LAPORAN TUGAS BESAR 3

IF2211 Strategi Algoritma

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

Disusun oleh:

Rafiki Prawhira Harianto	13522065
Muhammad Atpur Rafif	13522089
Indraswara Galih Jayanegara	13522119

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA

INSTITUT TEKNOLOGI BANDUNG

2024

Daftar Isi

BAB I
DESKRIPSI TUGAS

1.1 Deskripsi Tugas

Tugas besar kali ini akan dibuat aplikasi berbasis GUI dengan bahasa C# untuk mencari data dari seseorang menggunakan *Fingerprint*. Mencari kemiripannya dengan menggunakan algoritma String Matching Boyer-Moore dan Knuth-Morris-Pratt. Akan tetapi, pada biodata yang ada di database terjadi *corrupt* pada nama sehingga dibutuhkan regex untuk mencari kemiripan dari sebuah nama yang benar dengan nama yang sudah dimanipulasi.

1.2 Spesifikasi

Pada Tugas Besar ini, dibuat 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.
2. Program dapat memiliki basis data SQL yang telah mencocokkan berkas citra sidik jari yang telah ada dengan seorang pribadi. Basis data yang digunakan berupa 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 display sidik jari yang paling mirip dari basis data, informasi mengenai waktu eksekusi, informasi mengenai tingkat kemiripan sidik jari dengan gambar yang ingin dicari dalam persentase (%), dan list biodata hasil pencarian dari basis data.
5. Pengguna dapat memilih algoritma yang ingin digunakan antara KMP atau BM.
6. Biodata yang ditampilkan harus biodata yang memiliki nama yang benar (menggunakan Regex untuk memperbaiki nama yang rusak dan KMP atau BM untuk mencari orang yang paling sesuai).
7. Program memiliki antarmuka yang user-friendly.

BAB II
LANDASAN TEORI

2.1 Pattern String Matching

Pada permasalahan *String Matching*, diberikan sebuah teks yang relatif panjang dibandingkan dengan pola atau *pattern* yang diberikan. Tujuan dari algoritma adalah mencari indeks pertama kali ditemukannya *pattern* pada teks yang diberikan. Pencarian yang dilakukan berupa *exact match*, atau pola harus sama persis dengan teks pada sebuah indeks.

2.2 Algoritma KMP

Algoritma adalah KMP adalah algoritma String Matching yang membandingkan sebuah pattern dan text. Algoritma ini banyak dipakai di *Text Editor*, *Search Engines*, dan lainnya. Algoritma ini memanfaatkan kemunculan terbanyak dari prefix dan suffix sehingga pada saat ketidakcocokan terjadi akan digeser dengan prefix/suffix yang sesuai.

2.3 Algoritma BM

Algoritma Boyer-Moore adalah algoritma String Matching juga membandingkan sebuah pattern dan text pula, tetapi algoritma ini berbeda dari Knuth-Morris-Pratt. Pada algoritma ini memanfaatkan *Last Occurrence Table*. Tabel ini berisi setiap huruf yang ada di dalam *Pattern* sebanyak satu kali dan kemunculan terakhirnya pada *pattern*.

2.4 Algoritma *Levenshtein Distance*

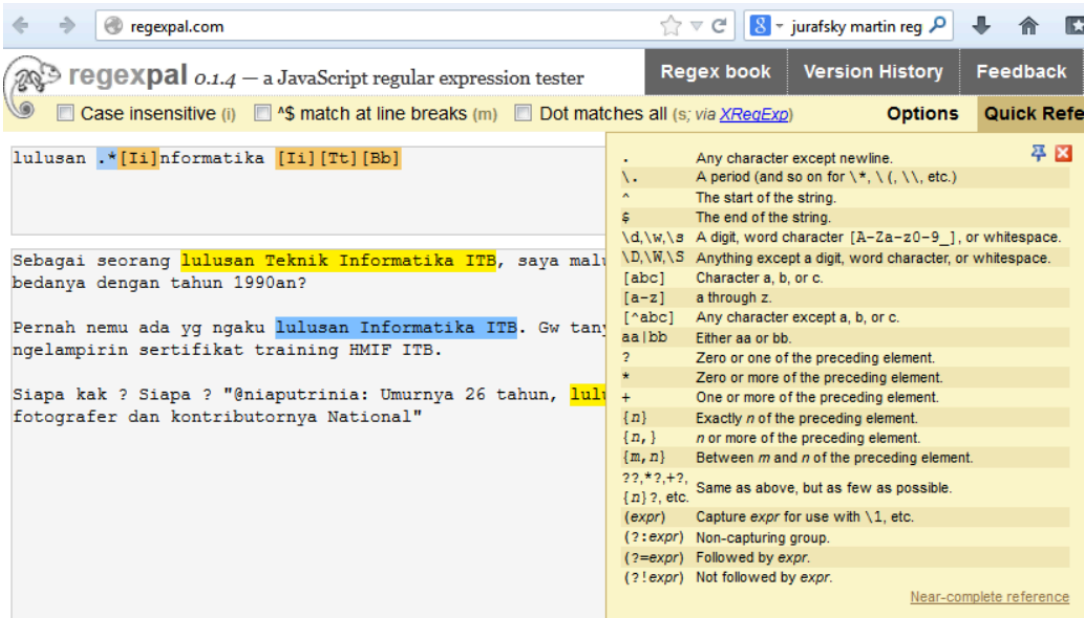
Levenshtein Distance adalah sebuah algoritma yang digunakan untuk mengukur tingkat kesamaan atau kemiripan antara dua buah kata atau *string*. Algoritma ini bekerja dengan menghitung jumlah operasi pengeditan yang diperlukan untuk mengubah satu string menjadi string lainnya. Operasi pengeditan yang dimaksud meliputi penyisipan karakter, penghapusan karakter, dan penggantian karakter. Dengan menggunakan algoritma ini, kita dapat menentukan seberapa mirip atau berbeda dua string berdasarkan jumlah operasi yang diperlukan untuk mentransformasikannya. Algoritma ini sering digunakan dalam berbagai aplikasi, seperti pemrosesan teks, koreksi ejaan, dan pengenalan pola.

2.5 Algoritma *Hamming Distance*

Algoritma ini mencari jarak antara dua buah teks, namun berbeda dengan *Levenshtein Distance* yang melakukan konsiderasi untuk pemasukan atau penghapusan huruf, algoritma ini hanya mencari banyaknya subtitusi yang dilakukan untuk dua buah teks. Sehingga pergeseran sebuah pola tidak dihitung menggunakan algoritma ini. Namun waktu yang dibutuhkan untuk mendapatkan hasil jauh lebih cepat dibandingkan dengan *Levenshtein Distance*.

2.6 Regex

Regex, atau Regular Expressions, merupakan sebuah teks (string) yang mendefinisikan sebuah pola pencarian sehingga dapat membantu kita untuk melakukan matching (pencocokan), locate (pencarian), dan manipulasi teks. Regex banyak digunakan dalam berbagai bahasa pemrograman dan alat-alat pengolahan teks untuk menemukan dan menggantikan pola tertentu dalam sebuah teks. Regex juga dapat digunakan untuk validasi input, parsing data, dan berbagai tugas pengolahan teks lainnya.



