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

Laporan Tugas Besar

Dibuat Untuk Memenuhi Tugas Besar Mata Kuliah Strategi Algoritma
Tahun Akademik 2023/2024

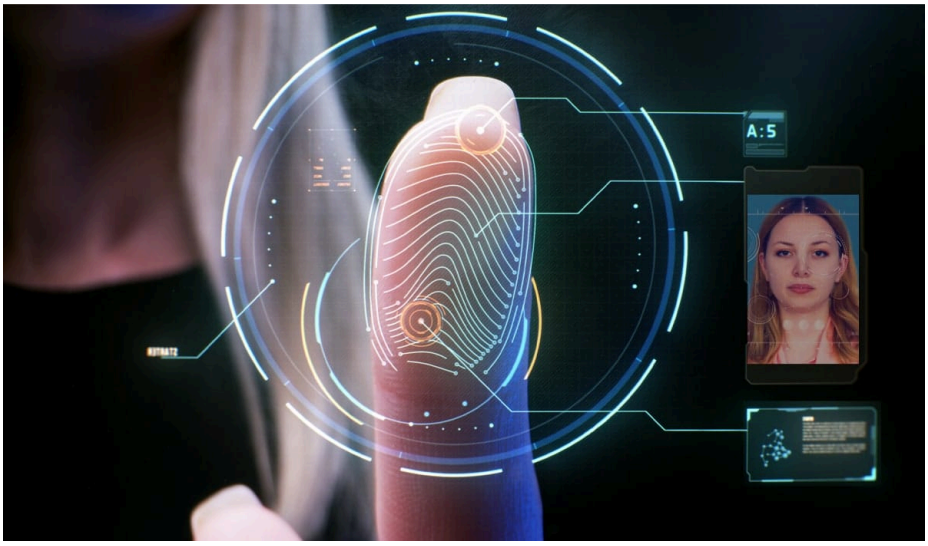


Oleh

Jimly Nur Arif 13522123
Auralea Alvinia Syaikha 13522148
Muhammad Roihan 13522152

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG BANDUNG
2024

BAB 1
DESKRIPSI TUGAS



Gambar 1. Ilustrasi fingerprint recognition pada deteksi berbasis biometrik.
Sumber: https://www.aratek.co/news/unlocking-the-secrets-of-fingerprint-recognition

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

Pattern matching merupakan teknik penting dalam sistem identifikasi sidik jari. Teknik ini digunakan untuk mencocokkan pola sidik jari yang ditangkap dengan pola sidik jari yang terdaftar di database. Algoritma pattern matching yang umum digunakan adalah Bozorth dan Boyer-Moore. Algoritma ini memungkinkan sistem untuk mengenali sidik jari dengan cepat dan akurat, bahkan jika sidik jari yang ditangkap tidak sempurna.

Dengan menggabungkan teknologi identifikasi sidik jari dan pattern matching, dimungkinkan untuk membangun sistem identifikasi biometrik yang aman, handal, dan mudah digunakan. Sistem ini dapat diaplikasikan di berbagai bidang, seperti kontrol akses, absensi karyawan, dan verifikasi identitas dalam transaksi keuangan. Di dalam Tugas Besar 3 ini, Kami 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 Deskripsi Algoritma

2.1.1 KMP

Algoritma Knuth-Morris-Pratt (KMP) adalah sebuah algoritma pencocokan string yang efisien digunakan untuk mencari kecocokan pola dalam sebuah teks. Algoritma ini mencari pola dari kiri ke kanan mirip seperti brute-force. Namun, algoritma ini melakukan *shift* pola lebih efisien daripada brute force. Sebagai contoh misalkan kita mempunyai text T dan pattern P. Jika terdapat *mismatch* dimana T[i] != P[j], maka algoritma akan melakukan *shift* paling banyak untuk menghindari *wasteful comparisons*. Menurut algoritma ini *shift* paling banyak adalah prefix terbesar dari P[0..j-1] yang mana juga merupakan suffix dari P[1..j-1]. Untuk mendapatkan prefix terbesar dari P[0..j-1] yang mana juga merupakan suffix dari P[1..j-1] diperlukan perhitungan menggunakan fungsi pinggiran. Fungsi pinggiran b(k) didefinisikan sebagai ukuran prefix terbesar P[0..j-1] yang mana juga merupakan suffix dari P[1..j-1].

