LAPORAN TUGAS BESAR 3

IF2211 Strategi Algoritma

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

Disusun oleh

Shafiq Irvansyah	:	13522003
Sa'ad Abdul Hakim	:	13522092

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG
2024

Daftar Isi

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 pattern yang unik dan tidak dapat ditiru, sehingga cocok untuk digunakan sebagai identitas individu.

Pattern matching merupakan teknik penting dalam sistem identifikasi sidik jari. Teknik ini digunakan untuk mencocokkan pattern sidik jari yang ditangkap dengan pattern 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, Akan diimplementasikan 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. Knuth-Morris-Pratt (KMP)

Algoritma string matching Knuth-Morris-Pratt (KMP) merupakan teknik efisien untuk mencari keberadaan sebuah string atau pattern dalam string lain yang lebih besar, sering diterapkan dalam berbagai aplikasi seperti pencarian teks, pengolahan data biologi, dan sistem keamanan. Algoritma ini dirancang untuk mengatasi kelemahan algoritma pencarian string naif, yang memiliki kompleksitas waktu terburuk O(mn), di mana m adalah panjang pattern dan n adalah panjang teks. KMP mengurangi waktu pencarian dengan menghindari pengulangan pencocokan karakter yang tidak perlu melalui penggunaan tabel lompatan (failure function atau partial match table).

Tabel lompatan, yang dibangun berdasarkan pattern sebelum proses pencocokan dimulai, merupakan kunci efisiensi KMP. Tabel ini menentukan seberapa jauh pattern harus digeser ketika terjadi kegagalan dalam pencocokan di suatu posisi. Dengan menggunakan informasi dari tabel lompatan, KMP dapat melanjutkan pencarian dari titik kegagalan tanpa perlu memulai dari awal, sehingga mengurangi jumlah perbandingan yang diperlukan dan mencapai kompleksitas waktu yang optimal, yaitu O(n + m).

Selain itu, berkat efisiensi dan kecepatannya, KMP sangat berguna dalam aplikasi yang membutuhkan pencarian dan pemrosesan string secara cepat dan akurat, seperti dalam editor teks dan database. Kompleksitas waktu linear yang ditawarkan oleh algoritma KMP menjadikannya pilihan yang lebih baik dibandingkan dengan algoritma pencarian naif, terutama untuk teks yang sangat panjang atau pattern yang sering berulang. Dengan demikian, algoritma KMP menjadi sangat relevan dan penting dalam pengembangan software yang memproses string dan pattern secara intensif.

2.2. Boyer-Moore (BM)

Algoritma Boyer-Moore (BM) adalah metode pencarian string yang sangat efisien, terutama efektif untuk teks yang panjang karena menggunakan heuristik yang meminimalkan jumlah perbandingan karakter. Keunggulan utama dari algoritma ini terletak pada dua heuristiknya: heuristik bad character dan heuristik good suffix, yang memungkinkan penggeseran pattern yang lebih jauh ketika terjadi ketidakcocokan.

Heuristik bad character berfokus pada respons terhadap ketidakcocokan karakter. Algoritma memeriksa karakter yang tidak cocok dalam teks dengan pattern, dan berdasarkan posisi terakhir kemunculan karakter tersebut dalam pattern, algoritma menggeser pattern ke kanan. Jika karakter yang tidak cocok tidak ada dalam pattern,

algoritma menggeser seluruh pattern melewati karakter tersebut, yang secara signifikan mengurangi jumlah perbandingan yang perlu dilakukan.

Heuristik good suffix mengatasi situasi di mana akhir dari pattern cocok dengan sebagian teks tetapi ada ketidakcocokan sebelumnya. Dalam kasus ini, algoritma memanfaatkan cocokan bagian akhir ini untuk mencari kesamaan lain dalam pattern dan menggeser pattern berdasarkan informasi tersebut, lagi-lagi untuk mengoptimalkan jumlah pergeseran dan meminimalkan perbandingan.

Kombinasi dari kedua heuristik ini membuat Boyer-Moore terutama efektif dalam kasus di mana pattern jarang muncul dalam teks, memungkinkan algoritma untuk melompati banyak bagian teks yang tidak mungkin mengandung pattern tanpa melewatkan potensi cocokan. Oleh karena itu, Boyer-Moore sering digunakan dalam aplikasi yang memerlukan pencarian teks yang cepat dan efisien, mengurangi waktu pencarian secara dramatis dibandingkan dengan metode pencarian string lainnya.

2.3. Regex

Regular expression, atau regex, adalah alat yang sangat kuat untuk pencarian dan manipulasi string berdasarkan pattern tertentu. Regex memungkinkan pengguna untuk mendefinisikan suatu pattern yang kompleks untuk pencarian atau pemisahan informasi dalam teks, membuatnya menjadi instrumen yang sangat berharga dalam pemrograman dan pengolahan data.

Konsep utama di balik regex adalah penggunaan sintaks khusus yang memungkinkan pencocokan berbagai kombinasi dan pattern karakter dalam sebuah string. Regex bekerja dengan mendefinisikan aturan atau pattern yang terdiri dari karakter literal dan meta-karakter yang memiliki fungsi khusus. Beberapa meta-karakter yang paling umum termasuk titik (.) yang mewakili karakter tunggal apapun, asterisk (*) yang menunjukkan pengulangan karakter sebelumnya nol atau lebih kali, dan tanda tanya (?) yang menandakan karakter sebelumnya adalah opsional.

Regex juga mendukung pembuatan grup dan subgrup dengan tanda kurung, yang memungkinkan pengguna untuk mengambil bagian tertentu dari teks yang cocok dengan pattern. Selain itu, regex menyediakan kemampuan untuk mencari berbagai variasi dari suatu pattern dengan menggunakan alternasi, ditandai dengan tanda pipa (|), yang berfungsi mirip dengan operator "atau" dalam logika.

Dalam konteks penggunaannya, regex dapat diterapkan dalam berbagai tugas pemrograman seperti validasi input, parsing data, dan penggantian teks. Alat ini sangat

penting dalam situasi di mana diperlukan fleksibilitas dan keakuratan dalam mencari dan mengelola string, seperti dalam pemrosesan log file, analisis data teks, dan pengembangan aplikasi web.

Meskipun regex menawarkan kemampuan yang sangat luas dan fleksibel, penggunaannya harus hati-hati karena kompleksitas dan ketelitian sintaks yang bisa mempengaruhi performa aplikasi. Salah satu tantangan dalam menggunakan regex adalah risiko penulisan ekspresi yang dapat menyebabkan proses pencarian menjadi sangat lambat, terutama pada teks yang sangat panjang atau pattern yang kompleks, fenomena yang dikenal sebagai "catastrophic backtracking."

Oleh karena itu, penggunaan regex memerlukan pemahaman yang baik tentang bagaimana ekspresi dibangun dan dioptimalkan agar dapat dijalankan dengan efisien. Dengan kemampuan untuk menyusun pattern pencarian yang sangat spesifik, regex merupakan komponen penting dalam toolset setiap pengembang dan analis data, memberikan kontrol yang sangat detail atas pencarian dan manipulasi string.

