

**Laporan Tugas Besar 3
IF2211 Strategi Algoritma**

**Pemanfaatan Pattern Matching dalam Membangun Sistem
Deteksi Individu Berbasis Biometrik Melalui Citra Sidik Jari
Semester II Tahun 2023/2024**



Disusun Oleh:

Ahmad Naufal Ramadan 13522005
Muhammad Althariq Fairuz 13522027
Muhammad Dava Fathurrahman 13522114

**Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung
2024**

DAFTAR ISI

BAB 1	1
Deskripsi Tugas	1
BAB 2	2
Landasan Teori	2
2.1. Algoritma Knuth-Morris-Pratt (KMP)	2
2.2. Algoritma Boyer-Moore	2
2.3. Regular Expression (Regex)	3
2.4. Pengukuran Kemiripan dengan Longest Common Subsequence (LCS)	4
2.5. Deskripsi Singkat Aplikasi Desktop yang Dibangun	4
BAB 3	5
Aplikasi Pemecahan Masalah	5
3.1. Langkah-langkah Penyelesaian Masalah	5
3.2. Proses Penyelesaian Masalah dengan KMP dan BM	6
3.3. Fitur Fungsional dari Aplikasi yang Dibangun	6
3.4. Ilustrasi Kasus	7
BAB 4	8
Implementasi dan Pengujian	8
4.1. Spesifikasi Program	8
4.2. Tata Cara Penggunaan Program	13
4.3. Hasil Pengujian	13
4.4. Analisa Hasil Pengujian	20
BAB 5	22
Kesimpulan dan Saran	22
5.1. Kesimpulan	22
5.2. Saran	22
Lampiran	23
Daftar Pustaka	24

BAB 1

Deskripsi Tugas

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.



Gambar 1 Simulasi pemindaian telapak tangan pada sistem pemerintahan
(Sumber: [Walikota Makasar Scan Tangan](#))

BAB 2

Landasan Teori

2.1. Algoritma Knuth-Morris-Pratt (KMP)

Algoritma Knuth-Morris-Pratt (KMP) mencari pola dari suatu string dalam suatu teks secara berurutan dari kiri ke kanan (mirip dengan algoritma brute force). Namun, KMP menggeser pola dengan lebih cerdas daripada algoritma brute force.

Algoritma ini memiliki beberapa kelebihan yang membuatnya efisien dalam pencocokan string. Mari kita bahas beberapa di antaranya:

1. Pergeseran yang Efisien

- Perbedaan utama antara algoritma KMP dan pencocokan string secara brute force adalah pergeseran ketika ditemukan ketidakcocokan.
- Pada pencocokan string secara brute force, pergeseran hanya dilakukan satu karakter ke kanan, dan pencocokan akan diulang dari awal pola jika ada ketidakcocokan.
- Namun, pada KMP, pergeseran dapat dilakukan beberapa karakter sekaligus, mengurangi perbandingan yang tidak perlu dan mempercepat kinerja program.

2. Preprocessing Pola

- Sebelum menjalankan pencocokan, KMP melakukan pemrosesan pada pola.
- Ini menghasilkan array bilangan bulat yang disebut Longest Proper Prefix Which Is Suffix (LPS) atau fungsi pinggiran.
- LPS memberikan informasi tentang berapa banyak karakter yang harus dilewati ketika ditemukan ketidakcocokan.
- Dengan demikian, KMP dapat menghindari pencocokan karakter yang sudah diketahui akan cocok, mengurangi jumlah perbandingan.

3. Tidak Ada Pergeseran Mundur

- Algoritma KMP tidak pernah melakukan pergeseran mundur.
- Ini membuatnya cocok untuk memproses file yang sangat besar, seperti program dengan kode yang panjang.

2.2. Algoritma Boyer-Moore

Algoritma Boyer-Moore adalah salah satu algoritma pencarian pola/pencocokan string yang efisien. Algoritma ini memanfaatkan informasi yang telah diproses sebelumnya untuk menghindari sebagian teks sehingga memiliki kompleksitas waktu yang lebih

rendah dibandingkan dengan algoritma pencarian lainnya. Algoritma Boyer-Moore membandingkan elemen dari akhir pola dengan teks. Selama tahap pemrosesan pada pola, algoritma ini mengumpulkan informasi yang memungkinkan pencarian untuk melewati bagian-bagian teks tertentu. Hal ini menghasilkan kompleksitas waktu yang lebih rendah dibandingkan dengan algoritma pencarian lainnya.

Algoritma ini memiliki beberapa kelebihan yang membuatnya efisien dalam pencocokan string. Mari kita bahas beberapa di antaranya:

1. Efisiensi

Algoritma ini memanfaatkan pra-pemrosesan dan heuristik untuk mengurangi jumlah perbandingan karakter sehingga lebih cepat daripada algoritma pencarian string lainnya.

2. Pencocokan dari Akhir Pola

Algoritma ini memulai pencocokan dari karakter terakhir dalam pola sehingga dapat mengurangi jumlah pergeseran dan mempercepat pencarian.

2.3. Regular Expression (Regex)

Regex (singkatan dari *Regular Expression*) adalah sebuah teks yang mendefinisikan pola pencarian dalam string. Regex dapat melakukan pencocokan, pencarian, dan manipulasi teks dengan lebih efisien.

Berikut beberapa kelebihan dari Regex:

1. Validasi Input

- Regex dapat memverifikasi apakah string yang diterima sesuai dengan pola yang ditentukan. Contohnya dalam validasi format email.

2. Manipulasi Teks

- Regex mampu mengganti, menghapus, atau memanipulasi bagian-bagian teks berdasarkan pola.
- Contoh penggunaannya ada pada penghapusan karakter non-alfanumerik dari string atau mengganti semua kata tertentu dengan kata lain.

2.4. Pengukuran Kemiripan dengan Longest Common Subsequence (LCS)

Longest Common Subsequence (LCS) didefinisikan sebagai urutan terpanjang yang umum untuk semua urutan yang diberikan selama unsur-unsur urutan tidak diperlukan untuk menempati posisi berturut-turut dalam urutan asli. LCS digunakan ketika tidak ada kemiripan yang dihasilkan saat menggunakan algoritma KMP atau BM sehingga akan dicari kemiripannya melalui subsekuensi terpanjang yang ada dalam input string (sidik jari yang telah diubah ke ASCII-8bit) yang diberikan sebelumnya. Subsekuensi ini mempertahankan urutan elemen yang sama seperti dalam urutan asli, tetapi tidak harus berurutan.

2.5. Deskripsi Singkat Aplikasi Desktop yang Dibangun

Aplikasi yang dibuat dapat menerima input suatu citra sidik jari yang kemudian akan dicari kemiripan sidik jari tersebut dengan yang ada di database. Proses pencarian kemiripan ini dilakukan dengan menggunakan algoritma string matching KMP dan BM. Jika tidak ditemukan kecocokan yang tepat, algoritma LCS akan digunakan untuk menemukan sidik jari yang paling mirip. Setelah sidik jari yang paling mirip ditemukan, biodatanya akan dicari dengan membandingkan nama pemilik sidik jari dengan data nama pada tabel biodata. Data nama pada tabel biodata mungkin rusak, oleh karena itu dilakukan rekonstruksi string dengan menggunakan regex untuk mengembalikan nama asli pemilik sidik jari. Data dari pemilik sidik jari tersebut akan ditampilkan di layar beserta waktu pencarian dan persentase kemiripan.

BAB 3

Aplikasi Pemecahan Masalah

3.1. Langkah-langkah Penyelesaian Masalah

Berikut adalah langkah penyelesaian masalah dalam mengidentifikasi sidik jari dengan menggunakan algoritma KMP dan BM:

1. Akuisisi dan Pra-Pemrosesan Sidik Jari

Proses dimulai dengan menerima gambar sidik jari, mengubahnya ke *grayscale*, lalu mengubahnya ke bentuk binary.

2. Konversi ke Kode ASCII

Untuk menerapkan algoritma KMP dan BM, gambar sidik jari dikonversi ke format yang dapat diproses oleh algoritma pencocokan string ini. Salah satu pendekatannya adalah memetakan intensitas piksel ke karakter ASCII berdasarkan kode binary-nya yang telah dilakukan sebelumnya.

3. Pembuatan Database Template Sidik Jari

Database disiapkan dengan menyimpan beberapa sampel sidik jari. Setiap sampel mewakili sidik jari unik dan berisi informasi tentang pemilik sidik jari tersebut.