Algoritma ini memiliki kompleksitas waktu O(m+n), dengan m adalah panjang pola dan n adalah panjang text. Perhitungan fungsi pinggiran memiliki kompleksitas O(m) dan kompleksitas pencarian string memiliki kompleksitas O(n). Keuntungan penggunaan algoritma ini tidak perlu mundur dalam melakukan pencocokan dengan text T. Hal ini menyebabkan algoritma ini bagus untuk memproses file besar yang dibaca melalui perangkat eksternal atau melalui *network stream*. Namun algoritma ini tidak berfungsi baik jika ukuran alfabet meningkat.

2.1.2 BM

Algoritma Boyer-Moore (BM) adalah sebuah algoritma pencocokan string yang efisien digunakan untuk mencari kecocokan pola dalam sebuah teks. Algoritma ini melakukan pencocokan string berdasarkan dua teknik yaitu *the looking-glass technique* dan *the character-jump technique*. Terdapat tiga kasus yang mungkin terjadi. Pertama, jika T[i] != P[j] dan P mengandung T[i] maka coba geser P ke kanan sejajarkan kemunculan terakhir T[i] di P dengan T[i]. Kedua, jika T[i] != P[j] dan P mengandung T[i] tetapi geser ke kanan tidak memungkinkan, maka geser P ke kanan sejauh satu karakter. Ketiga, jika T[i] != P[j] dan P tidak mengandung T[i], maka geser P sehingga P[0] sejajar dengan T[i+1]. Algoritma ini perlu melakukan pencarian *last occurrence* untuk setiap alfabet. L(x) atau *last occurrence x* adalah index (i) terbesar dimana P[i] == x atau -1 jika x tidak ditemukan. Untuk *worst case* algoritma ini memiliki kompleksitas waktu O(nm + A). Algoritma ini cepat alfabet (A) besar dan lambat bila alfabet (A) kecil.

2.1.3 Regex

Regex (Regular Expressions) adalah alat yang sangat efektif untuk mencocokkan pola dalam teks. Digunakan secara luas dalam berbagai bahasa pemrograman dan alat pengolah teks, regex memungkinkan pencarian, pengeditan, dan validasi string berdasarkan pola yang ditentukan. Regex bekerja dengan mendefinisikan pola yang terdiri dari karakter biasa (seperti huruf dan angka) dan karakter khusus (seperti titik, tanda bintang, kurung siku), yang mendefinisikan aturan pencocokan.

Saat menjalankan regex, mesin regex beroperasi secara iteratif, dimulai dari awal teks dan mencoba mencocokkan setiap bagian teks dengan pola yang diberikan. Karakter khusus dalam regex memiliki makna spesifik: titik (.) mencocokkan sembarang karakter tunggal kecuali baris baru, tanda bintang (*) mencocokkan nol atau lebih dari elemen sebelumnya, tanda tanya (?) mencocokkan nol atau satu dari elemen sebelumnya, kurung siku ([]) mencocokkan salah satu karakter di dalamnya, tanda (^) menandai awal dari sebuah string, dan tanda dolar ($) menandai akhir dari sebuah string. Untuk mencocokkan karakter literal yang memiliki makna khusus dalam regex, digunakan tanda garis miring terbalik (\).

Proses regex terdiri dari kompilasi pola menjadi bentuk internal yang dioptimalkan, kemudian pencocokan teks dengan pola yang dikompilasi. Mesin regex mulai dari awal teks dan berusaha mencocokkan teks dengan pola tersebut. Jika terjadi ketidakcocokan, mesin akan kembali (backtrack) dan mencoba alternatif lain untuk mencocokkan pola jika ada.

2.2 Teknik Pengukuran Persentase Kemiripan

Untuk metode pengukuran kemiripan kami menggunakan algoritma hamming distance. Alasan pemilihan algoritma hamming distance adalah karena menurut kami algoritma ini yang paling mudah untuk diimplementasikan dibanding dengan Levenshtein Distance, ataupun Longest Common Subsequence (LCS). Sederhananya hamming distance menghitung jumlah karakter berbeda antara dua buah string. Batas bawah persentase kemiripan yang kami tetapkan adalah 0.75 atau 75%. Hal ini didasari karena panjang pattern yang kami ambil adalah 32 pixel yang jika diubah menjadi ASCII 8 bit hanya akan menghasilkan ASCII sepanjang 4 karakter. Hal ini menyebabkan hanya terdapat 5 buah kemungkinan persentase kemiripan yaitu 0%, 25%, 50%, 75%, dan 100%.

