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

OLEH:

Mohammad Nugraha Eka Prawira	13522001
Bastian H. Suryapratama	13522034
Muhammad Syarafi Akmal	13522076

PROGRAM STUDI TEKNIK INFORMATIKA

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA

INSTITUT TEKNOLOGI BANDUNG

2024

BAB I
DESKRIPSI TUGAS

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# dengan kakas Visual Studio .NET yang mengimplementasikan algoritma KMP, BM, dan Regular Expression dalam mencocokkan sidik jari dengan biodata yang berpotensi rusak. Pelajarilah C# desktop development, disarankan untuk menggunakan Visual Studio untuk mempermudah pengerjaan.
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 II
LANDASAN TEORI

A. Deskripsi singkat algoritma KMP, BM, dan Regex

Algoritma Knuth-Morris-Pratt (KMP) adalah algoritma pencocokan string yang efisien, dirancang untuk menemukan pola dalam teks tanpa harus mencocokkan ulang karakter yang sudah dibandingkan sebelumnya. Algoritma ini terdiri dari dua fase utama:

1. Prapemrosesan

   Membuat tabel "lps" (longest proper prefix which is also suffix). Tabel ini menyimpan panjang dari proper prefix yang juga merupakan suffix untuk setiap posisi dalam pola. Tabel lps membantu menentukan langkah pergeseran yang diperlukan ketika terjadi ketidakcocokan karakter, sehingga menghindari perbandingan ulang yang tidak perlu.

2. Pencocokan

   Menggunakan tabel lps selama proses pencarian: Saat mencocokkan pola dengan teks, jika terjadi ketidakcocokan, KMP menggunakan informasi dari tabel lps untuk menentukan posisi berikutnya di mana pola dapat dilanjutkan, tanpa mengulang perbandingan dari awal. Hal ini memungkinkan pencarian yang lebih efisien dengan kompleksitas waktu O(n + m), di mana n adalah panjang teks dan m adalah panjang pola.

Efisiensi algoritma KMP berasal dari kemampuannya untuk “melompat” dalam pola berdasarkan informasi yang telah dihitung sebelumnya, sehingga mengurangi jumlah perbandingan yang diperlukan secara signifikan.

Algoritma Boyer-Moore (BM) adalah algoritma pencocokan string yang sangat efisien, terutama ketika pola yang dicari relatif panjang dibandingkan dengan teks. Algoritma ini bekerja dengan memanfaatkan informasi dari pola untuk mengurangi jumlah perbandingan karakter yang diperlukan. Algoritma BM menggunakan dua heuristik utama:

1. Bad Character Heuristic

   Saat terjadi ketidakcocokan, pola digeser ke kanan sehingga karakter teks yang tidak cocok disejajarkan dengan kemunculan terakhirnya dalam pola. Jika karakter tersebut tidak ada dalam pola, pola digeser melewati seluruh karakter tersebut.

2. Good Suffix Heuristic

   Jika terjadi ketidakcocokan setelah beberapa karakter cocok, pola digeser sehingga bagian yang sudah cocok sebelumnya tetap sejajar. Jika bagian yang cocok tersebut muncul lagi dalam pola, pola digeser untuk mencocokkan kemunculan tersebut. Jika tidak, pola digeser melewati seluruh bagian yang cocok.

Dengan menggabungkan kedua heuristik ini, algoritma BM sering kali dapat “melompat” lebih jauh dibandingkan dengan pencocokan satu per satu, sehingga mempercepat proses pencarian. Kompleksitas waktu terburuknya adalah O(nm), tetapi dalam praktiknya sering mendekati O(n/m) di mana n adalah panjang teks dan m adalah panjang pola.

Regex (Regular Expression) adalah suatu kode yang digunakan untuk mendeskripsikan pola pencarian dan manipulasi string. Regex memungkinkan pencarian yang sangat fleksibel dan efisien dalam teks menggunakan pola yang terdiri dari karakter dan simbol khusus. Berikut adalah beberapa fitur utama Regex:

1. Literal Character Matching

   Pencarian karakter secara langsung, misalnya, pola abc akan mencari urutan "abc" dalam teks.

2. Metakarakter

   Karakter khusus yang memiliki makna tertentu dalam pola, misalnya:

   . (titik) : Mewakili sembarang karakter kecuali baris baru.

   * (bintang) : Mewakili nol atau lebih dari karakter sebelumnya.

   ^ (caret) : Menandai awal baris.