Gambar 1: Regex[1]

BAB III
ANALISIS PEMECAHAN MASALAH

3.1 Langkah Pemecahan Masalah

Terdapat dua permasalahan dalam tugas besar ini, pertama mengenai bagaimana melakukan pencarian *fingerprint* paling sesuai dari gambar masukan dengan gambar yang sudah ada di dalam *database*. Kemudian, dikarenakan terdapat data nama yang mungkin *corrupt*, atau rusak, maka program harus mencari tahu nama asli sebelum data tersebut *corrupt* berdasarkan data yang lain. Secara garis besar, langkah-langkah yang dilakukan untuk setiap permasalahan tersebut adalah sebagai berikut:

1. Pencarian *fingerprint*
    a. Menyiapkan data yang berupa *folder* menuju gambar *fingerprint*, dan basis data yang berisi tabel (lebih lengkapnya dijelaskan pada 4.1.1)
    b. Meminta masukan gambar dari pengguna berupa *fingerprint* yang akan dicari, serta metode yang akan digunakan untuk pencarian *fingerprint*
    c. Mengambil data pixel 30x1 ditengah gambar, mengubah data pixel menjadi data binary, lalu mengubah binary menjadi string ASCII
    d. Melakukan komparasi dengan seluruh data pada *database* menggunakan metode yang dipilih oleh pengguna
    e. Jika tidak ditemukan, lakukan perbandingan dengan menggunakan Hamming Distance
    f. Tampilkan *fingerprint* yang sudah ditemukan kepada tampilan layar pengguna

Gambar 2: Langkah Pencarian *fingerprint*[3]

2. Pencarian nama alay
    a. Menyiapkan data biodata dengan mengambilnya dari database (data.db)
    b. Membuat list string berupa nama *corrupt* dari biodata, dan konversi menjadi normal (tanpa mengurus *corrupt* singkatan) menggunakan Regex
    c. Menggunakan list tersebut, cari nama biodata yang serupa dengan nama hasil pencarian *fingerprint* menggunakan metode yang dipilih oleh pengguna, sekaligus mengurus *corrupt* singkatan
    d. Tampilkan biodata yang sudah ditemukan kepada tampilan layar pengguna

3.2 Proses Penyelesaian Solusi

Terdapat dua kelas yang bernama KnuthMorrisPrat dan BoyerMoore, masing-masing memiliki sebuah fungsi yang bernama ProcessAll. Berikut merupakan penjelasan dari setiap algoritma:

1. KnuthMorrisPrat
    a. Membuat sebuah *lookup table* yang merupakan *border function*. Data ini memiliki tipe int[], dengan indeks adalah *k*, dan nilai didalam sebuah indeks adalah *b(k)*
    b. Melakukan iterasi dari 0 sampai sebelum panjang dari teks
    c. Tahap awal yang dilakukan setiap iterasi adalah memeriksa indeks teks dan pola memiliki karakter yang sama. Jika iya, maka indeks keduanya ditambahkan satu
    d. Selanjutnya periksa indeks dari pola sudah sepanjang pola, jika sudah maka pola sudah ditemukan
    e. Jika indeks dari keduanya berbeda, maka kita ubah indeks dari pola menggunakan data *border function* yang sudah dibuat
    f. Lakukan iterasi selanjutnya, kembali lagi menuju tahap c

2. BoyerMoore
    a. Buat *lookup table* yang berisi *last occurence* dari setiap karakter. Tabel ini memiliki tipe int[], dan terdapat 256 elemen, sesuai dengan banyaknya karakter yang mungkin dibuat pada struktur data char
    b. Lakukan iterasi dengan metode *looking glass*, yaitu mulai membandingkan dengan indeks terakhir dari pola, namun dimulai dari awal teks
    c. Jika terdapat perbedaan antara pola dengan teks, maka gunakan tabel *last occurence* yang telah dibuat untuk mengubah indeks dari teks
    d. Pola ditemukan apabila seluruh banyaknya perbandingan dari metode *looking glass* sudah sebanyak panjang dari pola, atau dengan kata lain indeks dari pola kurang dari nol.

3.3 Fitur Fungsional dan Arsitektur Aplikasi Desktop

Aplikasi yang dibuat berbasis C# dalam kakas Visual Studio .NET, menggunakan GUI Eto.Forms. Total terdapat 9 (sembilan) *project* secara terpisah. Empat diantaranya bertanggung jawab untuk program GUI. Terdiri dari tiga sesuai dengan *platform* masing-masing seperti Windows, MacOS, dan Linux. Kemudian satu sebagai *project* GUI secara general. Empat *project* ini menghasilkan *executable* yang dapat dijalankan. Selanjutnya, terdapat satu *project* yang bernama CLI, berguna untuk melakukan *testing* dari bagian program yang lain tanpa perlu melakukan *compile* GUI, sehingga pengetesan dilakukan dengan lebih cepat.

Bentuk selanjutnya berupa *library* yang memiliki fungsi utama yang dapat dipanggil oleh *executable* GUI. Berikut merupakan penjelasan masing-masing dari *library* tersebut:

1. Algorithm

   Berisi algoritma seperti Booyer-Moree, Knuth-Morris-Pratt dan Regex
2. Converter

   Mengubah gambar dalam bentuk bitmap menjadi ASCII, yang nantinya akan dibandingkan menggunakan algoritma yang telah dibuat
3. Database

   Bertanggung jawab menghubungkan antara database sqlite dengan program utama.

Pada pengetesan seluruh *library* diatas, diperlukan sebuah *database* yang telah terisi untuk melakukan pengetesan. Terdapat sebuah *project* yang bernama Faker, menghasilkan *executable* yang berguna untuk membuat data berdasarkan gambar pada *folder* “Data”.

3.4 Ilustrasi Kasus

Misalkan seluruh *database* sudah disiapkan, atau sudah terdapat *folder* berisi gambar *fingerprint* yang akan dibandingkan dengan masukan pengguna. Selanjutnya, pengguna akan memilih gambar yang akan dibandingkan, dan metode pencariian sesuai yang diinginkan. Berikut merupakan hal yang terjadi dalam program setelah pengguna menekan tombol “Search”:

1. Fungsi search akan dipanggil, dan mengambil informasi mengenai path dari gambar masukan dan metode pencarian yang akan dipakai.
2. Program akan memeriksa dan melakukan validasi mengenai gambar yang dimasukan
3. Timer akan dimulai untuk mendapatkan waktu eksekusi dari pencarian
4. Program menjalakan algoritma sesuai dengan pilihan dari masukan pengguna
5. Pada algoritma tersebut, dilakukan juga pencarian nama yang rusak dari nama yang didapat dari tabel sidik_jari
6. Program memberhentikan timer dan mencatat waktu ekseskusi program
7. Program menampilkan informasi mengenai biodata yang ditemukan