4. Algoritma KMP

Algoritma KMP membuat sebuah array yang telah diproses sebelumnya (Longest Proper Prefix Which Is Suffix, atau LPS, atau fungsi pinggiran) untuk menghindari perbandingan karakter yang tidak perlu. Kemudian, algoritma ini membandingkan 10 pixel terakhir pada bagian bawah dari sidik jari yang diinput dengan keseluruhan sampel sidik jari pada database. Jika sidik jari yang diberikan sudah ada di dalam database, KMP akan mengembalikan nilai true. Jika tidak, akan digunakan algoritma LCS untuk mencari persentase kemiripan antara sidik jari yang diberikan dengan keseluruhan sidik jari yang ada di database.

5. Algoritma Boyer-Moore

Algoritma BM memanfaatkan array last occurrence untuk mengurangi jumlah perbandingan karakter yang tidak perlu. Proses ini dilakukan sebelum pencarian untuk menghindari perbandingan yang tidak perlu. Selanjutnya, algoritma BM membandingkan 10 pixel terakhir dari bagian bawah sidik jari input dengan keseluruhan sampel sidik jari dalam database. Jika sidik jari input sudah ada dalam database, algoritma BM akan mengembalikan nilai true. Jika tidak ditemukan, algoritma akan melanjutkan dengan menggunakan algoritma Longest Common

Subsequence (LCS) untuk mencari persentase kemiripan antara sidik jari input dengan sidik jari dalam database.

6. Regex

Regex digunakan ketika ada data pada database yang *corrupt* sehingga perlu diperbaiki terlebih dahulu. Data yang *corrupt* ini meliputi penulisan nama yang disingkat, penulisan huruf vokal dengan angka, dan penggunaan huruf kapital yang tidak benar. Tiap nama pada database memiliki regexnya tersendiri yang dibuat berdasarkan nama aslinya (yang tidak *corrupt*), regex ini menyimpan semua kombinasi nama yang mungkin, baik penulisan nama yang tidak benar maupun penulisan nama yang disingkat. Tiap regex ini nantinya disimpan ke suatu dictionary dengan nama asli sebagai value-nya dan regex dari nama tersebut sebagai key-nya. Jika nama yang diberikan memiliki kemiripan dengan regex, nama tersebut akan diganti dengan value yang dimiliki oleh regex tersebut.

3.2. Proses Penyelesaian Masalah dengan KMP dan BM

Berikut adalah proses penyelesaian dengan algoritma KMP dan BM:

1. Input sidik jari yang telah dikonversi ke bentuk ASCII nya akan dijadikan sebagai pattern/pola, sedangkan seluruh sampel sidik jari dalam database akan dijadikan sebagai text/string-nya.
2. Input sidik jari dalam ASCII akan dibentuk dulu tabel KMP/fungsi pinggirannya jika menggunakan KMP dan akan dibentuk fungsi kemunculan terakhir/Last Occurrence Function (LOF) jika menggunakan BM sebelum dilakukan perbandingan dengan tiap sampel pada *database*.
3. Kemudian, tiap sampel di database, dalam bentuk ASCII nya, akan dicek apakah ia memiliki substring yang mirip dengan input sidik jari sebelumnya.
4. Jika input sidik jari ada di database, longestMatch/kemiripan yang diberikan akan. Jika tidak, akan dikembalikan semua sampel yang memiliki longestMatch/kemiripan terpanjang yang terbesar.

3.3. Fitur Fungsional dari Aplikasi yang Dibangun

1. Pencocokan sidik jari masukan dengan database menggunakan algoritma Boyer-Moore
2. Pencocokan sidik jari masukan dengan database menggunakan algoritma Knuth-Morris-Pratt

3. Pencocokkan sidik jari masukan dengan database menggunakan Longest Common Subsequence

3.4. Ilustrasi Kasus

Kasus 1: Pengguna menggunakan algoritma KMP untuk mencari kemiripan sidik jari

1. Pengguna pertama-tama akan mengunggah citra sidik jari ke aplikasi.
2. Aplikasi kemudian akan mencari sidik jari yang mirip dengan sidik jari yang diberikan di database. Ini dilakukan dengan menggunakan algoritma string matching KMP.
3. Jika tidak ditemukan kecocokan yang tepat, aplikasi akan menggunakan algoritma LCS untuk menemukan sidik jari yang paling mirip.
4. Setelah sidik jari yang paling mirip ditemukan, aplikasi akan mencari biodata pemilik sidik jari tersebut dengan membandingkan nama pemilik sidik jari dengan data nama pada tabel biodata.
5. Jika data nama pada tabel biodata rusak, aplikasi akan melakukan rekonstruksi string dengan menggunakan regex untuk mengembalikan nama asli pemilik sidik jari.
6. Akhirnya, data dari pemilik sidik jari tersebut akan ditampilkan di layar, lengkap dengan waktu pencarian dan persentase kemiripan.

Kasus 2: Pengguna menggunakan algoritma BM untuk mencari kemiripan sidik jari

1. Pengguna pertama-tama akan mengunggah citra sidik jari ke aplikasi.
2. Aplikasi kemudian akan mencari sidik jari yang mirip dengan sidik jari yang diberikan di database. Ini dilakukan dengan menggunakan algoritma string matching BM.
3. Jika tidak ditemukan kecocokan yang tepat, aplikasi akan menggunakan algoritma LCS untuk menemukan sidik jari yang paling mirip.
4. Setelah sidik jari yang paling mirip ditemukan, aplikasi akan mencari biodata pemilik sidik jari tersebut dengan membandingkan nama pemilik sidik jari dengan data nama pada tabel biodata.
5. Jika data nama pada tabel biodata rusak, aplikasi akan melakukan rekonstruksi string dengan menggunakan regex untuk mengembalikan nama asli pemilik sidik jari.
6. Akhirnya, data dari pemilik sidik jari tersebut akan ditampilkan di layar, lengkap dengan waktu pencarian dan persentase kemiripan.

BAB 4

Implementasi dan Pengujian

4.1. Spesifikasi Program

Program dibuat dengan bahasa C# dan berjalan di sistem operasi Windows dengan versi dotnet 8.0.

Kelas Biodata

```
public partial class Biodata
{
    public string Nik = null;

    public string? Nama

    public string? TempatLahir

    public DateOnly? TanggalLahir

    public string? JenisKelamin

    public string? GolonganDarah

    public string? Alamat

    public string? Agama

    public string? StatusPerkawinan

    public string? Pekerjaan

    public string? Kewarganegaraan
}
```

Kelas SidikJari

```
public partial class SidikJari
{
    public string? BerkasCitra

    public string? Nama
}
```

Kelas KnuthMorrisPratt

```
1  namespace src.Algorithm;
2
3  public class KnuthMorrisPratt
4  {
5      private static int[] ComputeKMPTable(string pattern)
6      {
7          int[] lps = new int[pattern.Length];
8          int length = 0;
9          int i = 1;
10
11         lps[0] = 0;
12
13         while (i < pattern.Length)
14         {
15             if (pattern[i] == pattern[length])
16             {
17                 length++;
18                 lps[i] = length;
19                 i++;
20             }
21             else
22             {
23                 if (length != 0)
24                 {
25                     length = lps[length - 1];
26                 }
27                 else
28                 {
29                     lps[i] = 0;
30                     i++;
31                 }
32             }
33         }
34         return lps;
35     }
36
37     public static bool KMPSearch(string text, string pattern)
38     {
39         int[] lps = ComputeKMPTable(pattern);
40         int i = 0; // index for text[]
41         int j = 0; // index for pattern[]
42
43         while (i < text.Length)
44         {
45             if (pattern[j] == text[i])
46             {
47                 j++;
48                 i++;
49             }
50
51             if (j == pattern.Length)
52             {
53                 // Pattern found
54                 return true;
55             }
56             else if (i < text.Length && pattern[j] != text[i])
57             {
58                 if (j != 0)
59                 {
60                     j = lps[j - 1];
61                 }
62                 else
63                 {
64                     i++;
65                 }
66             }
67         }
68
69         // Pattern not found
70         return false;
71     }
72 }
73 }
```

Algoritma KMP akan melakukan pencocokan string dari depan dengan membuat fungsi batas/tabel KMP terlebih dahulu yang digunakan sebagai acuan untuk melakukan pencocokan kembali jika terjadi ketidakcocokan sehingga kita tidak perlu mengulang dari awal.