2.4. Levenshtein Distance

Levenshtein Distance adalah sebuah algoritma yang digunakan untuk mengukur perbedaan antara dua string dengan menghitung jumlah operasi yang diperlukan untuk mengubah satu string menjadi string lainnya. Operasi yang dimaksud mencakup penyisipan, penghapusan, dan penggantian karakter. Algoritma ini sering digunakan dalam berbagai aplikasi seperti pengecekan ejaan, pencocokan pola, dan analisis teks.

Tahapan algoritma Levenshtein Distance adalah sebagai berikut:

1. **Inisialisasi Matriks**

   Mulai dengan membuat sebuah matriks berukuran (m+1) x (n+1), di mana m adalah panjang string pertama dan n adalah panjang string kedua. Baris dan kolom pertama dari matriks ini diisi dengan indeksnya masing-masing, mewakili kasus dasar di mana satu string dikosongkan.

2. **Pengisian Matriks**

   Iterasi melalui setiap elemen matriks dimulai dari elemen [1,1]. Untuk setiap elemen matriks pada posisi [i,j], hitung biaya operasi untuk penyisipan, penghapusan, dan penggantian karakter. Biaya untuk penyisipan dan penghapusan adalah 1, sementara biaya penggantian adalah 1 jika karakter pada posisi tersebut berbeda, dan 0 jika sama.

3. **Perhitungan Minimum**

Untuk setiap elemen [i,j], pilih nilai minimum dari tiga kemungkinan: nilai elemen di sebelah kiri (penyisipan), nilai elemen di atasnya (penghapusan), dan nilai elemen diagonal (penggantian) ditambah biaya operasi yang sesuai.

4. **Pengisian Matriks Secara Iteratif**

   Lanjutkan proses ini hingga seluruh matriks terisi. Nilai pada elemen [m,n] dari matriks ini akan menjadi Levenshtein Distance antara kedua string.

Dengan menggunakan pendekatan dinamis ini, algoritma Levenshtein Distance mampu menghitung jarak edit dengan kompleksitas waktu O(m*n), yang efisien untuk kebanyakan aplikasi praktis. Algoritma ini juga dapat dimodifikasi untuk berbagai kebutuhan, seperti penanganan kasus-kasus khusus atau penambahan operasi lainnya.

2.5. Pembangunan Aplikasi

Aplikasi ini akan dikembangkan menggunakan Windows Presentation Foundation (WPF). Pemilihan WPF didasarkan pada kemudahan implementasinya communitynya yang sudah meluas, sehingga menyediakan berbagai library yang siap pakai. Aplikasi ini akan memuat data dari database yang telah dibuat sebelumnya. Database tersebut dapat dimuat dari folder yang berisi bitmap atau dari file dump yang sudah ada. Aplikasi ini akan menerima path gambar dengan format .bmp dan melakukan pencarian untuk menemukan citra sidik jari yang paling mirip, dengan mengakses data dari database.

Database aplikasi ini akan menggunakan MySQL dan akan dipreproses sebelum pencarian dilakukan. Tujuan dari preprocessing ini adalah untuk mempercepat proses pencocokan. Langkah-langkah preprocessing meliputi konversi gambar bitmap menjadi string ASCII 8-bit. Selain itu, jika menggunakan dump SQL dari skema yang sudah ada, skema tersebut akan diubah untuk memenuhi kebutuhan aplikasi.

Bab 3
Analisis Pemecahan Masalah

Sebuah citra terdiri dari elemen-elemen kecil yang disebut piksel. Setiap piksel mengandung tiga komponen warna utama: merah (red), hijau (green), dan biru (blue), yang masing-masing diwakili oleh nilai tertentu. Dalam implementasi *pattern matching*, cukup dibutuhkan informasi tentang apakah piksel tersebut berwarna hitam atau putih. Oleh karena itu, citra tersebut harus diubah menjadi skala abu-abu (grayscale) terlebih dahulu menggunakan rumus: *(pixel Red × 0.3) + (pixel Green × 0.59) + (pixel Blue × 0.11)*. Setelah diubah, penentuan apakah sebuah piksel adalah hitam atau putih dilakukan dengan menetapkan nilai kritis sebesar 128, karena rentang nilai skala abu-abu adalah 0–256. Jika nilai piksel ≤ 128, maka piksel tersebut dianggap hitam; jika lebih, dianggap putih. Nilai dari semua piksel tersebut kemudian akan dikonversi menjadi sebuah string yang hanya berisi data biner.

Dalam implementasi *pattern matching*, penggunaan string yang hanya berisi data biner dapat menyebabkan proses pencocokan pattern menjadi lebih lambat. Hal ini terjadi karena dengan kosakata yang terbatas pada hanya dua karakter (0 dan 1), jumlah pemeriksaan karakter yang perlu dilakukan cenderung meningkat. Sehiingga, proses pencocokan pattern akan memerlukan lebih banyak iterasi untuk menemukan kecocokan yang tepat antara *pattern* dan teks.

3.1. Pattern matching dengan KMP (Knuth-Morris-Pratt)

Pertama-tama, program akan menerima sebuah string yang berisi ASCII 8-bit dari citra sidik jari yang akan dicari. Program kemudian akan memilih 30 karakter yang terletak di bagian tengah string sebagai pattern yang akan digunakan untuk pencocokan. Dipilih 30 pixel paling tengah karena dalam berbagai kasus, foto sidik jari yang diberikan akan mencong/miring, sehingga posisi tengah gambar akan cenderung selalu meng-cover bagian dari sidik jari. Dengan menggunakan *database* yang telah diimplementasi, program akan mengiterasi semua citra sidik jari yang terdapat dalam dataset yang telah dipilih. Pada tiap iterasi, program akan melakukan proses *pattern matching* menggunakan algoritma Knuth-Morris-Pratt (KMP):

1. **Preprocessing Pattern (Membangun Fungsi Border):**

   Algoritma KMP pertama-tama memproses pattern untuk membuat fungsi border, yang juga dikenal sebagai fungsi kegagalan (failure function). Fungsi ini membantu menentukan berapa jauh geseran yang harus dilakukan saat terjadi ketidakcocokan. Fungsi border ini dibangun dengan cara membandingkan prefix dan suffix dari pattern tersebut. Untuk setiap posisi dalam pattern, fungsi border

menentukan panjang maksimum dari sub-pattern (substring) yang merupakan suffix sekaligus prefix.

2. **Pencocokan String:**

   Setelah fungsi border dibangun, algoritma KMP memulai pencocokan dari awal teks menggunakan pattern yang telah diproses. Algoritma membandingkan karakter demi karakter dari teks dengan pattern. Jika karakter cocok, algoritma melanjutkan ke karakter berikutnya dari pattern dan teks.

3. **Penanganan Ketidakcocokan:**

   Jika terjadi ketidakcocokan antara karakter teks dan pattern, algoritma menggunakan fungsi border untuk memutuskan berapa banyak karakter yang bisa dilewatkan (di-skip) tanpa kehilangan kemungkinan kecocokan. Ini menghindari pencocokan ulang dari karakter yang telah diketahui tidak cocok.

4. **Pergeseran Pattern:**

   Berdasarkan nilai dari fungsi border, pattern digeser ke kanan. Jika tidak ada sub-pattern yang cocok, geseran akan lebih besar, memungkinkan algoritma untuk melewatkan lebih banyak karakter.