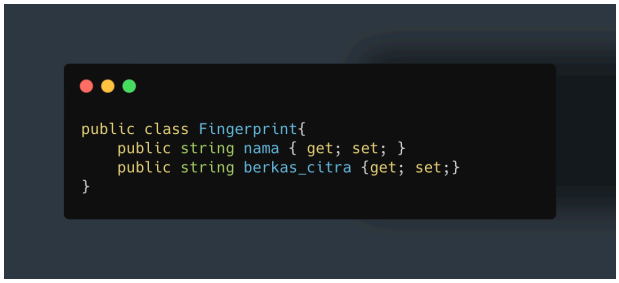
BAB IV
IMPLEMENTASI DAN PENGUJIAN

4.1 Spesifikasi dan Struktur Program

4.1.1 Struktur Data

Pada basis data, terdapat dua tabel yang berisi Biodata dan Fingerprint. Seluruh attribut pada Biodata sudah cukup jelas. Kemudian pada tabel Fingerprint, attribut berkas_citra merupakan path menuju gambar. Pada bahasa pemrograman C#, digunakan kelas sebagai berikut ini untuk merepresentasikan dua tabel tersebut.

```
public class Biodata{
    public string nama  {get; set;}
    public string tempat_lahir {get; set;}
    public string tanggal_lahir {get; set;}
    public string jenis_kelamin {get; set;}
    public string golongan_darah {get; set;}
    public string alamat {get; set;}
    public string agama {get; set;}
    public string status_perkawinan {get; set;}
    public string pekerjaan {get; set;}
    public string kewarganegaraan {get; set;}
}
```



```
public class Fingerprint{
    public string nama { get; set; }
    public string berkas_citra {get; set;}
}
```

4.1.2 Implementasi Algoritma KMP

Pada Algoritma KMP dibuat kelas tersendiri yaitu terletak pada folder Algorithm dengan file KnuthMorrisPratt.cs yang mana di dalam kelas tersebut terdapat public class dengan nama yang sama dengan nama file yang berisi method-method yang ada di bawah.

1. Fungsi ProcessAll

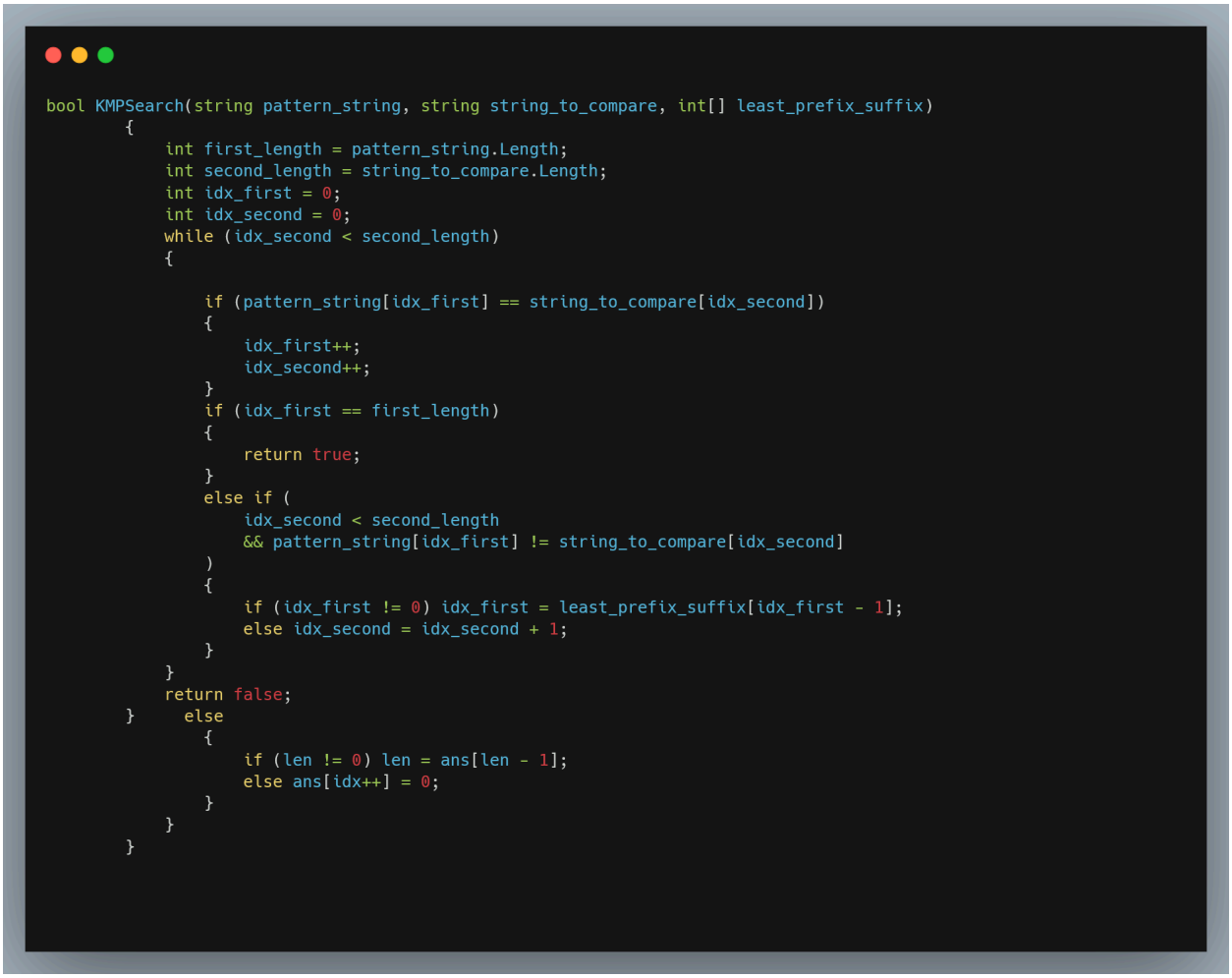
Fungsi ProcessAll adalah fungsi yang ditujukan untuk memproses pencarian dengan memanggil fungsi KMPSearch di dalamnya sebanyak data yang ada di dalam database. Keluaran yang dihasilkan dari ProcessAll ini adalah List tuple(string, string, int) yang mana ketiganya merepresentasikan (Closest_Match, Data_asli, Jarak_Terdekat). Jika *pattern* dengan Jarak_Terdekat sama dengan 0, *pattern* tersebut akan dimasukkan ke dalam List result, jika tidak akan dilakukan pencarian kembali menggunakan *levenshtein distance* untuk mencari jarak yang paling dekat dengan pola yang ada.

```
public List<(string, string, int)> ProcessAll(string pattern, List<string> database)
        {
            List<(string, string, int)> result = new List<(string, string, int)>();
            int[] lps = GenerateLPS(pattern);
            foreach (var data in database)
            {
                if (KMPSearch(pattern, data, lps))
                    result.Add((pattern, data, 0));
            }
            if (result.Count == 0)
            {
                foreach (var data in database)
                {
                    (string, int) closestMatch = Util.FindClosestMatch(pattern, data);
                    if (!string.IsNullOrEmpty(closestMatch.Item1))
                    {
                        result.Add((closestMatch.Item1, data, closestMatch.Item2));
                    }
                }
            }
            return result.OrderBy(tuple => tuple.Item3).ToList();
        }
  return go(f, seed, [])
}
```

2. Fungsi KMPSearch

Fungsi KMPSearch adalah fungsi pencarian dengan algoritma KMP fungsi ini mengembalikan *return value* berupa *boolean* mengapa? karena KMP sendiri adalah

algoritma yang *exact match* sehingga jika tidak ada sama sekali yang sama maka *return value*-nya adalah false.



```
bool KMPSearch(string pattern_string, string string_to_compare, int[] least_prefix_suffix)
        {
            int first_length = pattern_string.Length;
            int second_length = string_to_compare.Length;
            int idx_first = 0;
            int idx_second = 0;
            while (idx_second < second_length)
            {

                if (pattern_string[idx_first] == string_to_compare[idx_second])
                {
                    idx_first++;
                    idx_second++;
                }
                if (idx_first == first_length)
                {
                    return true;
                }
                else if (
                    idx_second < second_length
                    && pattern_string[idx_first] != string_to_compare[idx_second]
                )
                {
                    if (idx_first != 0) idx_first = least_prefix_suffix[idx_first - 1];
                    else idx_second = idx_second + 1;
                }
            }
            return false;
        }     else
                {
                    if (len != 0) len = ans[len - 1];
                    else ans[idx++] = 0;
                }
            }
        }
```