2.3 Aplikasi Desktop

Windows Presentation Foundation (WPF) adalah sebuah framework yang dikembangkan oleh Microsoft untuk membuat aplikasi desktop berbasis Windows menggunakan bahasa pemrograman C# dan .NET. Salah satu keunggulan utama WPF

adalah pemisahan antara logika bisnis dan antarmuka pengguna, yang memudahkan pengelolaan dan pemeliharaan kode. Antarmuka pengguna didefinisikan secara deklaratif menggunakan XAML (eXtensible Application Markup Language), yang memungkinkan pengembang untuk mendesain UI dengan lebih intuitif dan fleksibel. Fitur data binding yang kuat dalam WPF memungkinkan sinkronisasi otomatis antara elemen UI dan data, sementara kemampuan templating dan styling memberikan kontrol penuh atas tampilan dan perilaku elemen UI. Selain itu, WPF mendukung grafis yang lebih canggih melalui integrasi dengan DirectX, memungkinkan pembuatan aplikasi dengan tampilan grafis 2D dan 3D yang menarik. Untuk memulai pengembangan aplikasi WPF, pengembang dapat menggunakan Visual Studio, yang menyediakan alat dan lingkungan yang diperlukan untuk membuat proyek WPF. Dalam proyek WPF, antarmuka pengguna dirancang menggunakan XAML, sementara logika dan interaksi pengguna ditangani dalam file code-behind menggunakan C#. Contoh sederhana aplikasi WPF melibatkan pembuatan antarmuka dengan tombol melalui XAML, dan penanganan klik tombol dengan logika C# yang menampilkan pesan. Dengan fitur-fitur ini, WPF memungkinkan pembuatan aplikasi desktop yang modern, interaktif, dan kaya fitur.

BAB 3
ANALISIS PEMECAHAN MASALAH

3.1 Langkah - Langkah Pemecahan Masalah

3.1.1 Program

Berikut langkah - langkah pemecahan masalah menggunakan program yang kami buat:

1. Ambil sebuah input dari pengguna berupa gambar sidik jari.
2. Ambil 32 pixel paling tengah dari input.
3. Konversi pixel tersebut menjadi binary.
4. Konversi biner menjadi ASCII 8 bit, lalu simpan hasil dengan menggunakan variabel bertipe data string. ASCII ini nanti digunakan sebagai pola.
5. Untuk setiap data di database, konversi ke ASCII 8 bit. Data ini akan digunakan sebagai teks.
6. Lakukan pencocokan pola pada teks dengan menggunakan algoritma KMP / BM.
7. Jika pola ditemukan maka tingkat kemiripan 100%, jika tidak akan dilakukan perhitungan tingkat kemiripan menggunakan algoritma hamming distance.
8. Jika perhitungan menggunakan hamming distance < 75 %, Program akan menampilkan pesan bahwa sidik jari tidak ditemukan.
9. Jika sidik jari ditemukan lakukan pencocokan nama pada sidik jari yang ditemukan dengan nama pada tabel biodata menggunakan regex.
10. Jika nama pada sidik jari dan nama pada tabel biodata ditemukan tampilkan biodata.

3.1.2 Algoritma KMP

Berikut langkah - langkah pemecahan masalah menggunakan algoritma KMP:

1. Algoritma menerima sebuah text bertipe data string dan pattern bertipe data string.
2. Lakukan perhitungan fungsi pinggiran. Fungsi pinggiran b(k) didefinisikan sebagai ukuran prefix terbesar P[0..j-1] yang mana juga merupakan suffix dari P[1..j-1].
3. Algoritma melakukan pencocokan mulai dari kiri ke kanan teks.
4. Jika terdapat *mismatch* dimana T[i] != P[j], maka algoritma akan melakukan *shift* sesuai dengan fungsi pinggiran yang telah dibuat.
5. Jika pola ditemukan maka tingkat kemiripan 100%, jika tidak akan dilakukan perhitungan tingkat kemiripan menggunakan algoritma hamming distance.

3.1.3 Algoritma BM

Berikut langkah - langkah pemecahan masalah menggunakan algoritma BM:

1. Algoritma menerima sebuah text bertipe data string dan pattern bertipe data string.
2. Lakukan perhitungan *last occurrence*. L(x) atau *last occurrence x* adalah index (i) terbesar dimana P[i] == x atau -1 jika x tidak ditemukan.
3. Algoritma melakukan pencocokan dimulai dari akhir pola dan dibandingkan dengan teks dari kiri ke kanan.
4. jika T[i] != P[j] dan P mengandung T[i] maka coba geser P ke kanan sesuai tabel *last occurrence*.
5. T[i] != P[j] dan P tidak mengandung T[i], maka geser P sehingga P[0] sejajar dengan T[i+1].
6. Jika pola ditemukan maka tingkat kemiripan 100%, jika tidak akan dilakukan perhitungan tingkat kemiripan menggunakan algoritma hamming distance.

