan untuk meningkatkan kinerja program.

Selain itu, kami juga menyadari bahwa implementasi algoritma BM dan KMP yang kami lakukan masih memiliki ruang untuk peningkatan lebih lanjut. Misalnya, dengan mengoptimalkan penggunaan memori atau mengintegrasikan teknik-teknik lain seperti heuristik untuk meningkatkan kecepatan pencarian atau menggunakan optimalisasi lainnya yang tidak kami terapkan.

Namun, secara keseluruhan, pengalaman dalam mengembangkan program ini memberikan wawasan yang berharga tentang penerapan algoritma pencocokan pola dalam pemecahan masalah nyata di bidang biometrik dan pengolahan citra. Kami menyadari pentingnya memahami konsep-konsep dasar algoritma serta mengimplementasikannya secara efisien untuk mendapatkan solusi yang optimal. Ke depannya, kami berharap dapat meningkatkan kinerja dan akurasi program ini dengan mengeksplorasi teknik-teknik baru dalam

pengolahan citra dan pencocokan pola. Selain itu, kami juga berencana untuk mengintegrasikan program ini dengan sistem keamanan yang lebih luas, sehingga dapat memberikan kontribusi yang lebih besar dalam menjaga keamanan dan identifikasi individu.

## **LAMPIRAN**

Tautan repository GitHub: [https://github.com/akmalrmn/Tubes3\\_ResmiTamatStima](https://github.com/akmalrmn/Tubes3_ResmiTamatStima)

Tautan video: <https://youtu.be/LVosUFuAihQ>

## **DAFTAR PUSTAKA**

Munir, Rinaldi. 2024. Pencocokan string (String matching/pattern matching). [PowerPoint Presentation \(itb.ac.id\)](#). Diakses pada 18 May 2024.

Munir, Rinaldi. 2024. Pencocokan string dengan Regular Expression (Regex). [String Matching dengan Regular Expression \(itb.ac.id\)](#). Diakses pada 18 May 2024.