5. **Ulangi Proses:**

   Proses pencocokan diulang hingga seluruh teks telah diperiksa atau pattern telah berhasil ditemukan dalam teks.

6. **Proses Lanjutan:**

   Jika pencocokan berhasil, algoritma mengembalikan nilai *True*. Jika tidak ada kecocokan untuk semua citra, maka algoritma mengembalikan *False*

3.2. Pattern matching dengan BM (Boyer-Moore)

Pertama-tama, program akan menerima sebuah string yang berisi ASCII 8-bit dari citra sidik jari yang akan dicari. Program kemudian akan memilih 30 karakter yang terletak di bagian tengah string sebagai pattern yang akan digunakan untuk pencocokan. Dipilih 30 pixel paling tengah karena dalam berbagai kasus, foto sidik jari yang diberikan akan mencong/miring, sehingga posisi tengah gambar akan cenderung selalu meng-cover bagian dari sidik jari. Dengan menggunakan *database* yang telah diimplementasi, program akan mengiterasi semua citra sidik jari yang terdapat dalam dataset yang telah dipilih. Pada tiap iterasi, program akan melakukan proses *pattern matching* menggunakan algoritma Boyer-Moore (BM):

1. **Looking-Glass Technique:**

   Pencarian dimulai dari akhir pattern, bukan awalnya. Ini memungkinkan pengecekan pattern dari kanan ke kiri terhadap teks, yang bisa lebih efisien dalam mendeteksi ketidakcocokan lebih cepat.

2. **Character-Jump Technique:**

   Ketika terjadi ketidakcocokan antara karakter di teks dengan karakter di pattern, algoritma menggunakan informasi dari karakter tersebut untuk menentukan seberapa jauh pattern harus digeser ke kanan. Geseran ini didasarkan pada posisi terakhir karakter yang tidak cocok di dalam pattern.

3. **Pembuatan Tabel Last Occurrence:**

   Algoritma memproses pattern untuk membuat "last occurrence function" yang mengindeks lokasi terakhir setiap karakter dalam pattern. Jika karakter tidak ada dalam pattern, maka akan diisi dengan -1.

4. **Pencocokan String:**

   Mulai dari indeks akhir pattern dan teks, algoritma membandingkan karakter secara mundur. Jika karakter cocok, proses terus berlanjut ke kiri sampai seluruh pattern dicocokkan atau terjadi ketidakcocokan. Jika ketidakcocokan terjadi, algoritma menentukan jumlah geseran menggunakan "last occurrence function" berdasarkan karakter dari teks yang tidak cocok.

5. **Pergeseran Pattern:**

   Algoritma menghitung geseran berdasarkan posisi terakhir dari karakter tidak cocok dalam pattern. Jika karakter tersebut tidak ada dalam pattern, pattern digeser melewati karakter tersebut. Jika ada, pattern digeser sehingga karakter terakhir yang cocok dalam pattern sejajar dengan karakter tersebut dalam teks.

6. **Proses lanjutan:**

   Jika pencocokan berhasil, algoritma mengembalikan nilai *True*. Jika tidak ada kecocokan untuk semua citra, maka algoritma mengembalikan *False*

3.3. Pencarian Similarity Tertinggi dengan Levenshtein Distance

Pada implementasi kami, kami menggunakan Levenshtein Distance untuk mencari citra sidik jari yang paling mirip jika algoritma KMP/BM tidak

menemukannya. Alasannya diipilih Levenshtein Distance dipilih dibanding algoritma lain adalah karena Levenshtein Distance cenderung akan menghasilkan persentase similarity yang paling akurat. Hal ini tentu sangat diperlukan untuk membandingkan jika terdapat beberapa sidik jari yang memiliki persentase similaritas yang cenderung mirip. Namun , algoritma ini memiliki *downside* waktu proses yang lebih lama. Akan tetapi hal tersebut dapat diatasi dengan mengatur threshold sebesar 55% sehingga program tidak akan mengiterasi keseluruhan database.

3.4. Pencarian Biodata dengan Regex

Pertama, program akan menerima sebuah string nama dan sebuah list string nama dengan kondisi korup atau rusak. Kemudian string nama dengan kondisi yang bagus tersebut akan diproses hingga menghasilkan sebuah pattern regex berdasarkan aturan bahasa alay seperti kemungkinan perubahan huruf besar atau kecil, perubahan huruf menjadi angka, dan hilangnya huruf vokal. Regex hasil dari string tersebut kemudian akan di cek ke seluruh string nama korup atau rusak yang ada pada list. Jika ditemukan nama korup yang cocok dengan regex tersebut, maka akan diambil data biodata pada database yang memiliki nama korup yang cocok regex yang dihasilkan. Data biodata tersebut kemudian akan ditampilkan dengan nama yang sudah dalam kondisi bukan bahasa alay.

3.5. Implementasi Aplikasi

Implementasi aplikasi menggunakan framework WPF (Windows Presentation Foundation) dipilih karena sudah terintegrasi dengan .NET. Ini memfasilitasi akses ke berbagai perpustakaan yang tersedia, yang mendukung pengembangan aplikasi yang lebih efisien. WPF juga terintegrasi secara langsung dengan Visual Studio, yang merupakan IDE standar dari Microsoft. Integrasi ini menyediakan alat pengembangan yang komprehensif, termasuk fasilitas debugging yang kuat, mempercepat proses desain, pengembangan, dan pengujian aplikasi. Selain itu penggunaanya juga lebih mudah dipahami jika dibanding dengan framework lain.

3.6. Fitur Fungsional dan Arsitektur Aplikasi

a. Fitur Fungsional:

Fitur fungsional aplikasi mencakup beberapa hal berikut

- GUI untuk memuat dan melihat data.
- Fasilitas untuk mengunggah gambar bitmap (.bmp).
- Pencocokan sidik jari masukan dengan database menggunakan algoritma Boyer-Moore
- Pencocokkan sidik jari masukan dengan database menggunakan algoritma Knuth-Morris-Pratt

- Pencocokkan sidik jari masukan dengan database menggunakan Levenshtein Distance

b. Arsitektur Aplikasi

Arsitektur aplikasi yang diimplementasikan mencakup beberapa hal berikut

- Penggunaan Windows Presentation Foundation (WPF) sebagai framework untuk membangun GUI.
- Integrasi dengan MySQL sebagai sistem manajemen basis data.
- Modul untuk preprocessing gambar, seperti konversi bitmap menjadi string ASCII 8-bit.
- Struktur modular yang memisahkan logika bisnis, logika presentasi, dan logika akses data untuk meningkatkan maintainability dan skalabilitas.

Bab 4
Implementasi dan Pengujian

4.1. Spesifikasi Teknis

a. Struktur Data

b. Fungsi dan Prosedur

Dalam pelaksanaannya, berbagai fungsi dan prosedur dibangun untuk memenuhi kebutuhan program. Pada bagian ini, akan disajikan fungsi dan prosedur yang penting dalam pembangunan logikanya.