3.2 Fitur Fungsional Dan Arsitektur Aplikasi Desktop

Berikut merupakan fitur fungsional aplikasi desktop yang dibuat :

1. Antarmuka untuk melakukan input gambar dari pengguna
2. Konversi gambar ke binary
3. Konversi binary ke ASCII 8 bit
4. Pencarian pola menggunakan KMP
5. Pencarian pola menggunakan BM
6. Perhitungan tingkat kemiripan
7. Penanganan data korup
8. Tampilan hasil pencarian

Arsitektur Aplikasi Desktop:

1. Dibangun menggunakan bahasa C#
2. Menggunakan SQL untuk penyimpanan sidik jari dan biodata

3.3 Contoh Ilustrasi Kasus

3.3.1 Ilustrasi Algoritma KMP

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

a	b	a	c	c	a	b	a	c	a	b	a	a	b	b
a	b	a	c	a	b									
				a	b	a	c	a	b					
					a	b	a	c	a	b				

3.3.2 Ilustrasi Algoritma BM

A = {a, b, c, d}
P : “abacab”

x	a	b	c	d
L(x)	4	5	3	-1

a	b	a	c	a	a	b	a	d	c	a	b	a	c	a	b	a	a	b	b
a	b	a	c	a	b														
	a	b	a	c	a	b													
		a	b	a	c	a	b												
			a	b	a	c	a	b											
									a	b	a	c	a	b					
										a	b	a	c	a	b				

BAB 4
IMPLEMENTASI DAN PENGUJIAN

4.1 Implementasi Algoritma

4.1.1 BM

```
public class BM
{
    private double maxval = 100;
    private int[] buildLast(string pattern){

        int[] last = new int[256];

        for (int i = 0; i < 256; i++)
        {
            last[i] = -1; // initialize array
        }

        for (int i = 0; i < pattern.Length; i++)
        {
            int idx = pattern[i];

            last[idx] = i;
        }
        return last;
    }

    private void HammingDistance(string pattern, string text, int
startIndex)
    {
        int distance = 0;
        int patternLength = pattern.Length;
        int textLength = text.Length;

        for (int i = 0; i < patternLength; i++)
        {
            int textIndex = startIndex + i;
            if (textIndex >= textLength)
            {
                return;
            }
```

```
            if (pattern[i] != text[textIndex])
            {
                distance++;
            }
        }

        if(distance<maxval){
            maxval = distance;
        }

    }
    public double solveBM(string pattern,string text){
        int m = pattern.Length;
        int n = text.Length;

        int[] last = buildLast(pattern);

        int i = m - 1;
        if (i > n - 1){
            throw new ArgumentException($"gak bisa");
        }

        int j = m - 1;

        do
        {
            HammingDistance(pattern, text, i - (m - 1));

            if (pattern[j] == text[i])
            {
                if (j == 0)
                {
                    break;
                }
                else
                { // looking-glass technique
                    i--;
                    j--;
                }
            }
```

```
            else
            { // character jump technique
                int lo = last[text[i]]; // last occ
                i = i + m - Math.Min(j, 1 + lo);
                j = m - 1;
            }
        } while (i <= n - 1);

        double kemiripan = 1 - (maxval / m);
        return kemiripan;
        // Console.WriteLine($"Tingkat kemiripan adalah: {kemiripan}");

    }
}
```

maxval digunakan untuk melacak nilai minimum dari Hamming Distance yang ditemukan selama pencocokan pola dalam teks. Nilai awalnya adalah 100, yang kemudian akan diperbarui dengan nilai yang lebih kecil jika ditemukan jarak Hamming yang lebih kecil

BuildLPS ini membangun tabel LPS (Longest Prefix Suffix) untuk pola yang diberikan. Tabel LPS digunakan oleh algoritma KMP untuk menghindari pemeriksaan ulang karakter dalam pola.

Parameter: pattern adalah pola yang akan dicari dalam teks.
Output: Menghasilkan array lps yang menyimpan panjang dari awalan terpanjang yang juga merupakan sufiks untuk setiap posisi dalam pola.