Kelas BoyerMoore



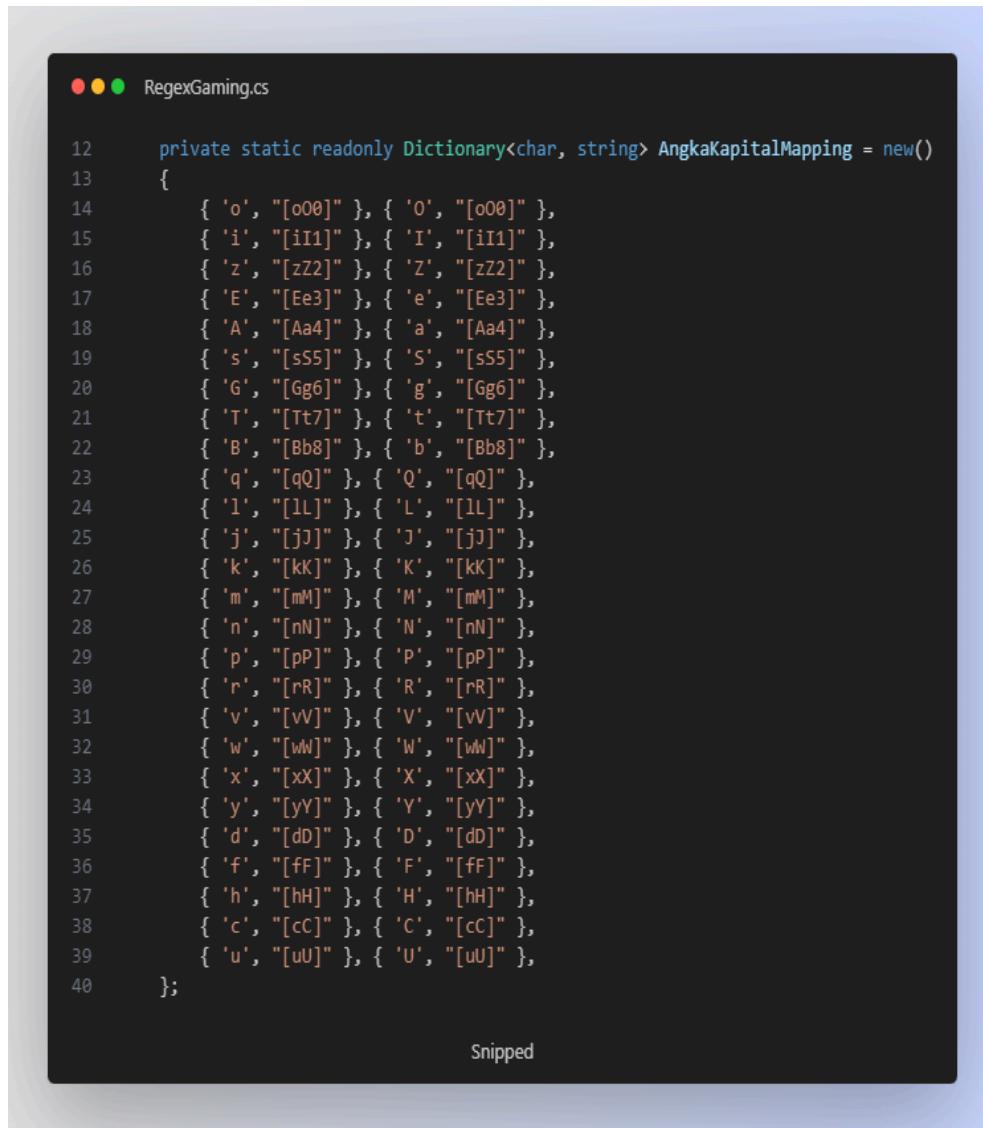
```
 1  namespace src.Algorithm;
 2
 3  public class BoyerMoore
 4  {
 5      private static Dictionary<char, int> BuildLastOccurrenceMap(string pattern)
 6      {
 7          Dictionary<char, int> lastOccurrence = [];
 8
 9          for (int i = 0; i < pattern.Length; i++)
10          {
11              lastOccurrence[pattern[i]] = i;
12          }
13
14          return lastOccurrence;
15      }
16
17      public static bool BMSearch(string text, string pattern)
18      {
19          if (string.IsNullOrEmpty(pattern) || string.IsNullOrEmpty(text)) return false;
20
21          Dictionary<char, int> lastOccurrence = BuildLastOccurrenceMap(pattern);
22          int m = pattern.Length;
23          int n = text.Length;
24          // s = shift of the pattern with respect to text
25          int s = 0;
26
27          while (s <= (n - m))
28          {
29              int j = m - 1;
30
31              // Compare the pattern with text from right to left
32              while (j >= 0 && pattern[j] == text[s + j])
33              {
34                  j--;
35              }
36
37              if (j < 0)
38              {
39                  // Pattern found
40                  return true;
41              }
42              else
43              {
44                  // Find the shift
45                  s += Math.Max(1, j - (lastOccurrence.TryGetValue(text[s + j], out int value) ? value : -1));
46              }
47
48          // Pattern not found
49          return false;
50      }
51  }
```

Snipped

Algoritma BM akan melakukan pencarian pola dari kanan ke kiri dan menggunakan last occurrence index. Fungsi BuildLastOccurrenceMap membuat peta yang menyimpan posisi kemunculan terakhir dari setiap karakter dalam pola, digunakan untuk menentukan seberapa jauh pola dapat digeser jika terjadi ketidakcocokan karakter selama pencarian. Fungsi BMSearch adalah fungsi utama yang melakukan pencarian pola dalam teks dengan membandingkan pola dari kanan ke kiri. Jika terjadi ketidakcocokan, algoritma menggunakan

peta kemunculan terakhir untuk menentukan seberapa jauh pola dapat digeser sehingga pencarian tidak perlu mengulang dari awal. Jika pola ditemukan, fungsi ini akan mengembalikan `true`; jika tidak, fungsi akan mengembalikan `false`.

Kelas RegexGaming



The screenshot shows a code editor window with a dark theme. The title bar says "RegexGaming.cs". The code itself is a C# class definition for a static readonly dictionary named "AngkaKapitalMapping". The dictionary maps lowercase letters to their corresponding uppercase counterparts. The code is heavily snipped, showing only lines 12 through 40. Lines 12-21 show mappings like 'o' to 'O', 'i' to 'I', 'z' to 'Z', etc. Lines 22-31 show mappings like 'A' to 'A', 's' to 'S', 'G' to 'G', 'T' to 'T', 'B' to 'B', 'q' to 'Q', 'l' to 'L', 'j' to 'J', 'k' to 'K', 'm' to 'M', 'n' to 'N', 'p' to 'P', 'r' to 'R', 'v' to 'V', 'w' to 'W', 'x' to 'X', 'y' to 'Y', 'd' to 'D', 'f' to 'F', 'h' to 'H', 'c' to 'C', and 'u' to 'U'. Line 32 ends with a closing brace for the class definition.

```
12     private static readonly Dictionary<char, string> AngkaKapitalMapping = new()
13     {
14         { 'o', "[oOθ]" }, { 'O', "[oOθ]" },
15         { 'i', "[iI1]" }, { 'I', "[iI1]" },
16         { 'z', "[zZ2]" }, { 'Z', "[zZ2]" },
17         { 'E', "[Ee3]" }, { 'e', "[Ee3]" },
18         { 'A', "[Aa4]" }, { 'a', "[Aa4]" },
19         { 's', "[sS5]" }, { 'S', "[sS5]" },
20         { 'G', "[Gg6]" }, { 'g', "[Gg6]" },
21         { 'T', "[Tt7]" }, { 't', "[Tt7]" },
22         { 'B', "[Bb8]" }, { 'b', "[Bb8]" },
23         { 'q', "[qQ1]" }, { 'Q', "[qQ]" },
24         { 'l', "[lL]" }, { 'L', "[lL]" },
25         { 'j', "[jJ]" }, { 'J', "[jJ]" },
26         { 'k', "[kK]" }, { 'K', "[kK]" },
27         { 'm', "[mM]" }, { 'M', "[mM]" },
28         { 'n', "[nN]" }, { 'N', "[nN]" },
29         { 'p', "[pP]" }, { 'P', "[pP]" },
30         { 'r', "[rR]" }, { 'R', "[rR]" },
31         { 'v', "[vV]" }, { 'V', "[vV]" },
32         { 'w', "[wW]" }, { 'W', "[wW]" },
33         { 'x', "[xX]" }, { 'X', "[xX]" },
34         { 'y', "[yY]" }, { 'Y', "[yY]" },
35         { 'd', "[dD]" }, { 'D', "[dD]" },
36         { 'f', "[fF]" }, { 'F', "[fF]" },
37         { 'h', "[hH]" }, { 'H', "[hH]" },
38         { 'c', "[cC]" }, { 'C', "[cC]" },
39         { 'u', "[uU]" }, { 'U', "[uU]" },
40     };