   $ (dolar) : Menandai akhir baris.

3. Grup dan Alternasi

   Mengelompokkan beberapa karakter atau sub-pola menggunakan tanda kurung () dan menggunakan | untuk menyatakan alternatif. Misalnya, pola (abc|def) akan cocok dengan "abc" atau "def".

4. Kelas Karakter

   Menyatakan satu dari sekumpulan karakter, misalnya [a-z] untuk semua huruf kecil atau [0-9] untuk semua digit.

Regex digunakan secara luas dalam berbagai aplikasi seperti validasi input, pencarian teks, penggantian teks, dan pemrosesan data dalam pemrograman. Regex memungkinkan

penulisan pola pencarian yang kompleks dan mendetail dengan cara yang ringkas dan efisien.

B. Penjelasan teknik pengukuran persentase kemiripan

Teknik pengukuran persentase kemiripan yang digunakan adalah *Levenshtein Distance*. *Levenshtein Distance* adalah sebuah metrik untuk mengukur perbedaan antara dua string. Metrik ini menghitung jumlah operasi yang diperlukan untuk mengubah satu string menjadi string lainnya, dengan tiga jenis operasi yang diperbolehkan:

1. Penggantian (*Substitution*): Mengganti satu karakter di string pertama dengan karakter lain di string kedua.
2. Penyisipan (*Insertion*): Menyisipkan satu karakter ke dalam string pertama.
3. Penghapusan (*Deletion*): Menghapus satu karakter dari string pertama.

Levenshtein Distance sering digunakan dalam aplikasi yang membutuhkan perbandingan kesamaan teks, seperti pengoreksian ejaan, pengenalan suara, dan pencarian *fuzzy*. Nilai *Levenshtein Distance* antara dua string adalah jumlah minimum operasi yang diperlukan untuk mengubah satu string menjadi string lainnya. Algoritma ini biasanya diimplementasikan menggunakan pemrograman dinamis untuk mencapai kompleksitas waktu O(n * m), di mana n dan m adalah panjang dari dua string yang dibandingkan.

C. Penjelasan singkat mengenai aplikasi desktop yang dibangun

Aplikasi desktop yang kami bangun dibuat dengan bahasa pemrograman C#. Terdapat dua bagian utama pada tampilan aplikasi desktop, yaitu bagian *input* data dan *output* hasil pencarian. Pada bagian *input*, data yang perlu dimasukkan adalah gambar sidik jari dan algoritma pencocokan string. Untuk algoritma pencocokan string, algoritma yang dapat dipilih adalah Knuth-Morris-Pratt (KMP) dan Boyer-Moore (BM). Pada bagian *output*, ditampilkan beberapa informasi, yaitu sidik jari yang cocok, biodata pemilik, waktu pencarian, serta persentase kecocokan.

BAB III
ANALISIS PEMECAHAN MASALAH

A. Langkah-Langkah Pemecahan Masalah

Masalah yang ingin diselesaikan adalah mencari biodata seseorang berdasarkan data sidik jari yang diberikan. Secara garis besar, terdapat 4 tahapan dalam menyelesaikan masalah tersebut, yaitu memasukkan dan mengkonversi data sidik jari, mencocokan sidik jari dengan nama pemiliknya, mencari biodata yang cocok berdasarkan nama pemiliknya, serta menampilkan biodata tersebut.

Pertama-tama, kita perlu memasukkan gambar sidik jari yang ingin dicari biodata pemiliknya. Dari gambar sidik jari tersebut, diambil data sebanyak 30 pixel untuk dipakai dalam proses pencocokan sidik jari. Selanjutnya, data pixel tersebut dikonversi menjadi *binary*. Untuk mempercepat proses *pattern matching*, kode *binary* tersebut dikelompokkan per 8 bit sehingga membentuk karakter ASCII. Karakter ASCII inilah yang akan digunakan dalam proses pencocokan.

Setelah melakukan konversi dari gambar sidik jari menjadi karakter ASCII, dilakukan proses pencocokan sidik jari terhadap daftar sidik jari yang terdapat di dalam basis data. Pencocokan sidik jari tersebut dapat dilakukan dengan algoritma pencocokan string KMP (Knuth-Morris-Pratt) ataupun BM (Boyer-Moore). Jika tidak ada satupun sidik jari di dalam basis data yang *exact match* dengan sidik jari masukan pengguna, digunakan sidik jari yang paling mirip, selama sidik jari tersebut masih di atas suatu ambang batas kemiripan. Metode yang dipakai untuk menentukan tingkat kemiripan adalah *Levenshtein Distance*.