Kelas KMP

```
using System;
using System.Collections.Generic;

namespace Tubes3_ImHim
{
    public class KMP
    {
        public static bool Search(string pattern, string text)
        {
            int p = pattern.Length;
            int t = text.Length;

            // Create the border function which will store the length of the longest prefix suffix
            int[] border_function = new int[p];
            int j = 0; // Index for pattern[]

            // Calculate the border_function[] array
            CreateBorderFunction(pattern, p, border_function);

            int i = 0; // Index for text[]
            while (i < t)
            {
                if (pattern[j] == text[i])
                {
                    j++;
                    i++;
                }

                if (j == p)
                {
                    return true; // Pattern found, return true
                }
                // Mismatch after j matches
                else if (i < t && pattern[j] != text[i])
                {
                    // Check if text[i] character is in pattern
                    if (!pattern.Contains(text[i]))
                    {
                        i += j + 1; // Continue from the position after the mismatch character
                        j = 0; // Reset j
                    }
                    else
                    {
                        if (j != 0)
                            j = border_function[j - 1];
                        else
                            i = i + 1;
                    }
                }
            }

            return false; // Pattern not found
        }

        // Function to calculate the border_function[] array
        private static void CreateBorderFunction(string pattern, int p, int[] border_function)
        {
            int len = 0;
            int i = 1;
            border_function[0] = 0; // border_function[0] is always 0

            while (i < p)
            {
                if (pattern[i] == pattern[len])
                {
                    len++;
                    border_function[i] = len;
                    i++;
                }
                else
                {
                    if (len != 0)
                    {
                        len = border_function[len - 1];
                    }
                    else
                    {
                        border_function[i] = 0;
                        i++;
                    }
                }
            }
        }
    }
}

```

Kelas BM

```cs
using System;
using System.Collections.Generic;

namespace Tubes3_ImHim
{
    public class KMP
    {
        public static bool Search(string pattern, string text)
        {
            int p = pattern.Length;
            int t = text.Length;

            // Create the border function which will store the length of the longest prefix suffix
            int[] border_function = new int[p];
            int j = 0; // Index for pattern[]

            // Calculate the border_function[] array
            CreateBorderFunction(pattern, p, border_function);

            int i = 0; // Index for text[]
            while (i < t)
            {
                if (pattern[j] == text[i])
                {
                    j++;
                    i++;
                }

                if (j == p)
                {
                    return true; // Pattern found, return true
                }
                // Mismatch after j matches
                else if (i < t && pattern[j] != text[i])
                {
                    // Check if text[i] character is in pattern
                    if (!pattern.Contains(text[i]))
                    {
                        i += j + 1; // Continue from the position after the mismatch character
                        j = 0; // Reset j
                    }
                    else
                    {
                        if (j != 0)
                            j = border_function[j - 1];
                        else
                            i = i + 1;
                    }
                }
            }

            return false; // Pattern not found
        }

        // Function to calculate the border_function[] array
        private static void CreateBorderFunction(string pattern, int p, int[] border_function)
        {
            int len = 0;
            int i = 1;
            border_function[0] = 0; // border_function[0] is always 0

            while (i < p)
            {
                if (pattern[i] == pattern[len])
                {
                    len++;
                    border_function[i] = len;
                    i++;
                }
                else
                {
                    if (len != 0)
                    {
                        len = border_function[len - 1];
                    }
                    else
                    {
                        border_function[i] = 0;
                        i++;
                    }
                }
            }
        }
    }
}

```

Kelas RegexClass

```cs
using System;
using System.Text.RegularExpressions;

namespace Tubes3_ImHim
{
    public class RegexClass
    {
        // static void Main()
        // {
        //     string Name = "Alay";
        //     string target = "4l4y";
        //     bool result = IsMatch(Name, target);
        //     Console.WriteLine(result);
        // }

        public static bool IsMatch(string Name, string target)
        {
            // Membersihkan input alayName dan target
            Name = CleanString(Name);
            target = CleanString(target);

            // Membuat regex berdasarkan input nama alay
            string regexPattern = GenerateRegex(Name);

            // Mencocokkan target string dengan regex
            Regex regex = new Regex(regexPattern, RegexOptions.IgnoreCase);

            return regex.IsMatch(target);
        }

        static string CleanString(string input)
        {
            // Menghilangkan spasi dan mengubah ke huruf kecil
            return input.Replace(" ", "").ToLower();
        }

        static string RemoveVokal(string input)
        {
            // Menghilangkan huruf vokal dari input
            return Regex.Replace(input, "[aiueo]", "");
        }

        static string GenerateRegex(string Name)
        {
            // konversi ke regex
            string pattern = Name.Replace("a", "[a4]?")
                                 .Replace("b", "[b8]")
                                 .Replace("e", "[e3]?")
                                 .Replace("g", "[g6]")
                                 .Replace("i", "[i1]?")
                                 .Replace("o", "[o0]?")
                                 .Replace("s", "[s5]")
                                 .Replace("t", "[t7]")
                                 .Replace("z", "[z2]")
                                 .Replace("u", "[u]?");
            return pattern;
        }
    }
}

```

Kelas LevenshteinDistance

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

namespace Tubes3_ImHim
{
    class LevenshteinDistance
    {

        public static int CalculateLevenshteinDistance(string s, string t)
        {
            int n = s.Length;
            int m = t.Length;
            int[] prev = new int[m + 1];
            int[] current = new int[m + 1];

            if (n == 0) return m;
            if (m == 0) return n;

            for (int j = 0; j <= m; j++)
                prev[j] = j;

            for (int i = 1; i <= n; i++)
            {
                current[0] = i;
                for (int j = 1; j <= m; j++)
                {
                    int cost = (s[i - 1] == t[j - 1]) ? 0 : 1;
                    current[j] = Math.Min(Math.Min(current[j - 1] + 1, prev[j] + 1), prev[j - 1] + cost);
                }
                prev = (int[])current.Clone();
            }

            return current[m];
        }

        public static float Calculate(string s, string t)
        {
            int maxLength = Math.Max(s.Length, t.Length);
            if (maxLength == 0)
                return 1.0f;

            int distance = CalculateLevenshteinDistance(s, t);
            return (1.0f - (float)distance / maxLength) * 100;
        }