```

Snipped

```

  ● ● ● RegExGaming.cs

private static readonly HashSet<char> Vowels = new() { 'a', 'i', 'u', 'e', 'o', 'A', 'I', 'U', 'E', 'O' };

public static string GenerateRegexPattern(string original)
{
    var methods = new[] { "orisinil", "angka-besar-kecil", "singkat" };
    var regexPatterns = methods.Select(method => method switch
    {
        "orisinil" -> GenerateOrisinilPattern(original),
        "angka-besar-kecil" -> GenerateAngkaAndBesarKecilPattern(original),
        "singkat" -> GenerateSingkatPattern(original),
        _ -> original
    }).ToList();
}

return string.Join("|", regexPatterns);
}

public static string FixAlayWord(string originalText, string alayText)
{
    // Split the original text into words
    var originalWords = originalText.Split(' ');

    // Create a dictionary to store the regex patterns and their corresponding original words
    var regexToOriginalWord = new Dictionary<string, string>();

    foreach (var word in originalWords)
    {
        // Generate regex pattern for each word (tiap kata ada regexnya masing-masing)
        string regexPattern = GenerateRegexPattern(word);
        // Console.WriteLine($"Kata: {word}, Regex: {regexPattern}");
        regexToOriginalWord[regexPattern] = word;
    }

    // Replace each alay word with the original word
    string[] alayWords = alayText.Split(' ');
    // Create a list to store the corrected words
    List<string> correctedWords = new List<string>();

    foreach (var alayWord in alayWords)
    {
        string correctedWord = alayWord;

        foreach (var pair in regexToOriginalWord)
        {
            string regexPattern = pair.Key;
            string originalWord = pair.Value;

            // Compile the regex pattern, pakai IgnoreCase biar gak peduli huruf besar kecil
            Regex alayRegex = new Regex(regexPattern, RegexOptions.IgnoreCase);

            // Check if the alay word matches the regex pattern
            if (alayRegex.IsMatch(alayWord))
            {
                // Replace the alay word with the original word
                correctedWord = originalWord;
                break; // Stop checking other patterns once a match is found
            }
        }

        correctedWords.Add(correctedWord);
    }

    // Join the corrected words back into a single string
    string correctedText = string.Join(" ", correctedWords);
}

return correctedText;
}

private static string GenerateOrisinilPattern(string original)
{
    return Regex.Escape(original); // Escape all special characters
}

private static string GenerateAngkaAndBesarKecilPattern(string original)
{
    return string.Concat(original.Select(c => AngkaKapitalMapping.ContainsKey(c) ? AngkaKapitalMapping[c] : Regex.Escape(c.ToString())));
}

private static string GenerateSingkatPattern(string original)
{
    var words = original.Split(' ');
    var result = new StringBuilder();

    foreach (var word in words)
    {
        if (word.Length > 0)
        {
            // Tambahkan huruf pertama dan cek di AngkaKapitalMapping
            if (AngkaKapitalMapping.ContainsKey(word[0]))
            {
                result.Append(AngkaKapitalMapping[word[0]]);
            }
            else
            {
                result.Append($"{char.ToLower(word[0])}{char.ToUpper(word[0])}");
            }

            // Tambahkan rincian huruf dengan cek di AngkaKapitalMapping dan mengabaikan vokal
            foreach (var c in word.Skip(1))
            {
                if (Vowels.Contains(c))
                {
                    // Jika karakter adalah vokal, jadi opional
                    if (AngkaKapitalMapping.ContainsKey(c))
                    {
                        result.Append($"{(AngkaKapitalMapping[c])?}");
                    }
                    else
                    {
                        result.Append($"{char.ToLower(c)}{char.ToUpper(c)}");
                    }
                }
                else
                {
                    if (AngkaKapitalMapping.ContainsKey(c))
                    {
                        result.Append(AngkaKapitalMapping[c]);
                    }
                    else
                    {
                        result.Append($"{char.ToLower(c)}{char.ToUpper(c)}");
                    }
                }
            }
        }
        // Tambahkan regex untuk mengabaikan nol atau lebih spasi
        result.Append(@"\s*");
    }
}

return result.ToString().Trim();
}

```

Snipped

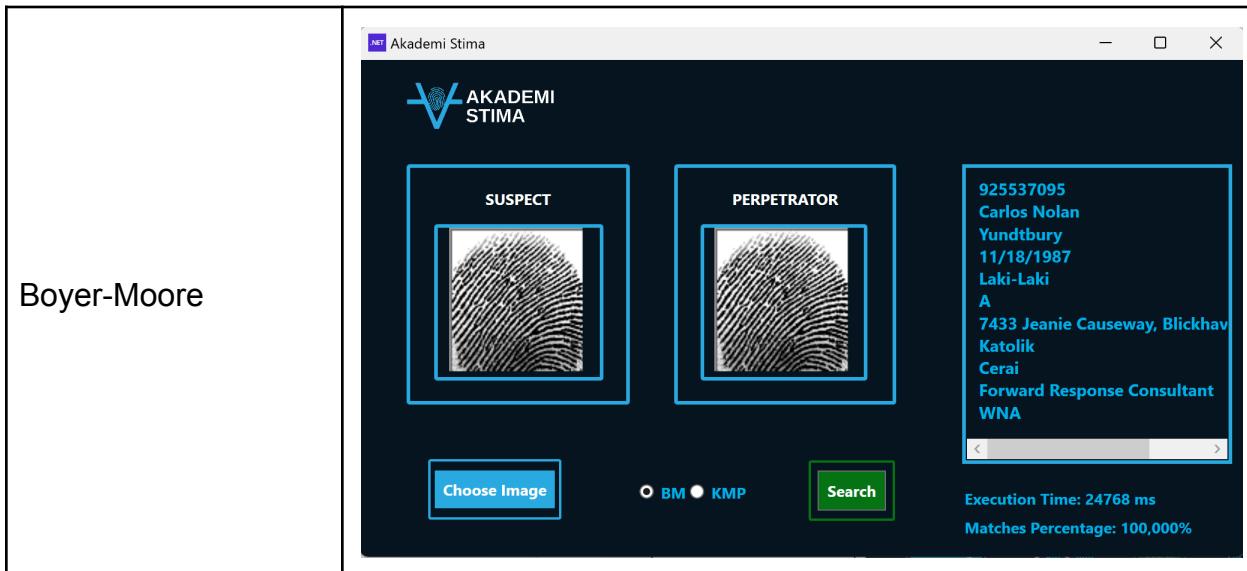
Kelas RegexGaming berisi algoritma untuk memperbaiki data yang *corrupt* (nama alay), sebelumnya telah disediakan sebuah dictionary yang nantinya digunakan sebagai patokan dalam pembuatan regex. Kemudian, program akan menerima input nama asli dan nama alay, program akan membuat regex per-kata berdasarkan dari nama asli yang diberikan dan regex ini disimpan dalam dictionary dengan regex sebagai key dan kata asli sebagai valuenya. Program kemudian mengecek apakah nama alay memiliki kemiripan dengan regex, jika ada, kata tersebut diganti dengan value dari regex tersebut, yaitu kata asli yang tidak terkena “virus” alay.

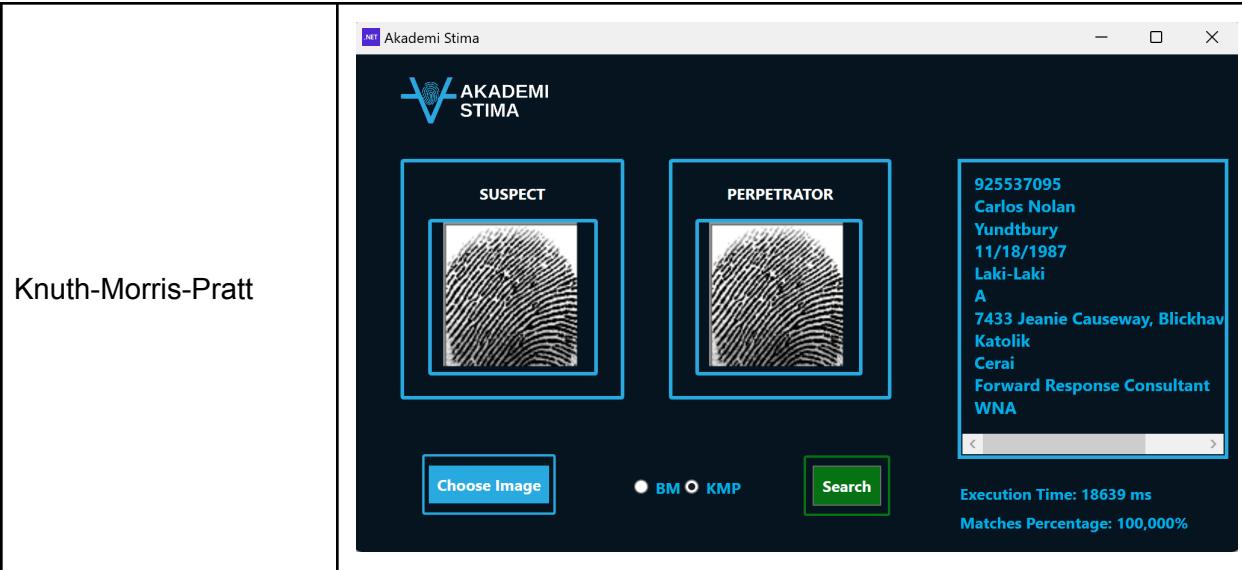
4.2. Tata Cara Penggunaan Program

1. Klik tombol “Choose Image” untuk mengunggah sidik jari yang ingin dicocokkan.
2. Pilih algoritma pencarian, yaitu BM (Boyer-Moore) atau KMP (Knuth-Morris-Pratt)
3. Klik tombol “Search” untuk memulai pencarian.
4. Tunggu beberapa saat hingga hasil pencarian keluar.