4.1.2 KMP

```
    public class KMP
    {
        private double maxval = 100;

        private int[] BuildLPS(string pattern)
        {
            int m = pattern.Length;
            int[] lps = new int[m];
            int length = 0;
```

```
            int i = 1;

            lps[0] = 0; // lps[0] is always 0

            while (i < m)
            {
                if (pattern[i] == pattern[length])
                {
                    length++;
                    lps[i] = length;
                    i++;
                }
                else
                {
                    if (length != 0)
                    {
                        length = lps[length - 1];
                    }
                    else
                    {
                        lps[i] = 0;
                        i++;
                    }
                }
            }
            return lps;
        }

        private void HammingDistance(string pattern, string text, int
startIndex)
        {
            int distance = 0;
            int patternLength = pattern.Length;
            int textLength = text.Length;

            for (int i = 0; i < patternLength; i++)
            {
                int textIndex = startIndex + i;
                if (textIndex >= textLength)
                {
```

```
                return;
            }

            if (pattern[i] != text[textIndex])
            {
                distance++;
            }
        }

        if (distance < maxval)
        {
            maxval = distance;
        }
    }

    public double solveKMP(string pattern, string text)
    {
        int m = pattern.Length;
        int n = text.Length;

        int[] lps = BuildLPS(pattern);

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

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

            if (j == m)
            {
                HammingDistance(pattern, text, i - j);
                j = lps[j - 1];
            }
            else if (i < n && pattern[j] != text[i])
            {
```

```
                    if (j != 0)
                    {
                        j = lps[j - 1];
                    }
                    else
                    {
                        i++;
                    }
                }
            }

            double kemiripan = 1 - (maxval / m);
            return kemiripan;
        }
    }
```

maxval digunakan untuk melacak nilai minimum dari Hamming Distance yang ditemukan selama pencocokan pola dalam teks. Nilai awalnya adalah 100, yang kemudian akan diperbarui dengan nilai yang lebih kecil jika ditemukan jarak Hamming yang lebih kecil.

BuildLPS adalah tabel last untuk pola yang diberikan. Tabel last menyimpan indeks terakhir dari setiap karakter dalam pola. Tabel ini digunakan oleh algoritma Boyer-Moore untuk menentukan jumlah lompatan ketika terjadi ketidakcocokan.

Algoritma Boyer-Moore yang diimplementasikan dalam kode ini berfungsi untuk mencari pola dalam teks dan menghitung tingkat kemiripan berdasarkan jarak Hamming. Algoritma dimulai dengan membangun tabel last yang menyimpan indeks terakhir dari setiap karakter dalam pola, yang kemudian digunakan untuk menentukan lompatan saat terjadi ketidakcocokan. Proses pencocokan dilakukan dari belakang ke depan dengan menggunakan dua teknik utama: looking-glass technique untuk mencocokkan karakter dari belakang ke depan dan character jump technique untuk melompat ke depan dalam teks ketika ada ketidakcocokan berdasarkan tabel last. Saat pola ditemukan dalam teks, jarak Hamming antara pola dan segmen teks dihitung, dan maxval diperbarui jika jarak yang ditemukan lebih kecil. Algoritma terus mencari hingga seluruh teks telah diperiksa atau pola ditemukan. Akhirnya, tingkat kemiripan dihitung sebagai 1 - (maxval / m), di mana m adalah panjang pola, dan nilai ini dikembalikan sebagai hasil. Algoritma ini menggabungkan efisiensi lompatan karakter Boyer-Moore dengan penghitungan jarak Hamming untuk menentukan seberapa mirip pola dengan segmen teks yang ditemukan.

4.1.3 Regex

```
using System.Text.RegularExpressions;
using System.Reflection;
public class RegexFunction{
    public string ConvertAlayToNormal(string input)
    {
        input = input.ToLower();
        var patterns = new Dictionary<string, string>
        {
            { "13", "b" },
            { "4", "a" },
            { "3", "e" },
            { "1", "i" },
            { "0", "o" },
            { "5", "s" },
            { "7", "t" },
            { "6", "g" },
            { "2", "z" },
        };
        foreach (var pattern in patterns)
        {
            input = Regex.Replace(input, pattern.Key, pattern.Value);
        }
        return input;
    }


    public bool CheckRegex(string input, string check)
    {
        // input = input.ToLower();
        int inputLength = input.Length;

        string pattern = @"^(";

        for (int i = 0; i < inputLength; i++)
        {
            if(input[i].ToString() == " "){
                pattern += "[";
                pattern += "\\s";
```