    }
}

```

Kelas Biodata

```
public class Biodata
{
    public string NIK { get; set; }
    public string Nama { get; set; }
    public string TempatLahir { get; set; }
    public DateTime TanggalLahir { get; set; }
```

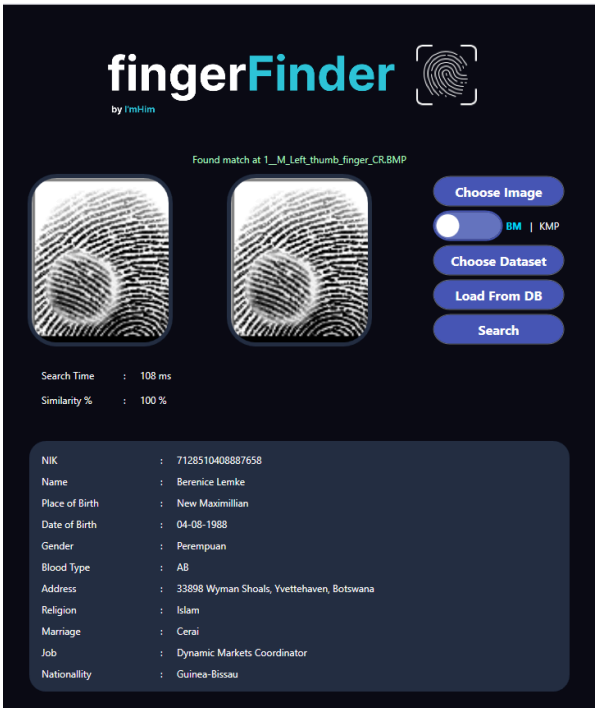
```
        public string JenisKelamin { get; set; }
        public string GolonganDarah { get; set; }
        public string Alamat { get; set; }
        public string Agama { get; set; }
        public string StatusPerkawinan { get; set; }
        public string Pekerjaan { get; set; }
        public string Kewarganegaraan { get; set; }
    }
```

4.2. Tata cara Penggunaan Program

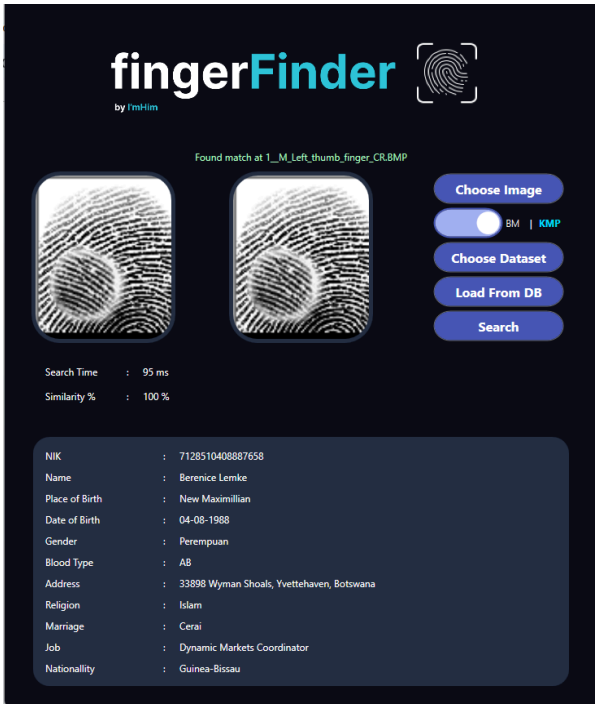
4.3. Pengujian

a. Uji 1

i.

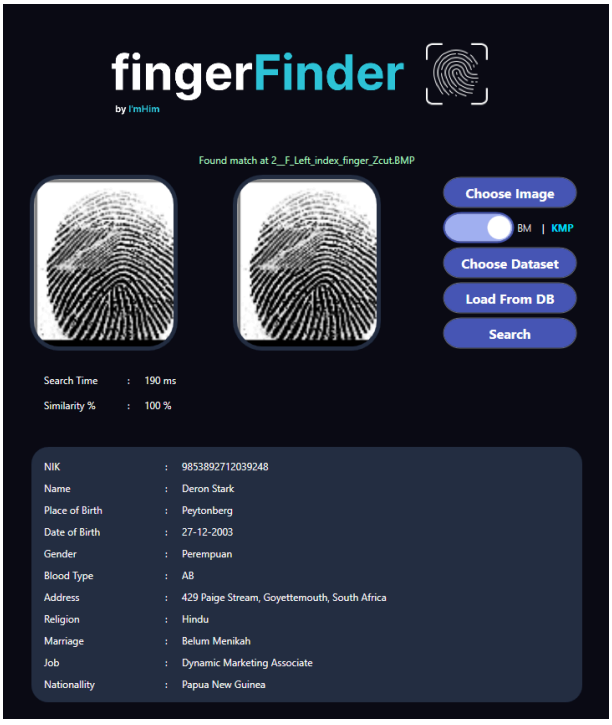


ii.

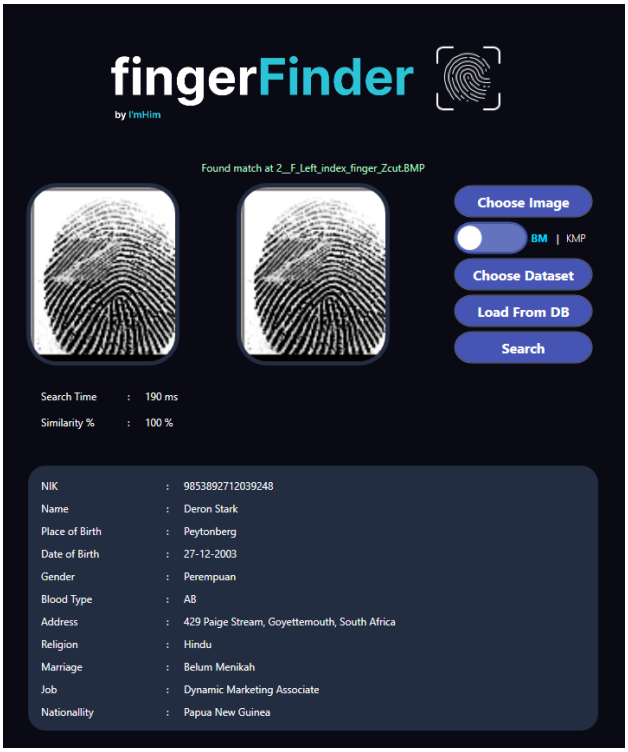

fingerFinder
by I'mHim
Found match at 1__M_Left_thumb_finger_CR.BMP
Choose Image
BM | KMP
Choose Dataset
Load From DB
Search
Search Time : 95 ms
Similarity % : 100 %
NIK : 7128510408887658
Name : Berenice Lemke
Place of Birth : New Maximillian
Date of Birth : 04-08-1988
Gender : Perempuan
Blood Type : AB
Address : 33898 Wyman Shoals, Yvettehaven, Botswana
Religion : Islam
Marriage : Cerai
Job : Dynamic Markets Coordinator
Nationallity : Guinea-Bissau

b. **Uji 2**

i.

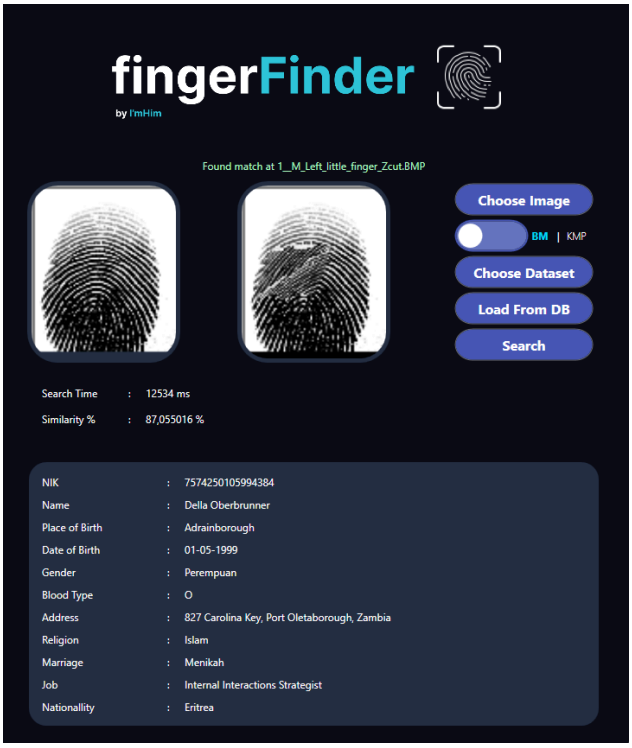

fingerFinder
by I'mHim
Found match at 2__F_Left_index_finger_Zcut.BMP
Choose Image
BM | KMP
Choose Dataset
Load From DB
Search
Search Time : 190 ms
Similarity % : 100 %
NIK : 9853892712039248
Name : Deron Stark
Place of Birth : Peytonberg
Date of Birth : 27-12-2003
Gender : Perempuan
Blood Type : AB
Address : 429 Paige Stream, Goyettemouth, South Africa
Religion : Hindu
Marriage : Belum Menikah
Job : Dynamic Marketing Associate
Nationallity : Papua New Guinea

ii.

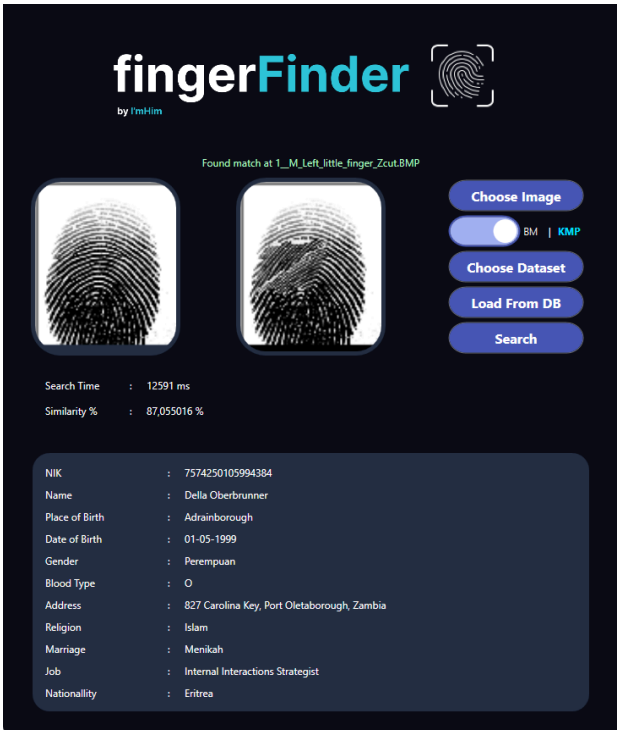

fingerFinder
by I'mHim
Found match at 2__F_Left_index_finger_Zcut.BMP
Choose Image
BM | KMP
Choose Dataset
Load From DB
Search
Search Time : 190 ms
Similarity % : 100 %
NIK : 9853892712039248
Name : Deron Stark
Place of Birth : Peytonberg
Date of Birth : 27-12-2003
Gender : Perempuan
Blood Type : AB
Address : 429 Paige Stream, Goyettemouth, South Africa
Religion : Hindu
Marriage : Belum Menikah
Job : Dynamic Marketing Associate
Nationallity : Papua New Guinea