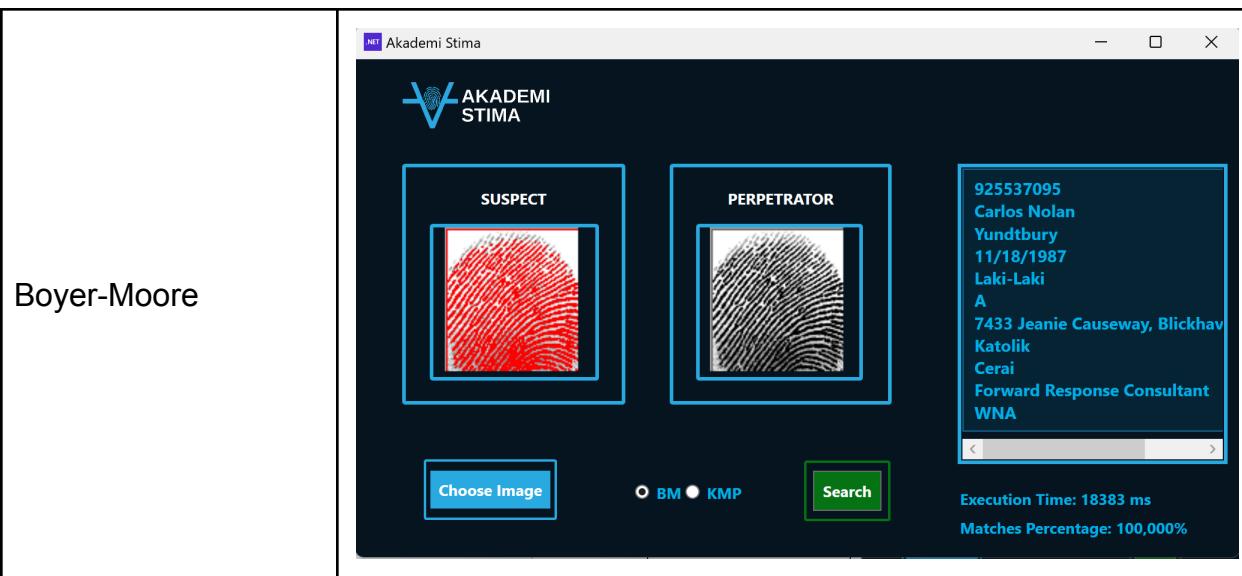
4.3. Hasil Pengujian

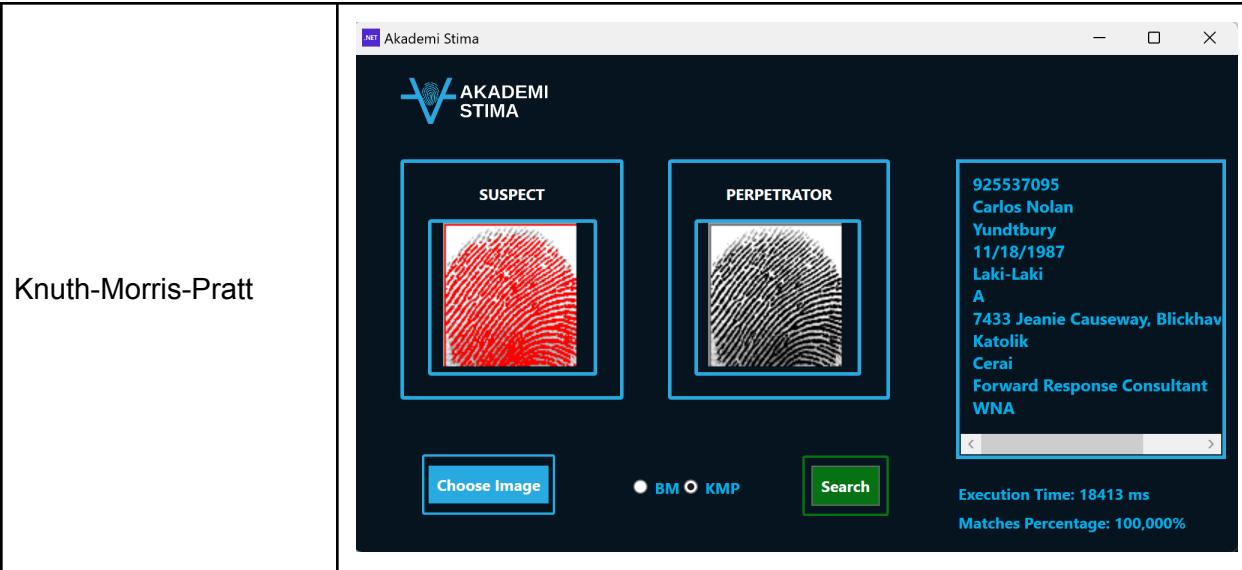
1. Sidik Jari Ibu Jari Penuh (8__M_Left_thumb_finger.BMP)