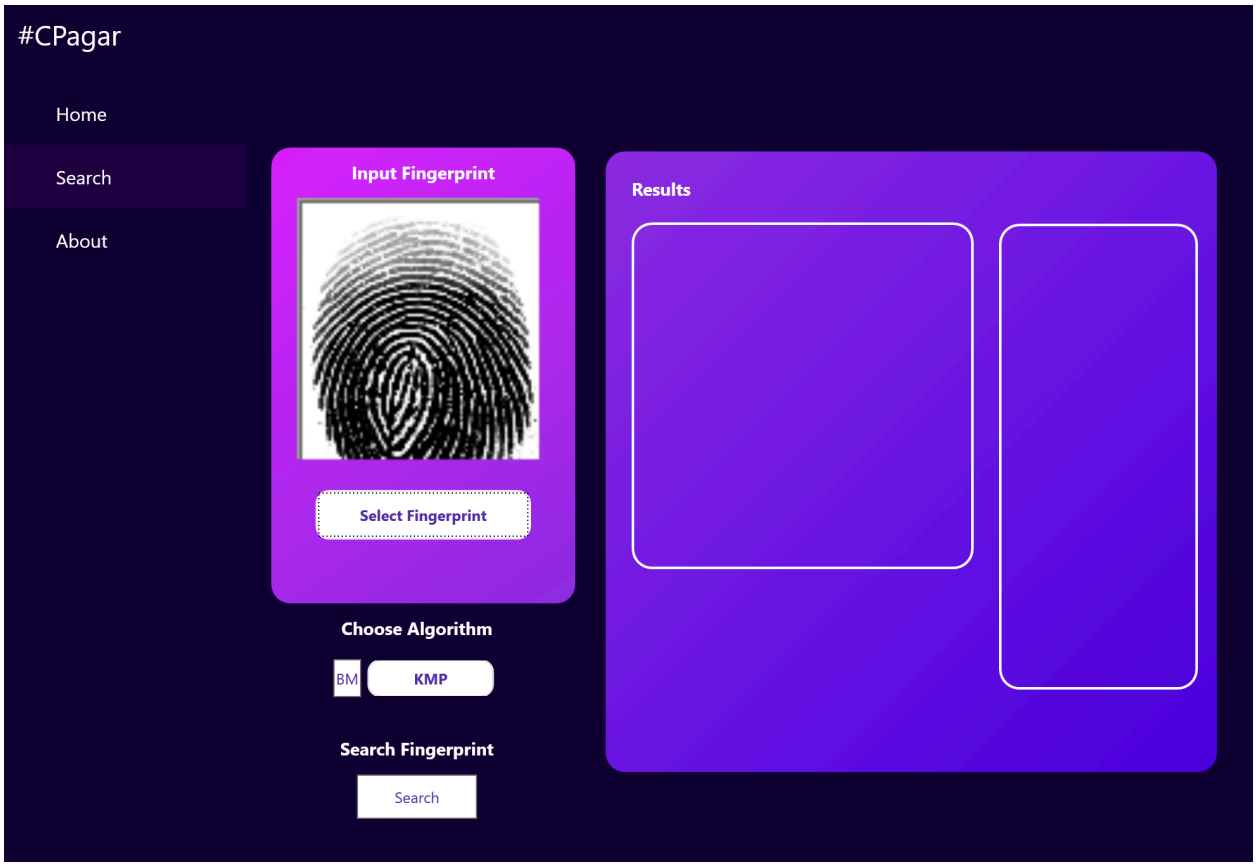
```
                pattern += "]*";
            }
            else{
                pattern += "[";
                pattern += input[i].ToString();
                pattern += "]?";
            }
        }
        pattern += ")$";

        // Console.WriteLine(pattern);

        bool match = Regex.IsMatch(check, pattern);

        if (match)
        {
            Console.WriteLine("String sesuai dengan pola.");
            return true;
        }
        Console.WriteLine("String tidak sesuai dengan pola.");
        return false;
    }

    public List<String> FindBiodata(string nameAns){
            DB db = new DB();
            List<string> namaList = db.GetAllBiodataNama();
            List<string> normalNamaList = new List<string>();
            foreach(var nama in namaList){
                normalNamaList.Add(ConvertAlayToNormal(nama));
            }
            bool cek = false;
            List<string> biodata = new List<string>();
            for(int i=0;i<normalNamaList.Count;i++){
                Console.WriteLine($"normal nama: {normalNamaList[i]}");
                cek = CheckRegex(nameAns.ToLower(), normalNamaList[i]);
                if(cek){
                    biodata = db.GetBiodataByNama(namaList[i]);
                    if (biodata.Count > 0)
                    {
                        Console.WriteLine("Biodata ditemukan:");
```

```
                        Console.WriteLine($"NIK: {biodata[0]}");
                        Console.WriteLine($"Nama: {biodata[1]}");
                        Console.WriteLine($"Tempat Lahir: {biodata[2]}");
                        Console.WriteLine($"Tanggal Lahir: {biodata[3]}");
                        Console.WriteLine($"Jenis Kelamin: {biodata[4]}");
                        Console.WriteLine($"Golongan Darah:
{biodata[5]}");
                        Console.WriteLine($"Alamat: {biodata[6]}");
                        Console.WriteLine($"Agama: {biodata[7]}");
                        Console.WriteLine($"Status Perkawinan:
{biodata[8]}");
                        Console.WriteLine($"Pekerjaan: {biodata[9]}");
                        Console.WriteLine($"Kewarganegaraan:
{biodata[10]}");
                    }
                    else
                    {
                        Console.WriteLine("Biodata tidak ditemukan.");
                    }
                    break;
                }
            }
            if(!cek){
                Console.WriteLine("Biodata tidak ditemukan.");
            }
        return biodata;
    }
}
```