3. Fungsi GenerateLPS

   Fungsi ini ditujukan untuk membuat tabel *bounding* pada algoritma KMP

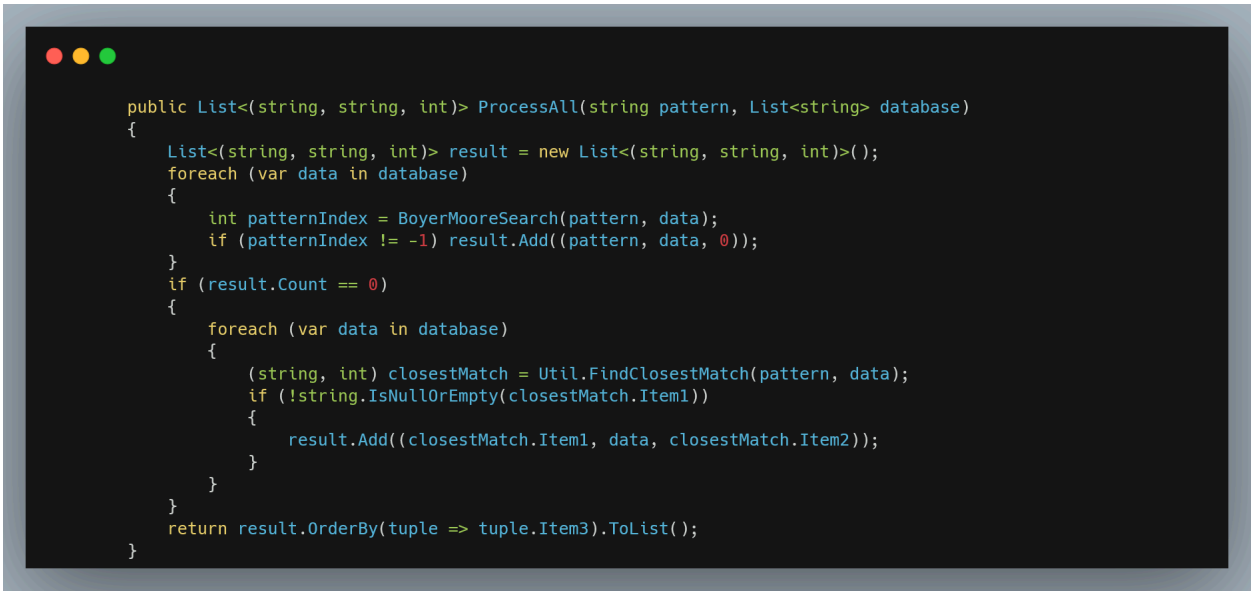
```
private static int[] GenerateLPS(string pattern)
 {
     int m = pattern.Length;
     int[] lps = new int[m];
     int length = 0;
     lps[0] = 0;
     int i = 1;
     while (i < m)
     {
         if (pattern[i] == pattern[length])
         {
             length++;
             lps[i] = length;
             i++;
         }
         else
         {
             if (length != 0)
             {
                 length = lps[length - 1];
             }
             else
             {
                 lps[i] = 0;
                 i++;
             }
         }
     }
     return lps;
 }
```

4.1.3 Implementasi Algoritma Boyer-Moore

Algoritma Boyer-Moore adalah algoritma *String Matching* tidak seperti algoritma *String Matching* lainnya. Pada Boyer-Moore pencocokan dimulai dari paling kanan.

1. ProcessAll

   Method ini memproses semua data yang ada di dalam database



```
public List<(string, string, int)> ProcessAll(string pattern, List<string> database)
{
    List<(string, string, int)> result = new List<(string, string, int)>();
    foreach (var data in database)
    {
        int patternIndex = BoyerMooreSearch(pattern, data);
        if (patternIndex != -1) result.Add((pattern, data, 0));
    }
    if (result.Count == 0)
    {
        foreach (var data in database)
        {
            (string, int) closestMatch = Util.FindClosestMatch(pattern, data);
            if (!string.IsNullOrEmpty(closestMatch.Item1))
            {
                result.Add((closestMatch.Item1, data, closestMatch.Item2));
            }
        }
    }
    return result.OrderBy(tuple => tuple.Item3).ToList();
}
```

2. BoyerMooreSearch

   Pada method ini akan dilakukan pencarian string yang cocok antara pattern dan text sesuai dengan algoritma *Boyer Moore Search*

```
private int BoyerMooreSearch(string pattern, string text)
{
    int m = pattern.Length;
    int n = text.Length;

    int[] badChar = BuildBadCharacterTable(pattern);
    int s = 0;
    while (s <= (n - m))
    {
        int j = m - 1;
        while (j >= 0 && pattern[j] == text[s + j]) j--;
        if (j < 0)
        {
            s += (s + m < n) ? m - badChar[text[s + m]] : 1;
            return s;
        }
        else s += Math.Max(1, j - badChar[text[s + j]]);
    }
    return -1;
}
```

3. BuildCharacterTable

   method ini digunakan adalah preproses untuk melakukan BoyerMooreSearch yang nantinya dijadikan patokan untuk menggeser pattern.

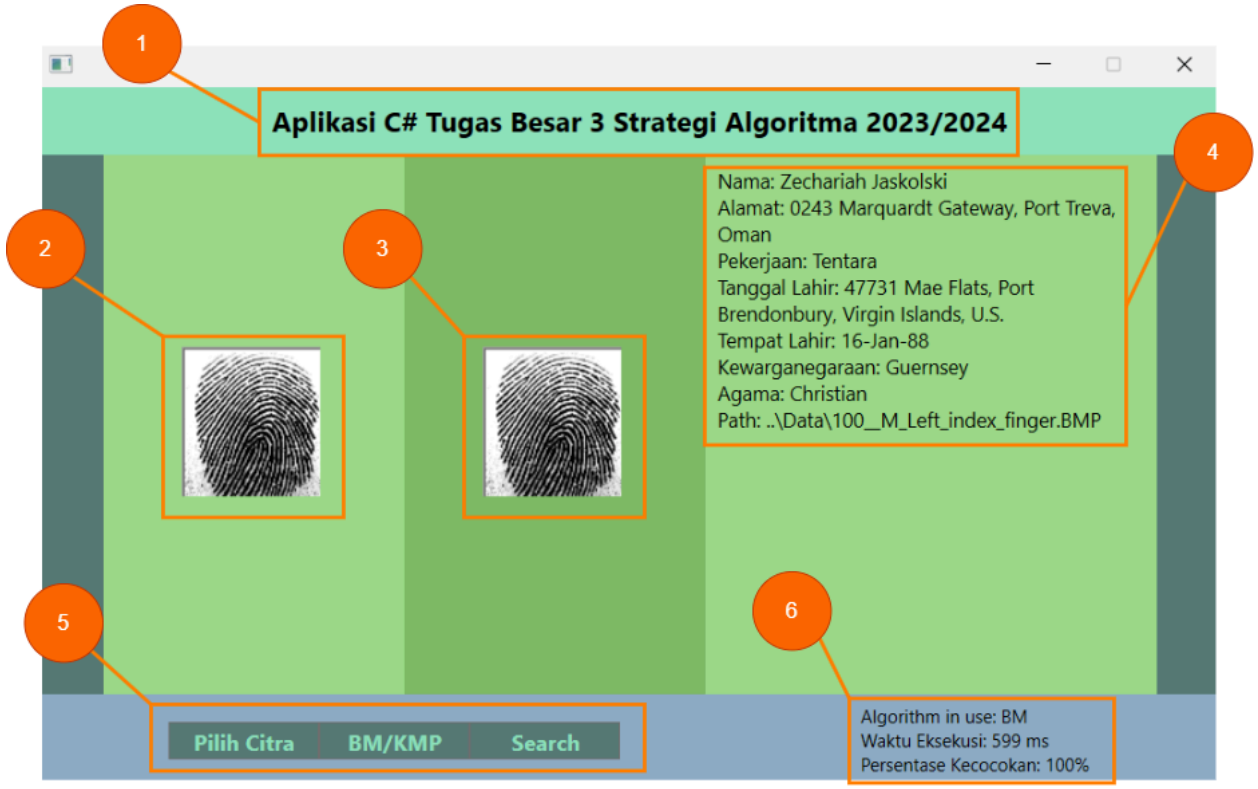
```
private int[] BuildBadCharacterTable(string pattern)
{
    int[] badChar = new int[256];
    int m = pattern.Length;

    for (int i = 0; i < 256; i++)
        badChar[i] = -1;

    for (int i = 0; i < m; i++)
        badChar[(int)pattern[i]] = i;

    return badChar;
}
```