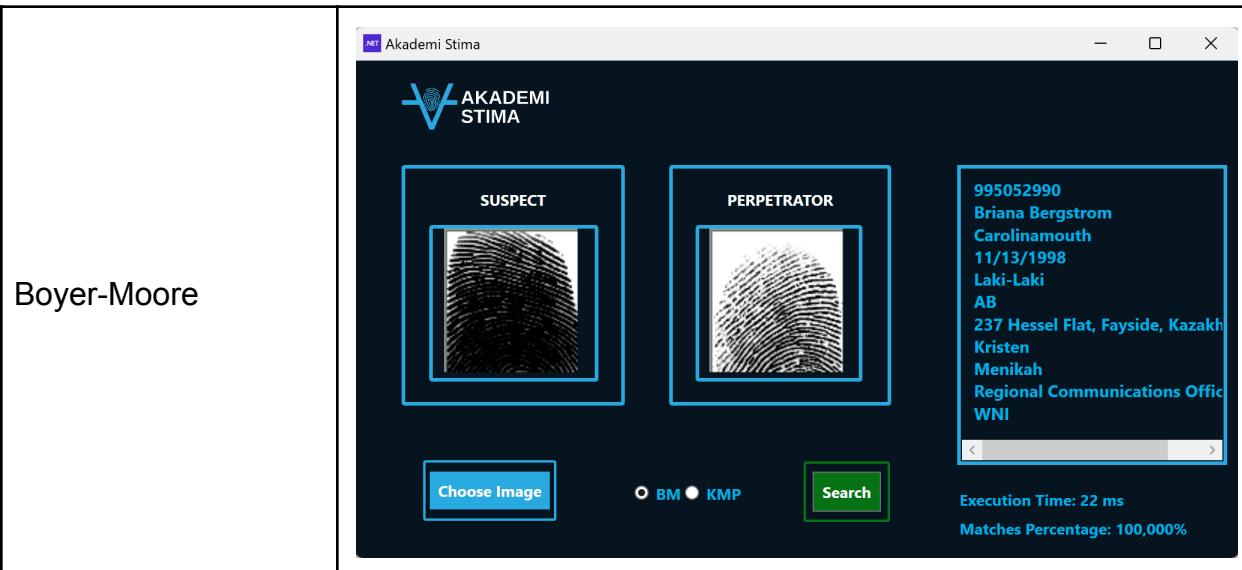


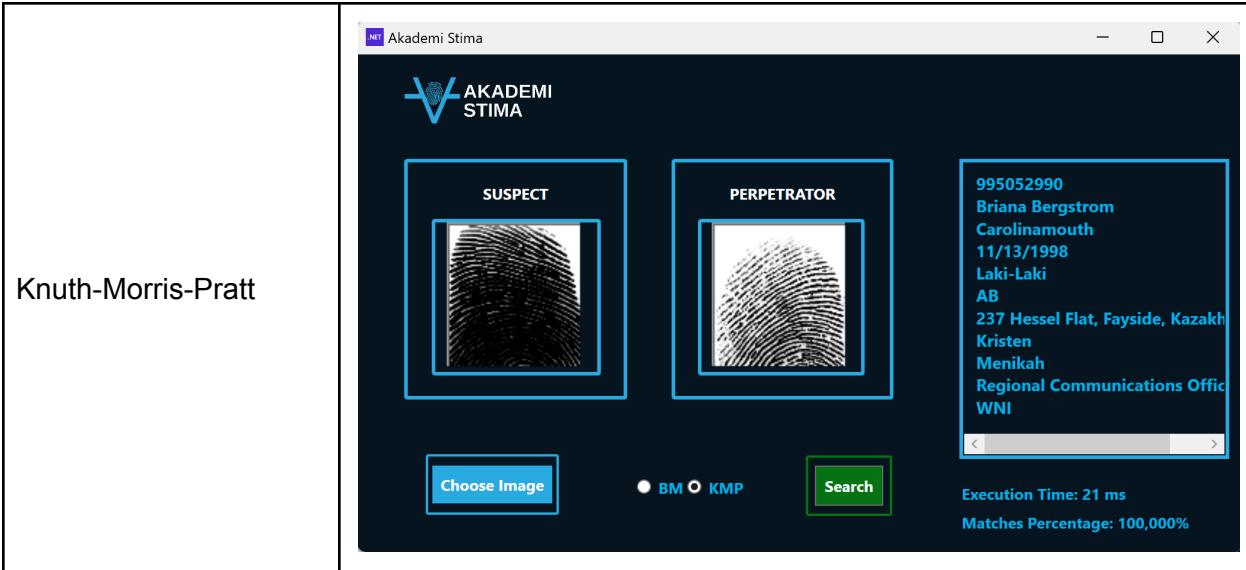
2. Sidik Jari Ibu Jari Penuh Berwarna (modifikasi 8__M_Left_thumb_finger.BMP)



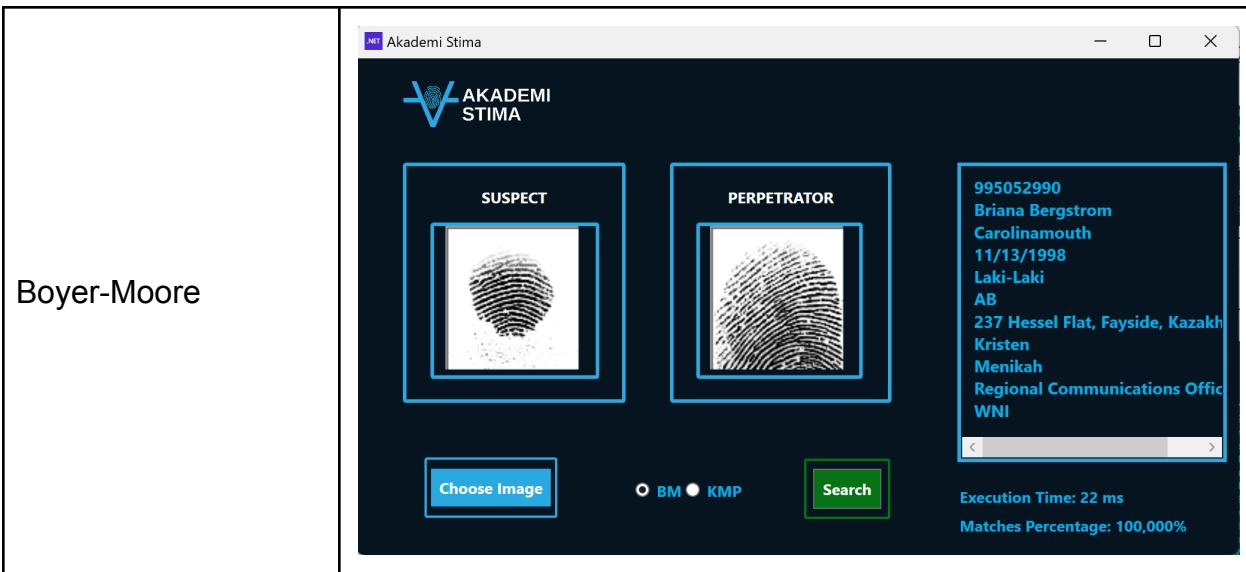


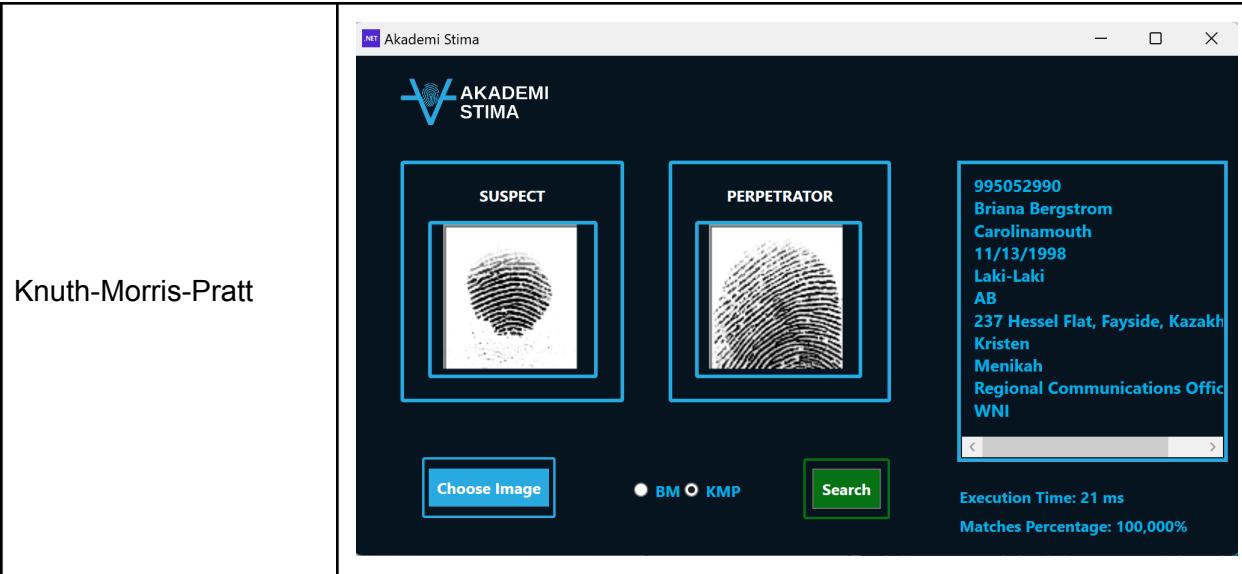
3. Sidik Jari Ibu Jari Penuh Sangat Hitam (16__M_Right_thumb_finger.BMP)