Kelas RegexFunction memiliki tiga metode utama: ConvertAlayToNormal, CheckRegex, dan FindBiodata. Metode ConvertAlayToNormal berfungsi untuk mengubah teks dalam gaya "alay" (menggunakan angka sebagai pengganti huruf) menjadi teks normal dengan huruf yang sesuai, seperti mengubah "13" menjadi "b" dan "4" menjadi "a". Metode CheckRegex membangun pola regex dinamis berdasarkan input yang diberikan dan memeriksa apakah string kedua sesuai dengan pola yang dibuat, memungkinkan sedikit variasi karakter dan mengabaikan spasi tambahan. Metode FindBiodata mengintegrasikan dua metode sebelumnya untuk mencari biodata seseorang dari database berdasarkan nama yang diubah dari gaya "alay" ke normal. Ia mengambil daftar nama dari database, mengonversinya ke format normal, dan kemudian menggunakan regex untuk mencocokkan nama yang diminta. Jika ditemukan

kecocokan, biodata yang sesuai diambil dan ditampilkan; jika tidak, pesan bahwa biodata tidak ditemukan ditampilkan. Metode ini memanfaatkan kekuatan regex untuk fleksibilitas pencarian dan penyesuaian karakter dalam nama.

4.1.4 Pengujian



```
File Name: 3.BMP
Binary Data: 10010000100001110000110000000000
ASCII Data: ??

pattern: ??

File Name: 1.BMP
File Name: 3.BMP
File Name: 5.BMP
File Name: 7.BMP
yang paling mirip adalah Jane Smith dengan Tingkat kemiripan  100%
path = ./sidiks/3.BMP
normal nama: jiml n a
String tidak sesuai dengan pola.
normal nama: jane s
String sesuai dengan pola.
nama: Jane S
imgPath hasil: ./sidiks/3.BMP
Biodata ditemukan:
NIK: 2345678901234567
Nama: Jane S
Tempat Lahir: Bandung
Tanggal Lahir: 02/02/1992 00:00:00
Jenis Kelamin: Perempuan
Golongan Darah: A
Alamat: Jl. Sudirman No. 2
Agama: Kristen
Status Perkawinan: Menikah
Pekerjaan: Desainer
Kewarganegaraan: Indonesia
```

```
SQL dump executed successfully.
File Name: 10.BMP
Binary Data: 10110001100011100000111100100011
ASCII Data: ±?#
pattern: ±?#
File Name: 1.BMP
File Name: 3.BMP
File Name: 5.BMP
File Name: 7.BMP
yang paling mirip adalah Jimly Nur Arif dengan Tingkat kemiripan  25%
path = ./sidiks/1.BMP
Sidik Jari Tidak Ditemukan
```

```
SQL dump executed successfully.
File Name: 5.BMP
Binary Data: 10111000111100111000011000001110
ASCII Data: ,ó?
pattern: ,ó?
File Name: 1.BMP
File Name: 3.BMP
File Name: 5.BMP
File Name: 7.BMP
yang paling mirip adalah Alice Johnson dengan Tingkat kemiripan  100%
path = ./sidiks/5.BMP
normal nama: jiml n a
String tidak sesuai dengan pola.
normal nama: jane s
String tidak sesuai dengan pola.
normal nama: alice joson
String sesuai dengan pola.
nama: Alic3 JoSon
imgPath hasil: ./sidiks/5.BMP
Biodata ditemukan:
NIK: 3456789012345678
Nama: Alic3 JoSon
Tempat Lahir: Surabaya
Tanggal Lahir: 03/03/1988 00:00:00
Jenis Kelamin: Perempuan
Golongan Darah: B
Alamat: Jl. Thamrin No. 3
Agama: Hindu
Status Perkawinan: Cerai
Pekerjaan: Pengacara
Kewarganegaraan: Indonesia
```

```
SQL dump executed successfully.
File Name: 1.BMP
Binary Data: 10111110001000100110010001000110
ASCII Data: _"dF
pattern: _"dF
File Name: 1.BMP
File Name: 3.BMP
File Name: 5.BMP
File Name: 7.BMP
yang paling mirip adalah Jimly Nur Arif dengan Tingkat kemiripan  100%
path = ./sidiks/1.BMP
normal nama: jiml n a
String sesuai dengan pola.
nama: J1ml n A
imgPath hasil: ./sidiks/1.BMP
Biodata ditemukan:
NIK: 1234567890123456
Nama: J1ml n A
Tempat Lahir: Jakarta
Tanggal Lahir: 01/01/1990 00:00:00
Jenis Kelamin: Laki-Laki
Golongan Darah: O
Alamat: Jl. Merdeka No. 1
Agama: Islam
Status Perkawinan: Belum Menikah
Pekerjaan: Programmer
Kewarganegaraan: Indonesia
```