Setelah pencocokan sidik jari selesai, kita akan memperoleh nama pemilik sidik jari. Nama pemilik sidik jari digunakan dalam mencari biodata lengkap pemilik sidik jari. Namun, biodata pemilik tidak dapat langsung dicari karena tabel biodata di dalam basis data mengalami *corrupt* pada atribut nama. Untuk menangani data nama yang *corrupt*, terlebih dahulu dilakukan percobaan untuk mengembalikan data *corrupt* tersebut menjadi data yang sebenarnya dengan menggunakan regex (*regular expression*). Setelah itu, dilakukan *pattern matching* untuk mencocokan nama dengan biodata dengan algoritma

KMP atau BM, seperti ketika mencocokan sidik jari. Karena tidak semua data *corrupt* dapat dikembalikan menjadi sama persis seperti keadaan sebenarnya, mungkin tidak ada nama yang *exact match*. Untuk mengatasinya, digunakan nama yang paling mirip selama nama tersebut masih di atas suatu ambang batas kemiripan. Sama seperti ketika mencocokkan sidik jari, metode yang dipakai untuk menentukan tingkat kemiripan adalah *Levenshtein Distance*.

Setelah melakukan tahapan tersebut, diperoleh biodata pemilik sidik jari. Sidik jari beserta biodata yang bersesuaian ditampilkan kepada pengguna. Selain itu, ditampilkan juga waktu pencarian serta persentase kecocokan.

B. Proses penyelesaian solusi dengan algoritma KMP dan BM

Proses penyelesaian solusi dengan algoritma KMP:

1. Input sidik jari yang telah dikonversi menjadi ASCII berperan sebagai *pattern*, sedangkan data sidik jari di dalam basis data berperan sebagai *text* yang akan dicocokkan.
2. Berdasarkan *pattern* tersebut, dibuat sebuah tabel lsp/*border function*. Tabel ini berperan dalam pergeseran *pattern* ketika terjadi *mismatch* dalam pencocokan.
3. Setelah tabel terbentuk, pencocokan dapat mulai dilakukan. Pencocokan dimulai dengan meletakkan pattern di ujung paling kiri teks. Pencocokan dilakukan mulai dari bagian kiri *pattern* (dimulai dari i = 0 dan j = 0, i adalah indeks yang menunjuk suatu karakter pada *text*, j adalah indeks yang menunjuk suatu karakter pada *pattern*).
4. Jika karakter pada *text* cocok dengan karakter pada *pattern*, lanjutkan pencocokan untuk karakter di sebelah kanannya. Jika tidak cocok/*mismatch*, gunakan tabel *border function* untuk menentukan pergeseran *pattern* berdasarkan posisi *mismatch* pada *pattern* (nilai j). Setelah pattern digeser, lanjutkan kembali pencocokan untuk karakter-karakter berikutnya.
5. Jika pencocokan berhasil sampai karakter terakhir *pattern*, pencocokan dihentikan dan dikembalikan nilai indeks posisi *text* paling kiri yang *match* dengan *pattern* tersebut. Jika sampai akhir *text* tidak ditemukan posisi yang *match* dengan *pattern*, dikembalikan nilai -1 yang menandakan bahwa tidak ada bagian *text* tersebut yang cocok dengan *pattern*.

Proses penyelesaian solusi dengan algoritma BM:

1. Input sidik jari yang telah dikonversi menjadi ASCII berperan sebagai *pattern*, sedangkan data sidik jari di dalam basis data berperan sebagai *text* yang akan dicocokkan.
2. Berdasarkan *pattern* tersebut, dibuat sebuah tabel *last occurrence function*. Tabel ini berperan dalam pergeseran *pattern* ketika terjadi *mismatch* dalam pencocokan.
3. Setelah tabel terbentuk, pencocokan dapat mulai dilakukan. Pencocokan dimulai dengan meletakkan pattern di ujung paling kiri teks. Berbeda dengan KMP, pencarian dilakukan mulai dari bagian kanan pattern (dimulai dari i = m - 1 dan j = m - 1, i adalah indeks yang menunjuk suatu karakter pada *text*, j adalah indeks yang menunjuk suatu karakter pada *pattern*, m adalah banyaknya karakter pada *pattern*).
4. Jika karakter pada *text* cocok dengan karakter pada *pattern*, lanjutkan pencocokan untuk karakter di sebelah kirinya. Jika tidak cocok/*mismatch*, gunakan tabel *last occurrence function* untuk menentukan pergeseran *pattern* berdasarkan posisi *mismatch* pada *pattern* (nilai j). Terdapat 3 kondisi yang mungkin:
   - Jika karakter *text* yang *mismatch* terdapat di posisi sebelah kiri karakter pattern yang *mismatch*, geser *pattern* sehingga karakter *text* yang *mismatch* menjadi sejajar dengan karakter terakhir yang terdapat dalam pattern yang juga sama dengan karakter *text* tersebut.
   - Jika karakter *text* yang *mismatch* terdapat di posisi sebelah kanan karakter *pattern* yang *mismatch*, geser *pattern* ke kanan sejauh 1 karakter.
   - Jika karakter *text* yang *mismatch* tidak terdapat di dalam *pattern*, geser *pattern* ke kanan sehingga karakter paling kiri *pattern* tersebut terletak 1 karakter di sebelah kanan karakter *text* yang *mismatch*.

   Setelah pattern digeser, lanjutkan kembali pencocokan untuk karakter-karakter berikutnya.
5. Jika pencocokan berhasil sampai karakter awal *pattern*, pencocokan dihentikan dan dikembalikan nilai indeks posisi *text* paling kiri yang *match* dengan *pattern* tersebut. Jika sampai akhir *text* tidak ditemukan posisi yang *match* dengan *pattern*, dikembalikan nilai -1 yang menandakan bahwa tidak ada bagian *text* tersebut yang cocok dengan *pattern*.

C. Fitur fungsional dan arsitektur aplikasi desktop yang dibangun

Aplikasi desktop ini menawarkan beberapa fungsionalitas, yaitu:

1. Aplikasi dapat mencari sidik jari yang cocok dengan algoritma Knuth-Morris-Pratt (KMP) serta menampilkan list biodata pemilik sidik jari.
2. Aplikasi dapat mencari sidik jari yang cocok dengan algoritma Boyer-Moore (BM) serta menampilkan list biodata pemilik sidik jari.

Aplikasi desktop yang kami buat dibangun dengan bahasa pemrograman C# dengan *library/package* tambahan:

- Untuk pembuatan *Graphical User Interface* (GUI), kami menggunakan WPF.
- Untuk menginisiasi WPF kami menggunakan package .NET versi 6.0.
- Untuk backend kami menggunakan library tambahan yaitu Sixlabors untuk pengolahan citra.
- Untuk koneksi ke basis data, kami menggunakan MySql.Data.

D. Contoh ilustrasi kasus

Sebagai contoh ilustrasi kasus, misalkan program diberikan sebuah input sidik jari yang ingin dicocokkan beserta algoritma pencocokan yang ingin dipakai. Berikut ini adalah alur keberjalanan program:

1. Program mengambil sampel pixel dari gambar sidik sidik jari yang dimasukkan, kemudian melakukan konversi hingga menjadi karakter-karakter ASCII.
2. Sidik jari tersebut dicocokkan dengan setiap data sidik jari yang terdapat di dalam basis data. Algoritma pencocokan yang dipakai adalah algoritma yang dipilih oleh pengguna.
3. Jika tidak ada sidik jari yang *exact match*, dicari sidik jari yang paling mirip dengan sidik jari masukan. Cara perhitungan yang digunakan untuk menentukan tingkat kemiripan adalah *Levenshtein Distance*.
4. Setelah hasil pencocokan sidik jari sebelumnya, kita akan mendapatkan nama pemilik sidik jari.
5. Berdasarkan nama yang didapat, program akan mencari biodata lengkap pemilik sidik jari tersebut. Akan tetapi, karena basis data mengalami *corrupt*, diperlukan percobaan untuk memperbaiki data yang *corrupt* dengan regex.

6. Setelah data *corrupt* diperbaiki, dilakukan pencocokan antara nama dengan biodata pemilik sidik jari dengan algoritma yang sama dengan algoritma yang dipakai saat pencocokan sidik jari. Jika tidak ada yang *exact match*, dicari biodata dengan nama yang paling mirip dengan nama yang dicari. Cara perhitungan yang digunakan untuk menentukan tingkat kemiripan adalah *Levenshtein Distance*.
7. Setelah memperoleh sidik jari dan biodata pemilik sidik jari yang cocok, sidik jari beserta biodata tersebut ditampilkan kepada pengguna.