4.2 Penggunaan Program



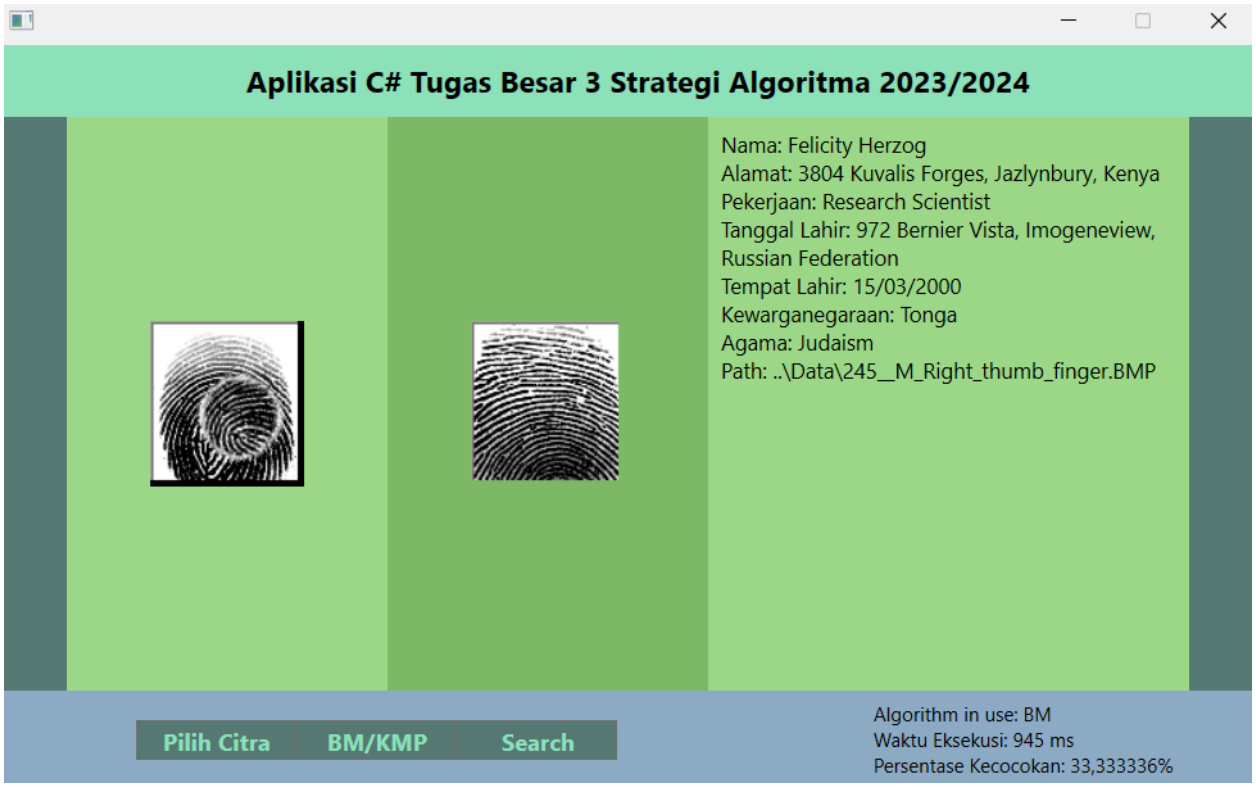
1. Judul Aplikasi
2. Gambar Input
3. Gambar Output
4. Biodata
5. Tombol Pilih Citra, BM/KMP, Search
6. Data Algoritma yang dipakai, Waktu eksekusi, dan Persentase kecocokan

Berikut adalah cara menggunakan program. Pertama, tekan tombol "Pilih Citra" untuk memilih gambar input. Selanjutnya, pilih algoritma yang akan digunakan dengan menekan tombol "BM/KMP". Setelah itu, tekan tombol "Search". Hasil pencarian berupa gambar dan biodata yang sesuai, serta waktu eksekusi dan persentase kecocokan akan ditampilkan.