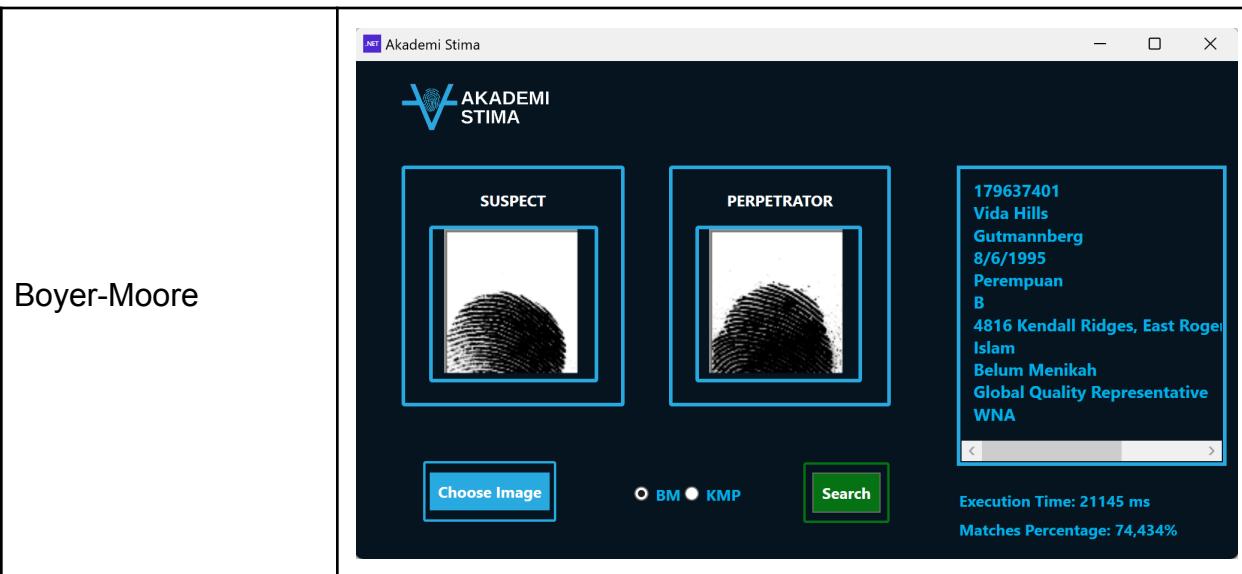


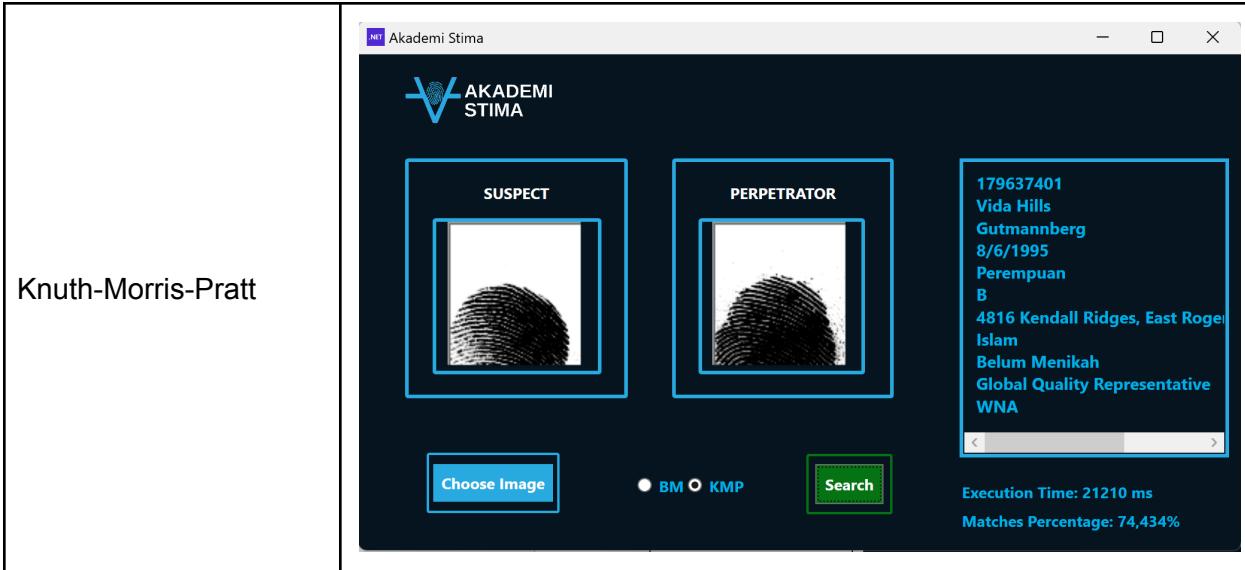
4. Sidik Jari Ibu Jari Sebagian (470__F_Right_little_finger.BMP)



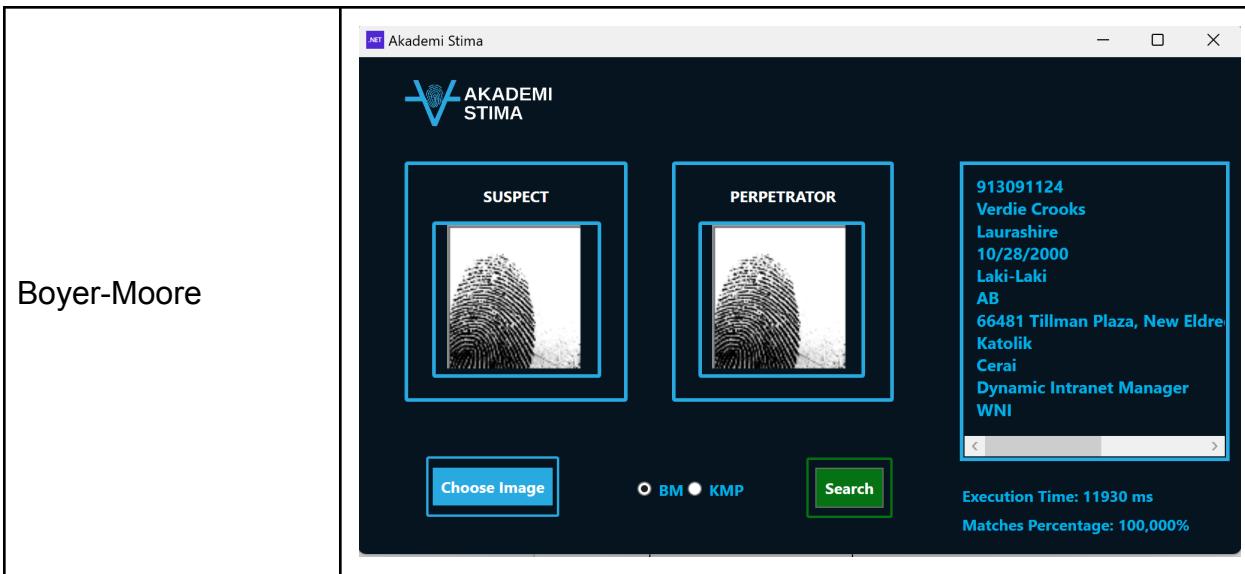


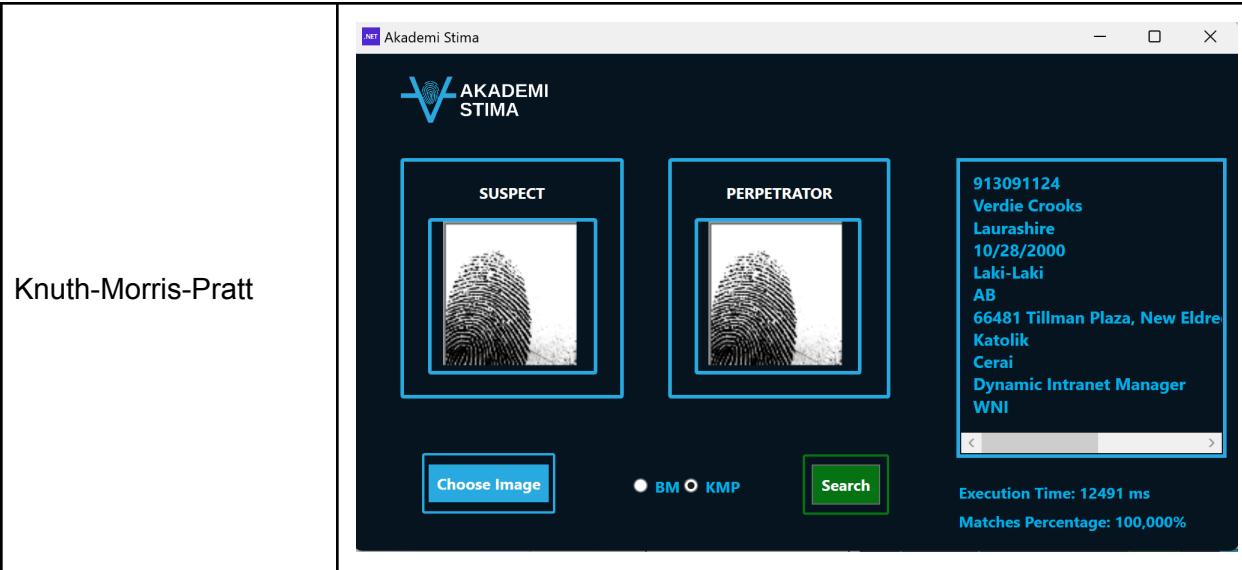
5. Sidik Jari Ibu Jari Sedikit ke Bawah (22__M_Right_ring_finger.BMP)