BAB IV
IMPLEMENTASI DAN PENGUJIAN

A. Spesifikasi teknis program

Spesifikasi teknis program (struktur data, fungsi, ataupun prosedur yang dibangun).

Pada program kami, terdapat beberapa penggunaan struktur data, fungsi, dan prosedur yang dibuat untuk keberjalanan program. Secara garis besar akan dijelaskan berdasarkan spesifikasi *Backend* dan *Graphical User Interface*.

1. *Backend* :

Berikut adalah struktur data yang digunakan pada program utama di *backend*:

a. **List<>**, digunakan untuk menampung data-data dalam jumlah banyak seperti hasil, list biodata, dll.

b. **StringBuilder**, digunakan untuk memanipulasi data-data string.

Berikut adalah kelas-kelas yang digunakan pada program utama di *backend*:

a. **Program**, kelas ini digunakan sebagai *main handler* program utama *backend*.

b. **Algorithm**, kelas ini digunakan sebagai pemroses data-data input berdasarkan algoritma KMP atau BM.

c. **Database**, kelas ini digunakan sebagai penghubung antar program dengan database.

d. **Encryption**, kelas ini digunakan untuk mengenkripsi mendekripsi data-data yang berada di database.

Berikut ini adalah fungsi/prosedur yang berkaitan dengan perbaikan data nama yang *corrupt* menggunakan regex:

a. **FixCorruptedName(string corruptedName)**, fungsi utama yang mengeksekusi fungsi-fungsi lainnya yang berkaitan dengan perbaikan data nama yang *corrupt*.

b. **FixNumber(string corruptedName)**, berfungsi untuk mencari karakter-karakter angka untuk dikonversi menjadi huruf yang sebenarnya.

c. **NumberToChar(Match number)**, berfungsi untuk mengkonversi angka menjadi huruf yang sebenarnya.

d. **FixUpperLower(string input)**, berfungsi untuk mencari penggunaan *uppercase* dan *lowercase* yang tidak normal. *Uppercase* hanya digunakan untuk karakter pertama pada setiap kata, sisanya menggunakan *lowercase*.

e. **CapitalizeWord(Match word)**, berfungsi untuk memperbaiki huruf agar huruf *uppercase* hanya terdapat di karakter pertama setiap kata dan sisanya adalah *lowercase*.

2. ***Graphical User Interface* :**

Secara umum, struktur data spesifik di bagian GUI hanyalah sebagai *placeholder* untuk data yang akan diterima dari *Backend* dan sisanya merupakan fitur-fitur dari WPF. Struktur data yang digunakan berupa :

a. ***ObservableCollection*<CardItem>** (sebagai source dari tampilan ListView pada GUI).
b. **CardItem** (kelas buatan untuk menampung data-data yang akan ditampilkan di GUI melalui binding)
c. **Array of String** (result yang digunakan untuk parameter yang di-*pass* ke window *details*)
d. **String** (imagePathQuery : *path* dari image yang akan diinput oleh program, method : sebagai parameter untuk program metode algoritma pencarian di *Backend*, imagePath : *path* dari image yang ditemukan oleh program).

Terdapat beberapa fungsi dan prosedur tambahan (diluar inisialisasi WPF) yang kami gunakan untuk keberjalanan program. Fungsi dan prosedur tersebut berupa:

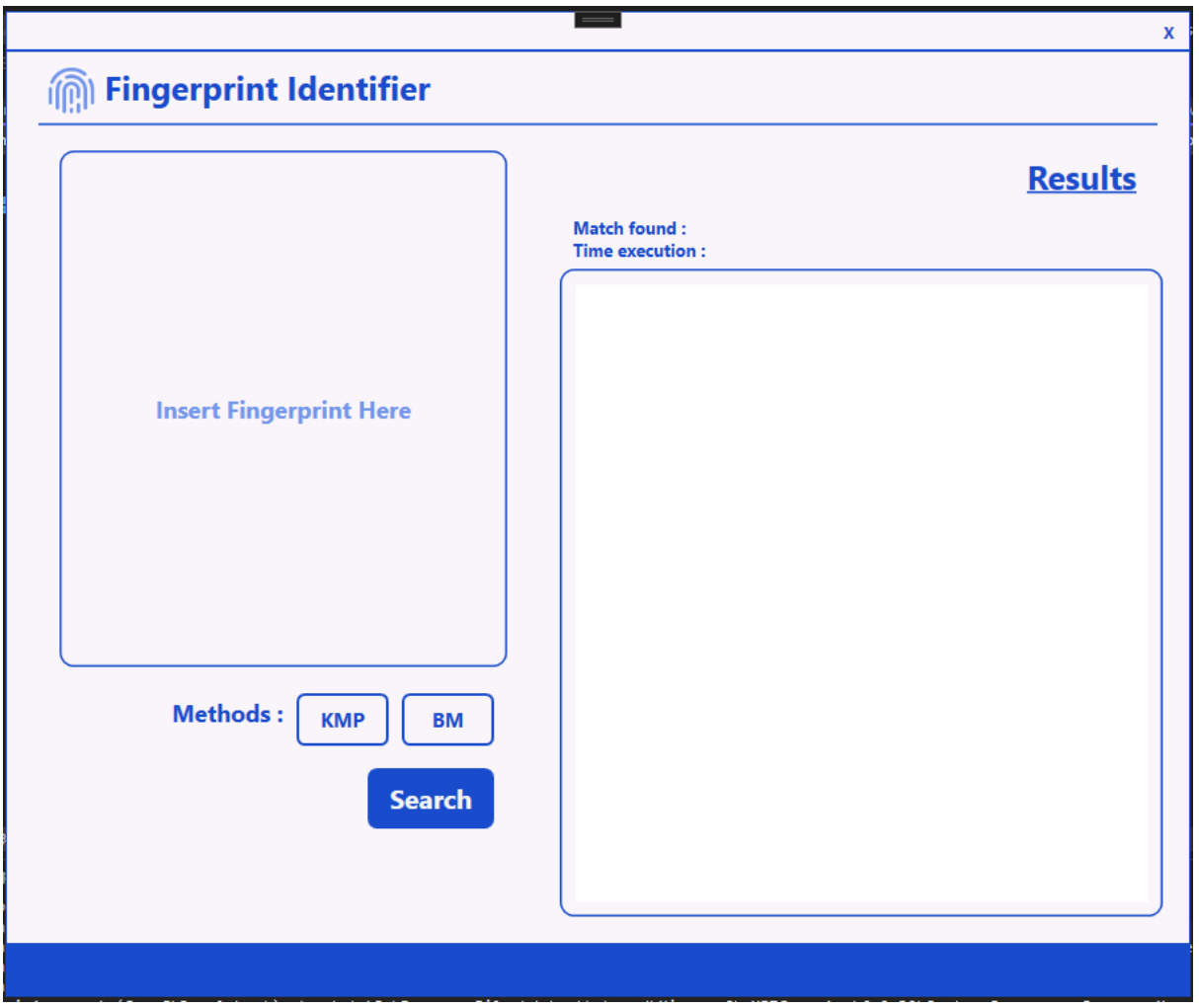
a. **OnPropertyChange(string propertyName)**, prosedur ini berfungsi untuk meng-*invoke* perubahan pada suatu data yang di-*bind* kepada GUI untuk menampilkan perubahannya pada GUI.
b. **Window_MouseLeftButton(object sender, MouseEventButtonArgs e)**, prosedur ini berfungsi sebagai fitur *window drag* yang diimplementasikan pada *custom window bar*.
c. **btnClose_Click(object sender, RoutedEventArgs e)**, prosedur ini berfungsi untuk menutup window dan mematikan program utama pada *custom exit button* di *custom window bar*.
d. **btnOpenImage_Click(object sender, RoutedEventArgs e)**, prosedur ini berfungsi sebagai fitur *input image* yang diimplementasikan pada bagian *placeholder image* di GUI.
e. **btnBM_Click(object sender, RoutedEventArgs e),** prosedur ini berfungsi untuk mengubah atribut *method* pada kelas *MainWindow* menjadi “BM”.

f. **btnKMP_Click(object sender, RoutedEventArgs e)**, prosedur ini berfungsi untuk mengubah atribut *method* pada kelas *MainWindow* menjadi "KMP".
g. **Details_Click(object sender, RoutedEventArgs e)**, prosedur ini berfungsi untuk menampilkan window tambahan untuk detail dari hasil penemuan.
h. **btnSearch_Click(object sender, RoutedEventArgs e)**, prosedur ini berfungsi untuk meng-*trigger* program pencarian pada *Backend* yang akan menginisiasi kelas *program* dan akan meng-*invoke* metode program.mainProgram(this.Method, 80.0, this.imagePathQuery), dengan this.Method adalah metode algoritma yang akan digunakan, 80.0 adalah batas atas persentase kemiripan, dan this.imagePathQuery adalah *path* dari *input image*.
i. **string NormalizePath(string path)**, fungsi ini berguna untuk melakukan *formatting* dari *image path* yang akan diterima dari *Backend* agar dapat diterima oleh URI *receiver*-nya.