c. **Uji 3**

i.

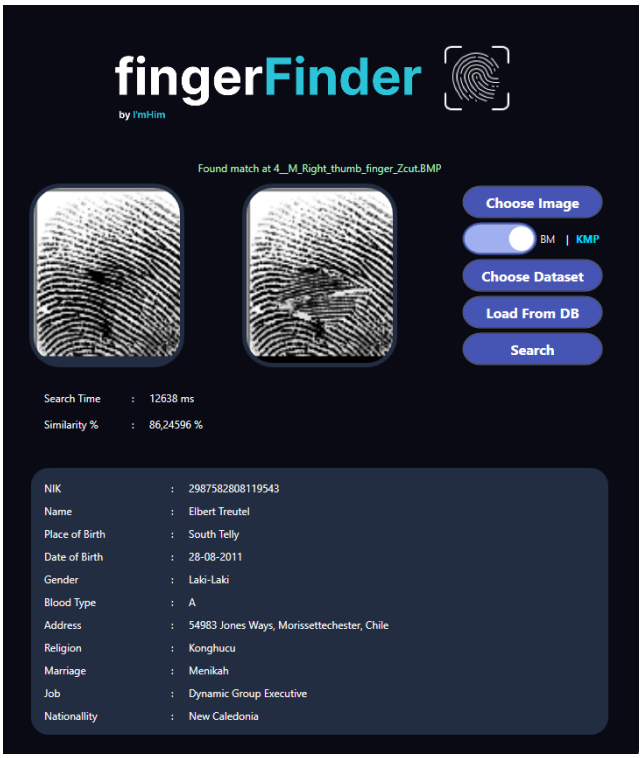

fingerFinder
by I'mHim
Found match at 1__M_Left_little_finger_Zcut.BMP
Choose Image
BM | KMP
Choose Dataset
Load From DB
Search
Search Time : 12534 ms
Similarity % : 87,055016 %
NIK : 7574250105994384
Name : Della Oberbrunner
Place of Birth : Adrainborough
Date of Birth : 01-05-1999
Gender : Perempuan
Blood Type : O
Address : 827 Carolina Key, Port Oletaborough, Zambia
Religion : Islam
Marriage : Menikah
Job : Internal Interactions Strategist
Nationallity : Eritrea

ii.

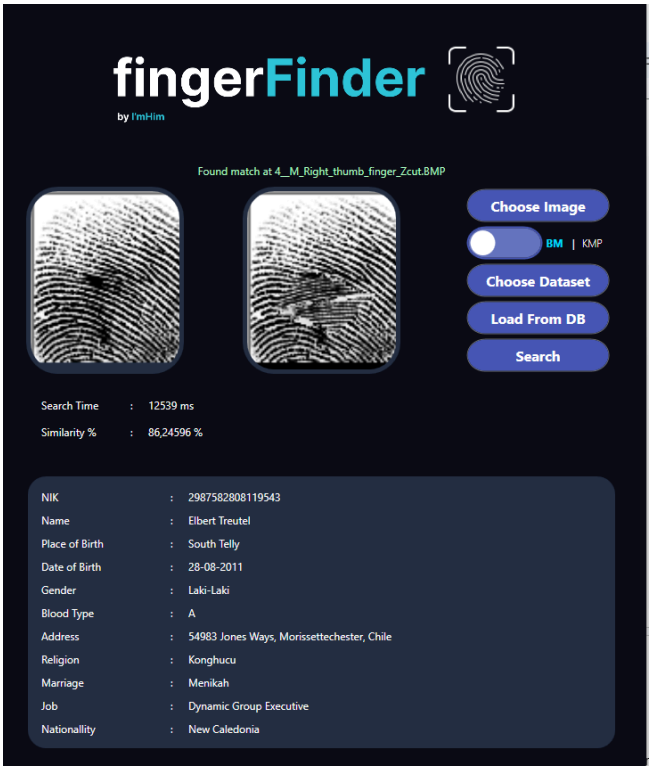

fingerFinder
by I'mHim
Found match at 1__M_Left_little_finger_Zcut.BMP
Choose Image
BM | KMP
Choose Dataset
Load From DB
Search
Search Time : 12591 ms
Similarity % : 87,055016 %
NIK : 7574250105994384
Name : Della Oberbrunner
Place of Birth : Adrainborough
Date of Birth : 01-05-1999
Gender : Perempuan
Blood Type : O
Address : 827 Carolina Key, Port Oletaborough, Zambia
Religion : Islam
Marriage : Menikah
Job : Internal Interactions Strategist
Nationallity : Eritrea

d. Uji 4

i.

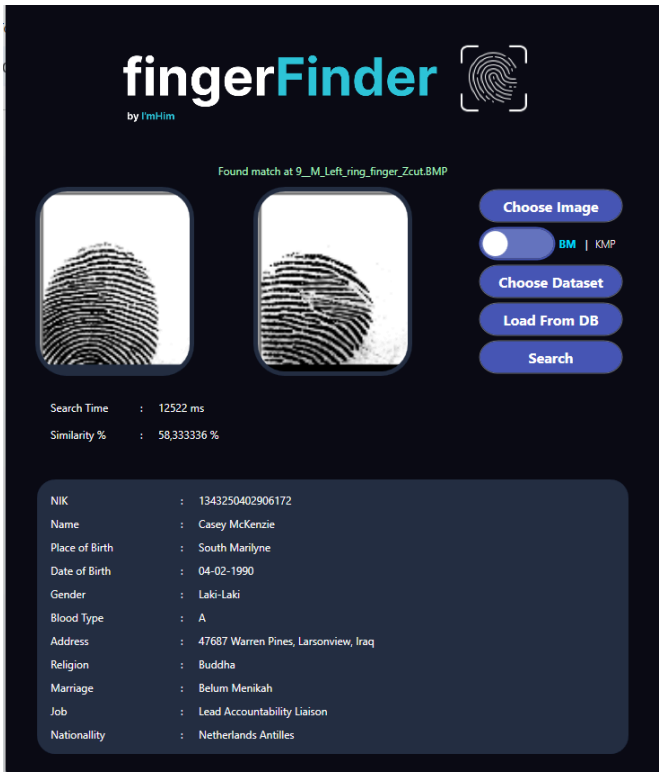

fingerFinder
by I'mHim
Found match at 4__M_Right_thumb_finger_Zcut.BMP
Choose Image
BM | KMP
Choose Dataset
Load From DB
Search
Search Time : 12638 ms
Similarity % : 86,24596 %
NIK : 2987582808119543
Name : Elbert Treutel
Place of Birth : South Telly
Date of Birth : 28-08-2011
Gender : Laki-Laki
Blood Type : A
Address : 54983 Jones Ways, Morissettechester, Chile
Religion : Konghucu
Marriage : Menikah
Job : Dynamic Group Executive
Nationallity : New Caledonia

ii.

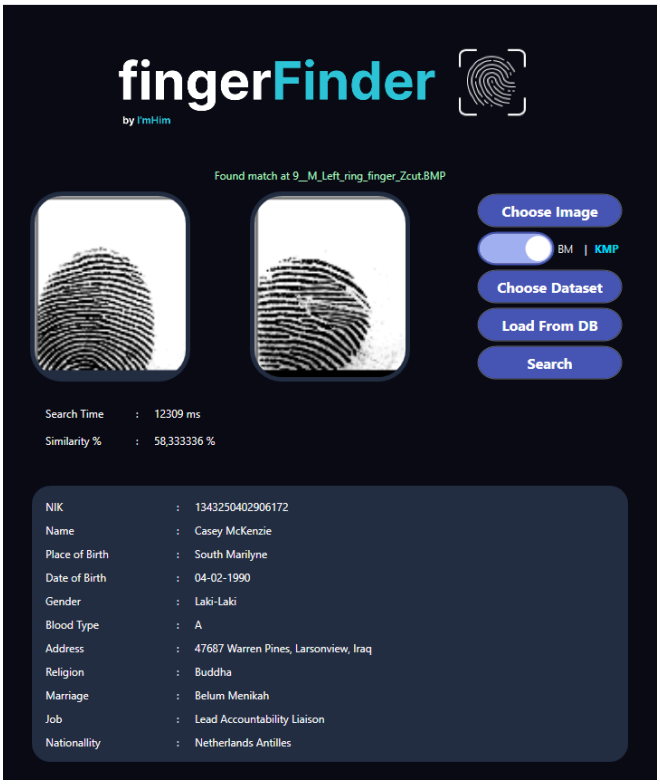

fingerFinder
by I'mHim
Found match at 4__M_Right_thumb_finger_Zcut.BMP
Choose Image
BM | KMP
Choose Dataset
Load From DB
Search
Search Time : 12539 ms
Similarity % : 86,24596 %
NIK : 2987582808119543
Name : Elbert Treutel
Place of Birth : South Telly
Date of Birth : 28-08-2011
Gender : Laki-Laki
Blood Type : A
Address : 54983 Jones Ways, Morissettechester, Chile
Religion : Konghucu
Marriage : Menikah
Job : Dynamic Group Executive
Nationality : New Caledonia

e. **Uji 5**

i.

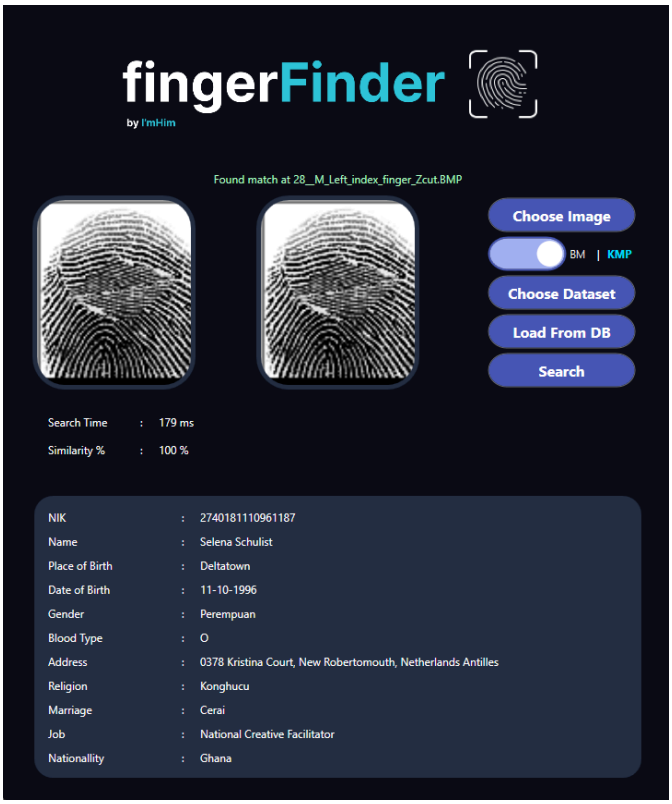

fingerFinder
by I'mHim
Found match at 9__M_Left_ring_finger_Zcut.BMP
Choose Image
BM | KMP
Choose Dataset
Load From DB
Search
Search Time : 12522 ms
Similarity % : 58,333336 %