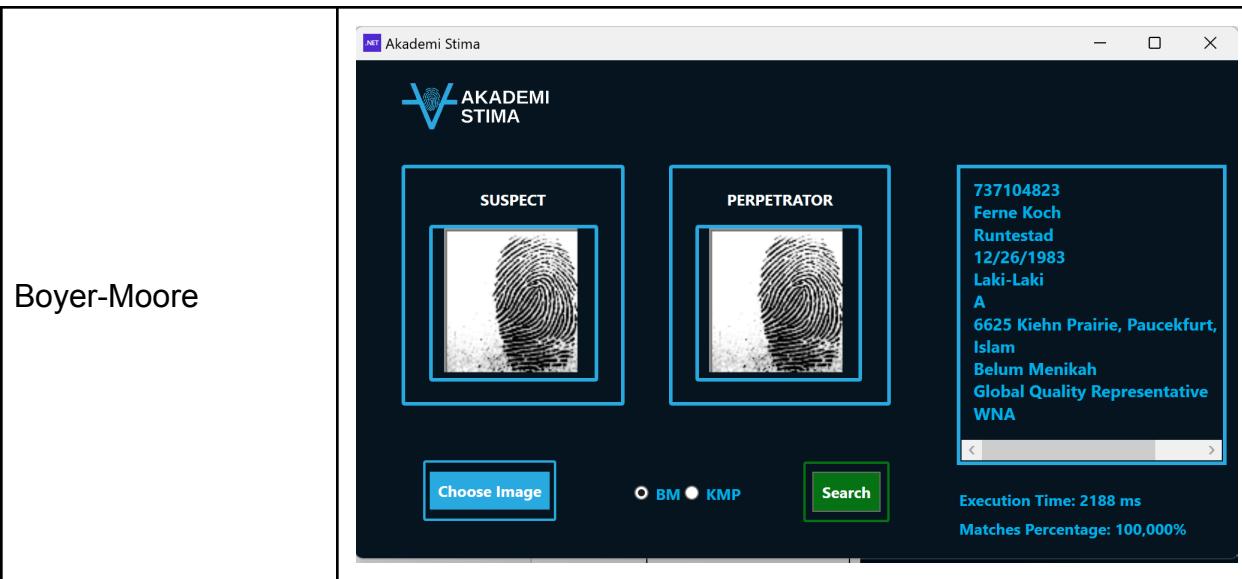


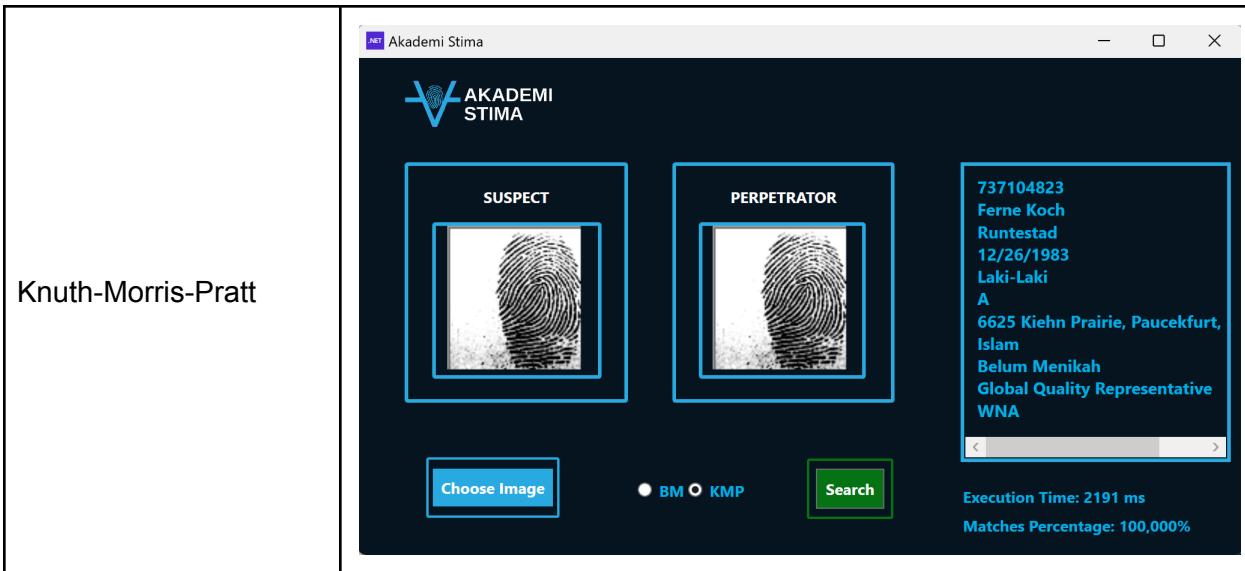
6. Sidik Jari Ibu Jari Sedikit ke Kiri (600__M_Left_little_finger.BMP)





7. Sidik Jari Ibu Jari Sedikit ke Kanan (560_F_Left_index_finger.BMP)





4.4. Analisa Hasil Pengujian

Hasil pengujian yang telah dilakukan memberikan beberapa informasi mengenai algoritma yang digunakan dalam pencarian sidik jari ini. Hasil pengujian dapat dilihat dari beberapa aspek, diantaranya akurasi dan waktu pencarian.

Akurasi kedua algoritma, KMP dan BM, dapat dikatakan mirip atau bahkan sama. Algoritma yang digunakan, yaitu KMP dan BM, pada dasarnya mencari kecocokan antara sebuah pola dengan sebuah string. Jika ditemukan pola yang sama pada string tersebut maka kedua algoritma tersebut akan mengembalikan nilai true dan begitu juga sebaliknya. Dapat disimpulkan bahwa kedua algoritma ini memiliki tingkat akurasi yang sama.

Berdasarkan hasil pengujian, waktu yang dibutuhkan dalam mencari pola dalam sebuah string oleh kedua algoritma ini relatif sama. Berdasarkan teori, performa KMP akan menurun jika ukuran alfabet semakin besar, sebaliknya performa BM akan menurun jika ukuran alfabet semakin kecil. Pada implementasi program ini, kami menggunakan karakter 256-bit untuk melakukan pengecekan, yang berarti ada 256 karakter yang mungkin untuk diperiksa. Untuk ukuran karakter sebanyak ini, performa untuk kedua algoritma relatif sama, seperti yang terlihat dari hasil pengujian yang menunjukkan waktu pencarian yang tidak jauh berbeda.

Jika kedua algoritma sebelumnya tidak menemukan pola yang cocok dalam citra yang sedang dicari, pencarian akan dilanjutkan dengan algoritma LCS. Akurasi dari LCS dikategorikan baik, di mana hasil persentase kemiripan sesuai dengan kemiripan kedua citra. Dalam implementasi ini, pengecekan menggunakan algoritma LCS dilakukan secara bersamaan dengan pengecekan menggunakan algoritma KMP atau BM, sehingga hasil dari LCS akan muncul setelah pencarian seluruh database selesai. Hal ini mengakibatkan pencarian dengan

menggunakan LCS memerlukan waktu paling lama di antara ketiga algoritma tersebut walaupun waktu pencarian pada satu citra mungkin sama.

BAB 5

Kesimpulan dan Saran

5.1. Kesimpulan

Melalui tugas besar ini, kami menerapkan konsep algoritma pattern matching yang dipelajari dari kuliah IF2211 Strategi Algoritma dalam menentukan kecocokan sidik jari. Proyek ini memberikan pemahaman yang lebih mendalam mengenai algoritma string matching, seperti KMP dan Boyer-Moore, serta kemampuan mereka dalam mencocokkan pola secara efisien. Selain itu, kami juga mempelajari implementasi algoritma Longest Common Subsequence (LCS) untuk mengukur kemiripan sidik jari. Selain itu kami juga mengeksplorasi mengenai pembuatan aplikasi GUI dengan menggunakan dotnet dan c#. Pembuatan aplikasi ini juga memanfaatkan pemahaman kami mengenai OOP yang telah diajarkan di mata kuliah Pemrograman Berbasis Objek.

5.2. Saran

Proses pencarian sidik jari dapat dipercepat dengan menggunakan multithreading, yang memungkinkan pencarian dilakukan secara paralel, sehingga mempercepat keseluruhan proses pencocokan sidik jari dengan database. Selain itu, harus lebih diperhatikan proses pengambilan sampel citra agar bandingan lebih akurat.

Lampiran

Link *repository* github : [SandWithCheese/Tubes3_AkademiStima \(github.com\)](https://github.com/SandWithCheese/Tubes3_AkademiStima)

Link video youtube : [Pattern Matching dalam Menentukan Kecocokan Sidik Jari | Tugas Besar 3 Strategi Algoritma 2024 - Akademi Stima](https://www.youtube.com/watch?v=Kecocokan%20Sidik%20Jari%20-%20Tugas%20Besar%203%20Strategi%20Algoritma%202024%20-%20Akademi%20Stima)

Daftar Pustaka

Munir, Rinaldi. "Pencocokan String." Institut Teknologi Bandung, 2020-2021.

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

Munir, Rinaldi. "Pencocokan String dengan Regular Expression." Institut Teknologi Bandung, 2022-2023.

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

GeeksforGeeks - Longest Common Subsequence

[Longest Common Subsequence \(LCS\) - GeeksforGeeks](#)