B. Penjelasan tata cara penggunaan program

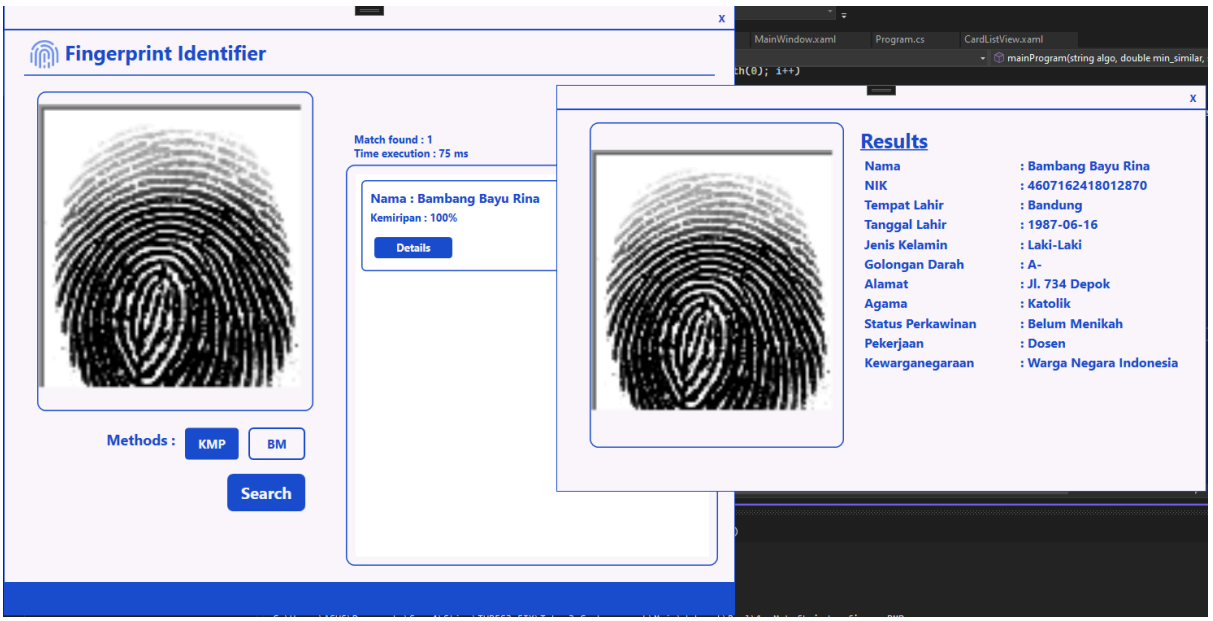
Penjelasan tata cara penggunaan program (interface program, fitur-fitur yang disediakan program, dan sebagainya).

Berikut adalah tata cara penggunaan program :



1. Klik “Insert Fingerprint Here” untuk menginput file *image* fingerprint yang ingin dicari.
2. Klik tombol “KMP” atau “BM” untuk memilih algoritma pencariannya, tombol-tombol ini merupakan *toggle button* yang berada di satu *grouping* sehingga tidak akan bisa memilih dua metode secara bersamaan.
3. Saat semua input sudah diterima (*image* dan metode), klik tombol “Search” untuk memulai pencarian.

4. Setelah menemukan hasil, program akan menampilkan hasil dengan persentase tertinggi secara kemiripan citra dan kemiripan nama. User dapat melihat detail dari biodatanya dengan meng-klik “Details”.



5. User akan mendapatkan tampilan window baru yang menyertai semua informasi detail terkait biodata pemilik sidik jari tersebut.