NIK : 1343250402906172
Name : Casey McKenzie
Place of Birth : South Marilyne
Date of Birth : 04-02-1990
Gender : Laki-Laki
Blood Type : A
Address : 47687 Warren Pines, Larsonview, Iraq
Religion : Buddha
Marriage : Belum Menikah
Job : Lead Accountability Liaison
Nationality : Netherlands Antilles

ii.

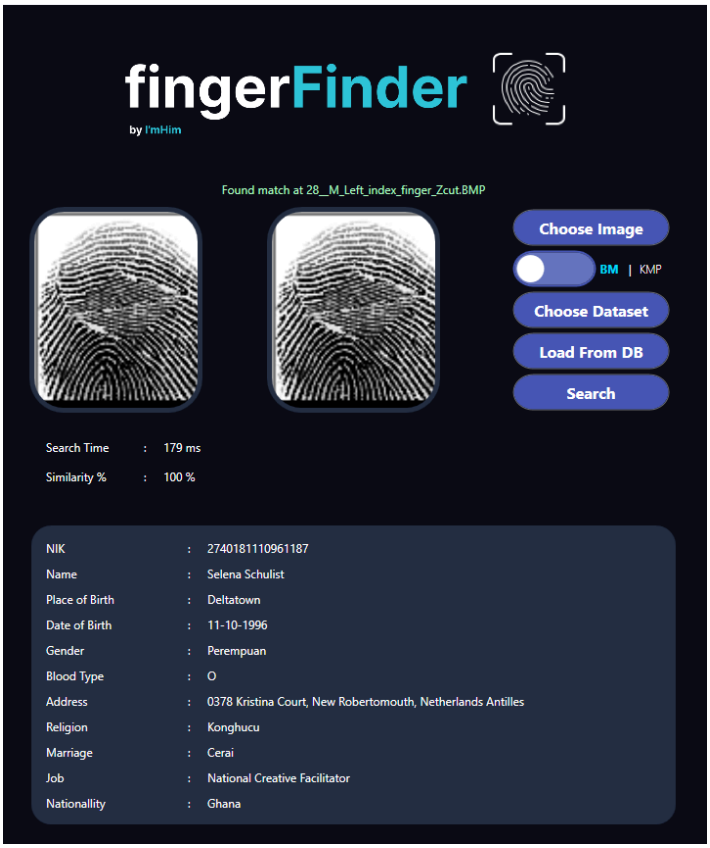

fingerFinder
by I'mHim
Found match at 9__M_Left_ring_finger_Zcut.BMP
Choose Image
BM | KMP
Choose Dataset
Load From DB
Search
Search Time : 12309 ms
Similarity % : 58,333336 %
NIK : 1343250402906172
Name : Casey McKenzie
Place of Birth : South Marilyne
Date of Birth : 04-02-1990
Gender : Laki-Laki
Blood Type : A
Address : 47687 Warren Pines, Larsonview, Iraq
Religion : Buddha
Marriage : Belum Menikah
Job : Lead Accountability Liaison
Nationallity : Netherlands Antilles

f. Uji 5

i.


fingerFinder
by I'mHim
Found match at 28__M_Left_index_finger_Zcut.BMP
Choose Image
BM | KMP
Choose Dataset
Load From DB
Search
Search Time : 179 ms
Similarity % : 100 %
NIK : 2740181110961187
Name : Selena Schulist
Place of Birth : Deltatown
Date of Birth : 11-10-1996
Gender : Perempuan
Blood Type : O
Address : 0378 Kristina Court, New Robertomouth, Netherlands Antilles
Religion : Konghucu
Marriage : Cerai
Job : National Creative Facilitator
Nationallity : Ghana

ii.



4.4. Analisis Hasil Pengujian

Hasil pengujian memberikan beberapa informasi penting mengenai algoritma yang digunakan dalam pencarian sidik jari. Pengujian dilakukan berdasarkan beberapa aspek, termasuk akurasi dan waktu pencarian.

Akurasi kedua algoritma, KMP dan BM, dapat dikatakan mirip atau bahkan sama. KMP dan BM pada dasarnya mencari kecocokan antara pola dan string. Jika ditemukan pola yang sama dalam string tersebut, kedua algoritma akan mengembalikan nilai true, dan begitu juga sebaliknya. Dengan demikian, dapat disimpulkan bahwa kedua algoritma ini memiliki tingkat akurasi yang setara.

Berdasarkan hasil pengujian, waktu yang dibutuhkan untuk mencari pola dalam sebuah string oleh kedua algoritma ini relatif sama. Secara teori, performa KMP akan menurun jika ukuran alfabet semakin besar, sementara performa BM akan menurun jika ukuran alfabet semakin kecil. Dalam implementasi program ini, kami menggunakan karakter 256-bit untuk pengecekan, yang berarti ada 256 kemungkinan karakter yang harus