4.3 Pengujian

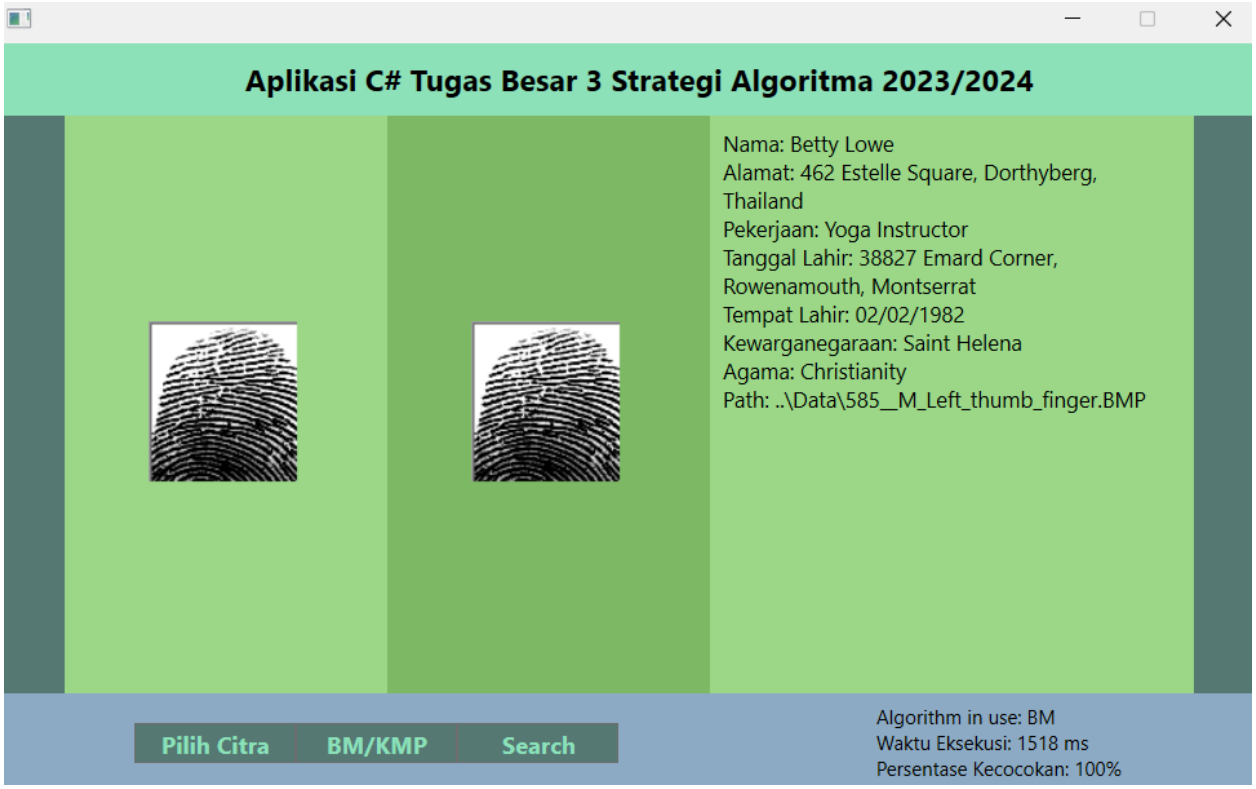
4.3.1 BM

1. File input: 1__M_Left_index_finger_CR.BMP

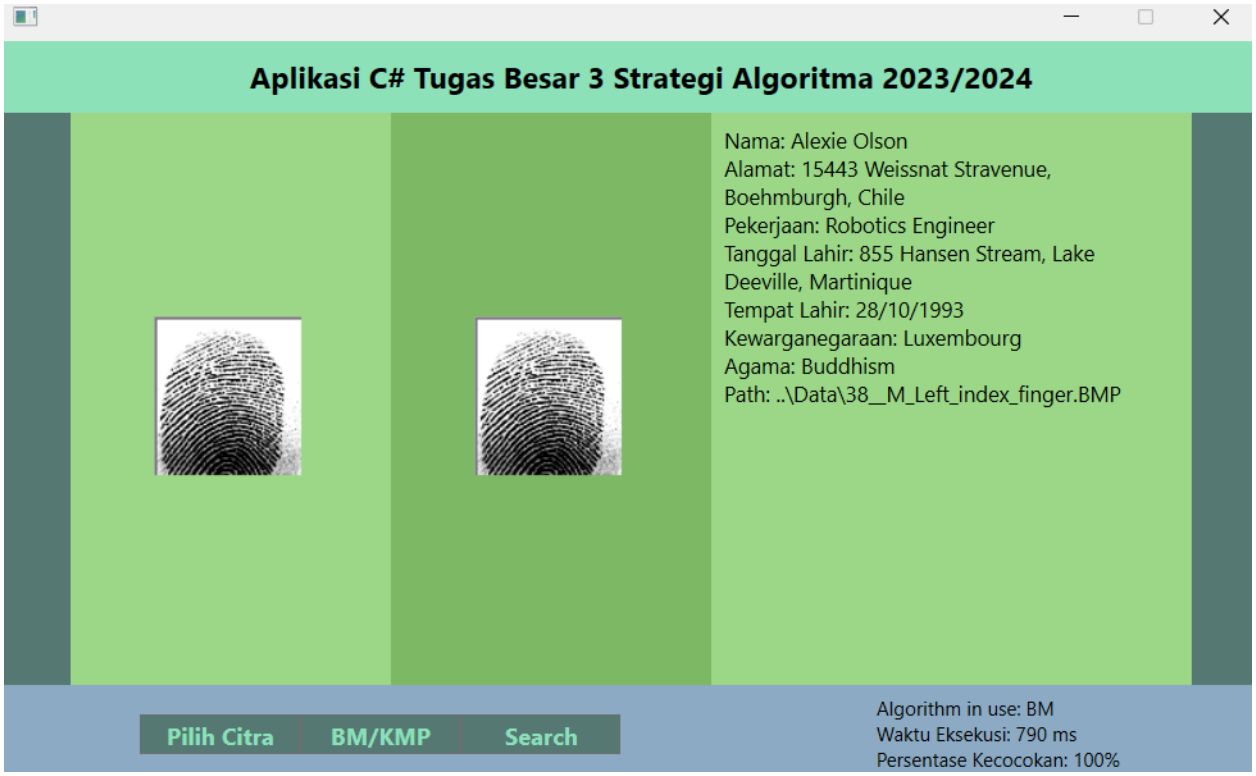


2. File input: 101__M_Right_middle_finger_CR.BMP

3. File input: 585__M_Left_thumb_finger.BMP