BAB 5
KESIMPULAN, SARAN, TANGGAPAN, DAN REFLEKSI

5.1 Kesimpulan

Penggunaan algoritma Knuth-Morris-Pratt (KMP) dan Boyer-Moore (BM) dalam program pencocokan sidik jari menawarkan pendekatan yang efisien untuk mencari pola dalam data sidik jari. KMP unggul dalam efisiensi waktu dengan memastikan bahwa setiap karakter dalam teks sumber hanya diperiksa satu kali, sementara BM memanfaatkan heuristik yang mengurangi jumlah perbandingan yang diperlukan dengan melompati bagian teks yang tidak perlu. Kedua algoritma ini memiliki kelebihan masing-masing yang bisa dimanfaatkan tergantung pada karakteristik data sidik jari dan kebutuhan spesifik dari aplikasi yang dikembangkan.

5.2 Saran

Pemilihan Algoritma Berdasarkan Kebutuhan:

- Jika pola sidik jari yang dicari relatif pendek atau jika performa terbaik dibutuhkan dalam kasus terburuk, KMP bisa menjadi pilihan yang lebih baik karena konsistensinya dalam waktu pencarian.
- Jika data sidik jari memiliki banyak pengulangan karakter yang berbeda, atau jika pencocokan rata-rata lebih penting daripada pencocokan kasus terburuk, algoritma BM dapat memberikan kinerja yang lebih cepat.

Preprocessing Data:

- Lakukan preprocessing pada data sidik jari untuk mengurangi noise dan meningkatkan kualitas data sebelum pencocokan. Teknik seperti normalisasi dan pencocokan minutiae dapat membantu dalam meningkatkan akurasi pencocokan.

Penggunaan Hybrid Algoritma:

- Pertimbangkan untuk mengembangkan pendekatan hybrid yang memanfaatkan keunggulan kedua algoritma. Misalnya, menggunakan KMP untuk tahap awal pencarian dan kemudian menerapkan BM untuk pengoptimalan lebih lanjut pada hasil sementara.

Optimasi Performa:

- Pastikan implementasi algoritma dioptimalkan untuk performa. Hal ini termasuk penggunaan struktur data yang efisien dan pengelolaan memori yang tepat.

- Gunakan paralelisasi jika memungkinkan, terutama ketika memproses dataset sidik jari yang besar.

Pengujian dan Validasi:

- Lakukan pengujian ekstensif menggunakan dataset sidik jari yang representatif untuk memastikan bahwa algoritma berfungsi dengan baik dalam berbagai kondisi.
- Validasi hasil pencocokan dengan metode lain seperti visualisasi grafis atau verifikasi manual untuk memastikan keakuratannya.

Pertimbangan Keamanan:

- Karena data sidik jari merupakan informasi biometrik yang sensitif, pastikan program dilengkapi dengan mekanisme keamanan yang kuat untuk melindungi data dari akses yang tidak sah.

5.3 Tanggapan
Makasih kakak kakak asisten

5.4 Refleksi
GUI harusnya develop bersamaan dengan backend, penggabungan keduanya itu sulit sekali

LAMPIRAN

REPO : https://github.com/auraleaas/Tubes3_CPagar

BONUS VIDEO
https://drive.google.com/drive/folders/1FkGZV68yWahnzc-vfkbOpN8hjZGPsAEa?usp=drive_link

DAFTAR PUSTAKA

informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2023-2024/stima23-24.htm