diperiksa. Untuk ukuran karakter sebanyak ini, performa kedua algoritma relatif sama, seperti yang ditunjukkan oleh hasil pengujian yang menunjukkan waktu pencarian yang hampir identik.

Jika kedua algoritma sebelumnya tidak menemukan pola yang cocok dalam citra yang dicari, pencarian akan dilanjutkan dengan algoritma Levenshtein Distance. Akurasi dari Levenshtein Distance dikategorikan baik, di mana hasil persentase kemiripan sesuai dengan kemiripan kedua citra. Dalam implementasi ini, pengecekan menggunakan algoritma Levenshtein Distance dilakukan secara bersamaan dengan pengecekan menggunakan algoritma KMP atau BM, sehingga hasil dari Levenshtein Distance akan muncul setelah pencarian seluruh database selesai. Hal ini mengakibatkan pencarian dengan menggunakan Levenshtein Distance memerlukan waktu paling lama di antara ketiga algoritma tersebut, walaupun waktu pencarian pada satu citra mungkin sama.

Bab 5
Kesimpulan, Saran, dan Refleksi

5.1. Kesimpulan

Dalam tugas besar ini, kami menerapkan konsep algoritma pencocokan pola yang dipelajari dari mata kuliah IF2211 Strategi Algoritma untuk menentukan kecocokan sidik jari. Proyek ini memberikan pemahaman yang lebih mendalam tentang algoritma string matching, seperti KMP dan Boyer-Moore, serta efektivitas mereka dalam mencocokkan pola secara efisien. Selain itu, kami mempelajari implementasi algoritma *Levenshtein Distance* untuk mengukur kemiripan sidik jari. Kami juga mengeksplorasi pembuatan aplikasi GUI menggunakan .NET dan C#. Pengembangan aplikasi ini memanfaatkan pemahaman kami tentang Pemrograman Berbasis Objek (OOP) yang telah diajarkan di mata kuliah terkait.

5.2. Saran

Proses pencarian sidik jari dapat ditingkatkan kecepatannya dengan menggunakan multithreading, yang memungkinkan pencarian dilakukan secara paralel sehingga mempercepat keseluruhan proses pencocokan sidik jari dengan database.

Lampiran

- Tautan Repository GitHub: https://github.com/shafiqIrv/Tubes3_ImHim
- Tautan Figma: https://www.figma.com/design/yTZpzDxZhho0kULUR4bOxT/Untitled?node-id=0%3A1&t=O5Xi5Ai6UbuGSpS7-1

Daftar Pustaka

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

https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2005-2006/Makalah2006/MakalahStmik2006-30.pdf

https://github.com/bchavez/Bogus