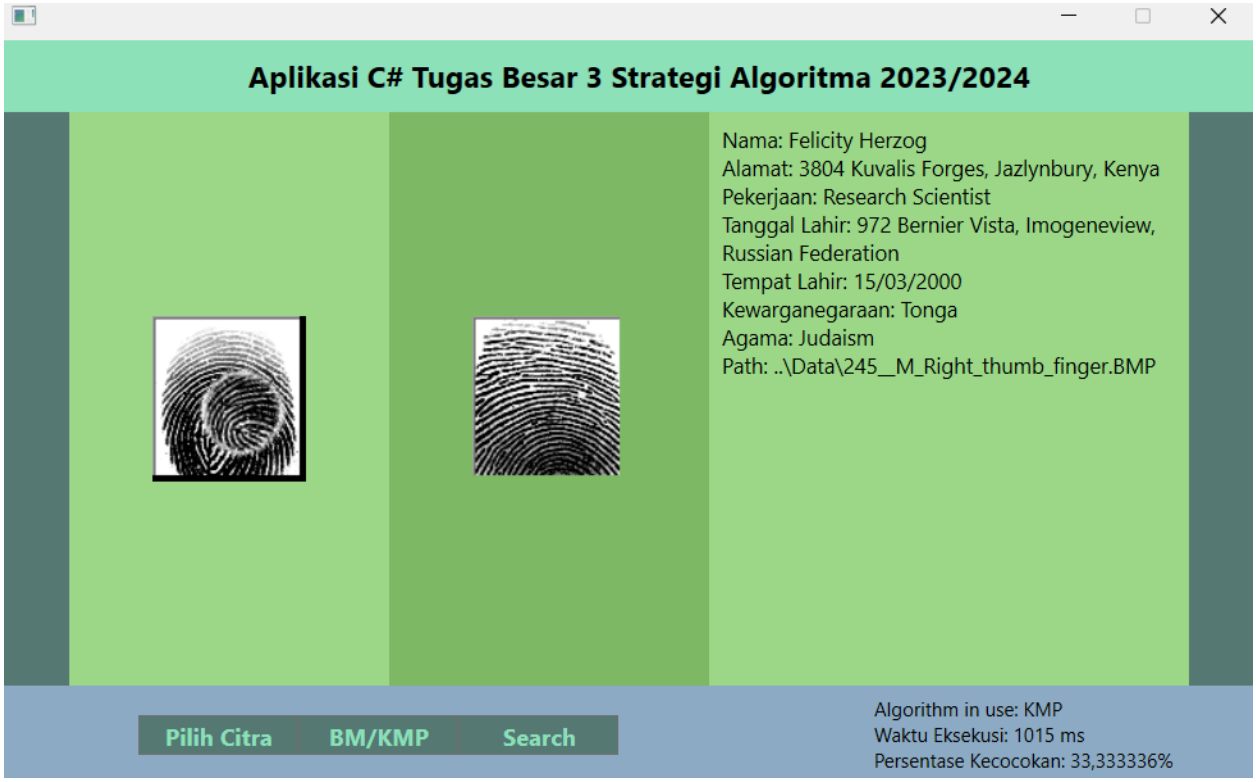


4. File input: 38__M_Left_index_finger.BMP

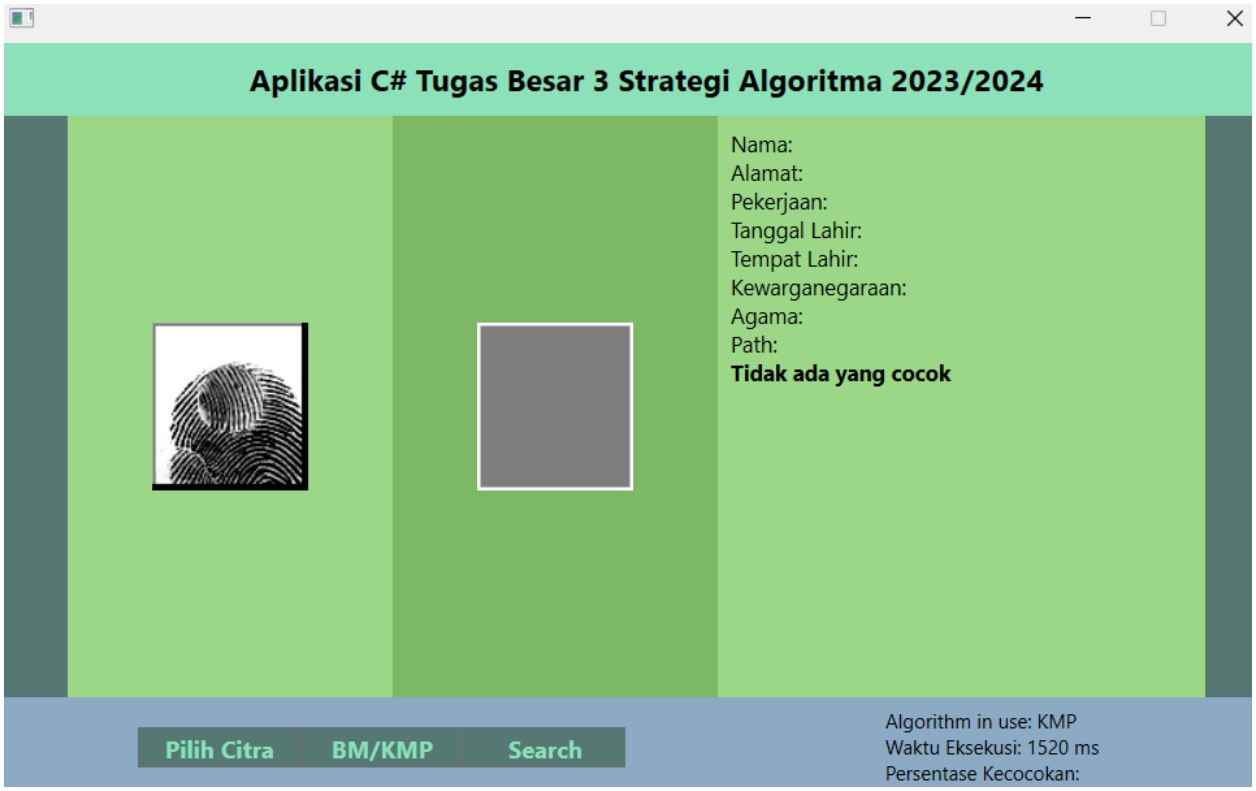


4.3.2 KMP

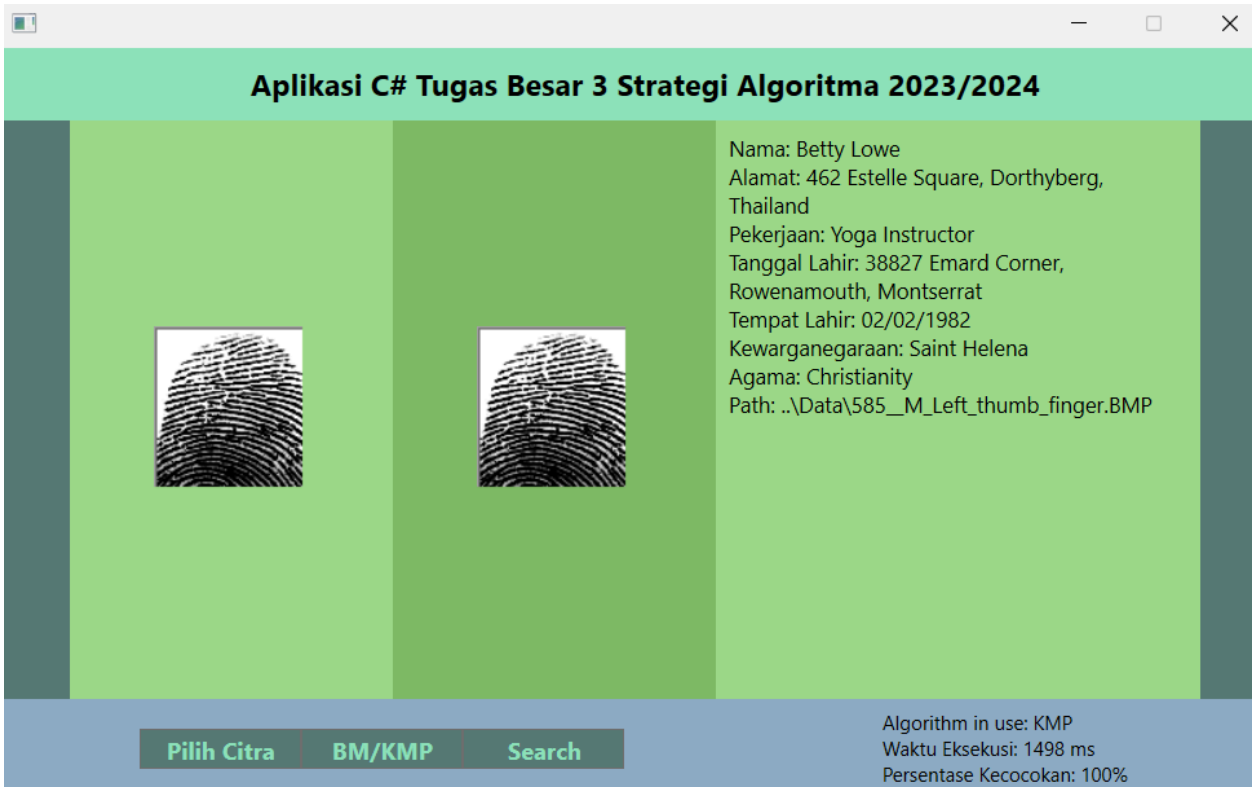
1. File input: 1__M_Left_index_finger_CR.BMP