C. Hasil Pengujian

Kami melakukan pengujian pada tiga citra sidik jari dengan 20 sampel dari masing-masing algoritma. Berikut adalah hasilnya:

1. Citra 1 :
    a. BM (ms) : 66, 73, 72, 71, 64, 72, 68, 68, 66, 65, 70, 66, 68, 69, 72, 71, 69, 66, 64, 61; rata-rata : 68.05 ms.
    b. KMP (ms) : 67, 68, 67, 74, 72, 73, 69, 71, 73, 69, 68, 70, 71, 75, 70, 71, 69, 67, 66, 69; rata-rata : 68.1 ms.
2. Citra 2 :
    a. BM (ms) : 55, 59, 59, 61, 68, 70, 72, 71, 59, 58, 56, 79, 58, 64, 65, 66, 74, 70, 68, 62; rata-rata : 65.7 ms.
    b. KMP (ms) : 75, 72, 74, 79 , 69, 76, 80, 83, 73, 79, 84, 76, 81, 79, 66, 70, 71, 84, 77, 69; rata-rata : 74.85 ms.
3. Citra 3 :
    a. BM (ms) : 34, 40, 66, 42, 48, 47, 33, 31, 39, 58, 49, 43, 63, 48, 43, 49, 38, 42, 62, 44; rata-rata : 42.7 ms.
    b. KMP (ms) : 58, 59, 60, 55, 59, 56, 56, 52, 59, 66, 54, 53, 58, 55, 50, 52, 55, 68, 53, 51; rata-rata : 55.6 ms.

D. Analisis Hasil Pengujian

Hasil pengujian menunjukkan bahwa perbedaan waktu eksekusi algoritma BM dan KMP lumayan signifikan pada Citra 2 dan Citra 3. Hal ini masuk akal karena KMP secara waktu eksekusi akan lebih lambat dari BM apabila terdapat karakter pada string yang relatif banyak variasinya. Algoritma BM akan lebih cepat karena pada algoritmanya menerapkan *smart jumping* yang akan mempercepat proses algoritmanya karena tidak seperti KMP yang iterasi string satu per satu, BM akan melakukan lompat indeks pada string yang mempercepat algoritma. Apabila dilihat dari ketiga citra, terdapat perbedaan waktu yang lumayan signifikan, hal ini kemungkinan disebabkan karena area sidik jari yang dicari dari Citra 3 relatif lebih kecil dari Citra 2 dan Citra 1 sehingga pemrosesan citra jauh lebih cepat.

BAB V
PENUTUP

A. Kesimpulan

Kesimpulannya adalah Algoritma KMP dan BM adalah salah satu algoritma pencocokan string, algoritma ini dapat digunakan dalam pencarian pattern dari sebuah string dalam kasus ini adalah nilai ASCII dari citra sidik jari. Pada program ini, algoritma BM dan KMP digunakan untuk mencari kepemilikan sidik jari dan mengambil biodata pemilik sidik jari tersebut. Program ini sangat berguna dalam banyak kasus penyelesaian masalah kehidupan, salah satu kegunaannya adalah pada bidang forensik. Pada bidang forensik, lokasi kejadian tindakan kriminal biasanya akan dicari jejak-jejak dari pelaku yang biasanya adalah sidik jari yang ditinggal pada gagang pintu ataupun tubuh korban. Algoritma ini dapat menentukan identitas dari pelaku dengan mencocokan sidik jari dengan database kependudukan. Pada tugas besar ini, kami memahami bahwa bidang keinformatikaan dan pemrosesan citra sangat berguna dalam menyelesaikan permasalahan masyarakat.

B. Saran

Kami menyarankan mempelajari lebih mendalam mengenai bahasa pemrograman yang digunakan untuk mengurangi kebingungan saat mengimplementasikan algoritma-algoritma yang diperlukan. Selain itu, perlu ditingkatkan pula komunikasi antaranggota dalam pengerjaan tugas.

C. Tanggapan

Pembuatan aplikasi ini cukup menantang karena kami perlu banyak belajar hal baru, mulai dari bahasa pemrograman, koneksi ke basis data, sampai pembuatan GUI. Terlebih lagi, awalnya durasi pengerjaan pembuatan aplikasi ini cukup terbatas. Karena batas waktu pengerjaan yang diperpanjang, kami dapat membuat aplikasi ini menjadi lebih sempurna.

D. Refleksi

Hasil refleksi kelompok kami adalah tugas ini sangat membutuhkan komunikasi di dalam kelompok untuk saling membantu, terutama saat membuat GUI dan menghubungkan

antara GUI dengan *logic* utama program. Proses tersebut tidak sederhana dan terkadang mengalami *stuck* karena cukup rumitnya manajemen *project* pada IDE Visual Studio.

LAMPIRAN

Tautan repository GitHub:

https://github.com/Akmal2205/Tubes3_C-sharp-apek

DAFTAR PUSTAKA

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

https://www.kaggle.com/datasets/ruizgara/socofing

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

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