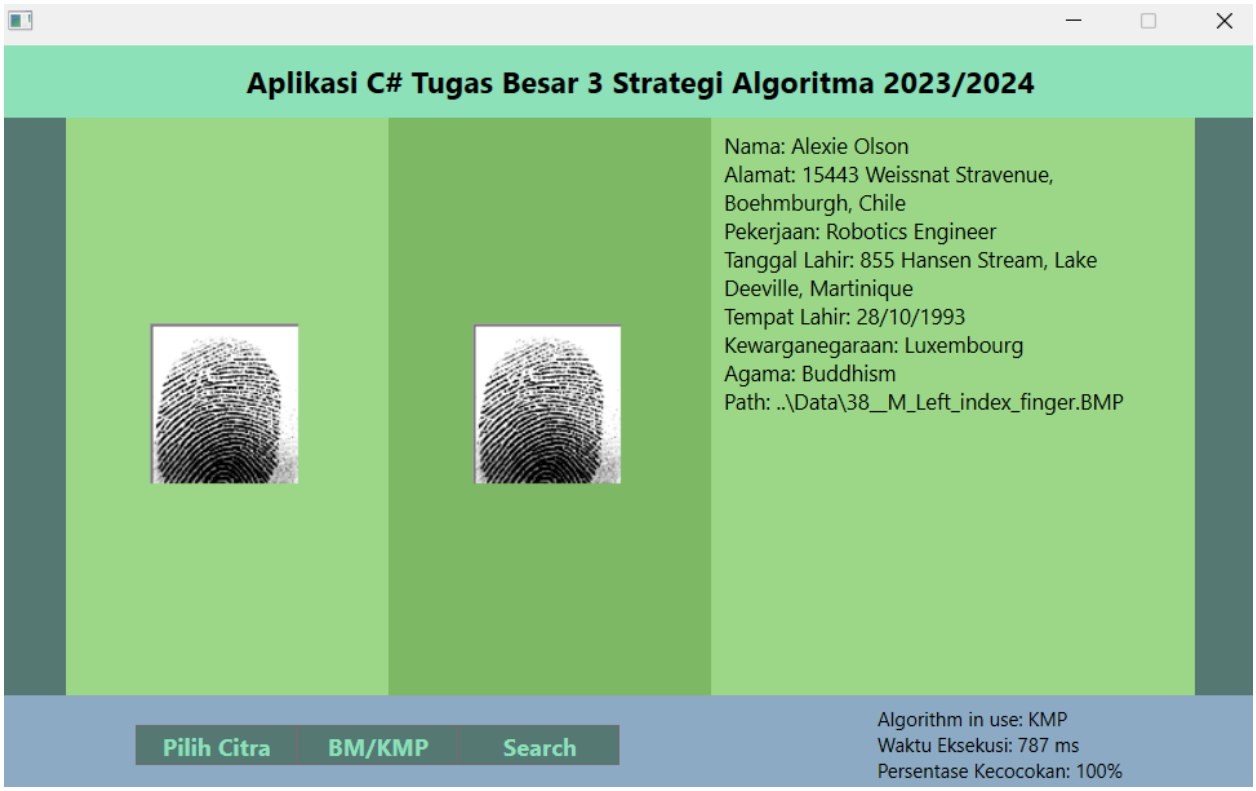
2. File input: 101__M_Right_middle_finger_CR.BMP



3. File input: 585__M_Left_thumb_finger.BMP



4. File input: 38__M_Left_index_finger.BMP



Waktu Eksekusi BM: 945, 1608, 1518, 790 ms

Waktu Eksekusi KMP: 1015, 1520, 1498, 787 ms

Persentase Kecocokan: Tidak ditemukan (<25%), (KMP/BM): 33,3%, 100%, 100%

4.4 Analisis Hasil

Dalam pengujian waktu eksekusi algoritma, Boyer-Moore (BM) menunjukkan hasil dengan waktu eksekusi berturut-turut sebesar 945 ms, 1608 ms, 1518 ms, 790 ms. Sementara itu, algoritma Knuth-Morris-Pratt (KMP) mencatat waktu eksekusi sebesar 1015 ms, 1520 ms, 1498 ms, dan 787 ms. Dari data ini, terlihat bahwa secara umum KMP memiliki waktu eksekusi yang serupa dengan BM, dengan perbedaan yang tidak signifikan.

Selanjutnya, analisis persentase kecocokan menunjukkan bahwa dalam untuk algoritma BM dan KMP, persentase kecocokan berturut-turut adalah Tidak ditemukan, 33,3%, 100%, dan 100%, dengan kedua gambar dengan persentase 100% berupa gambar yang sama. Hal ini menunjukkan bahwa BM dan KMP berhasil menemukan hasil fingerprint yang sesuai, dan Levenshtein distance dapat mencakup fingerprint yang tidak ditemukan dengan BM atau KMP.

BAB V
KESIMPULAN DAN SARAN

5.1 Kesimpulan

Dalam tugas besar ini, telah dikembangkan sebuah aplikasi berbasis GUI menggunakan bahasa C# untuk mendeteksi individu melalui citra sidik jari. Aplikasi ini mengimplementasikan algoritma pencocokan string Boyer-Moore (BM) dan Knuth-Morris-Pratt (KMP), serta menggunakan Regular Expression (Regex) untuk mengatasi data nama yang korup pada biodata dalam database. Dari hasil pengujian, terlihat bahwa algoritma KMP umumnya menunjukkan waktu eksekusi yang sedikit lebih cepat dibandingkan BM, meskipun perbedaannya tidak signifikan. Persentase kecocokan untuk kedua algoritma menunjukkan bahwa KMP dan BM dapat menemukan hasil yang serupa. Hasil ini menunjukkan bahwa kombinasi algoritma-algoritma ini dapat digunakan untuk membangun sistem deteksi individu berbasis biometrik yang andal.

5.2 Saran dan Refleksi

Sebuah hambatan dalam pengerjaan tugas besar ini adalah pemilihan bahasa yang sangat tidak ramah dengan *cross-platfrom*. Bahasa tersebut adalah C#, yang menggunakan dotnet sebagai *framework* dan dibuat oleh Microsoft. Salah satu *library* GUI yang dapat digunakan adalah Wpf, dan menggunakan Visual Studio untuk *drag and drop*, namun hanya bisa digunakan sistem operasi Windows. Sehingga terdapat salah satu anggota kami tidak bisa mengerjakannya. Oleh karena itu kami menggunakan *library* Eto.Form untuk membuat GUI. Namun tetap saja terdapat permasalahan dalam *library* sqlite pada akhir pengerjaan tugas, sehingga tidak bisa dijalankan di *platform* lain. Saran kami dari pengerjaan tubes ini adalah pemilihan bahasa selanjutnya mungkin melakukan konsiderasi bagi pengguna non-Windows.

LAMPIRAN

- Tautan *repository* Github: https://github.com/Intermaze/Tubes3_PatternAddict

DAFTAR PUSTAKA

[1] Munir, Rinaldi. 2024. “Pencocokan String”.
https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Pencocokan-string-2021.pdf
[2] Munir, Rinaldi. 2024. “String Matching dengan Regex”.
https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2022-2023/String-Matching-dengan-Regex-2019.pdf
[3] Lab IRK. 2024. “Spesifikasi Tugas Besar 3 Stima 2023/2024”.
https://docs.google.com/document/d/15Dk7FbcraVDCYDYtT6d743h649ZMN6xE