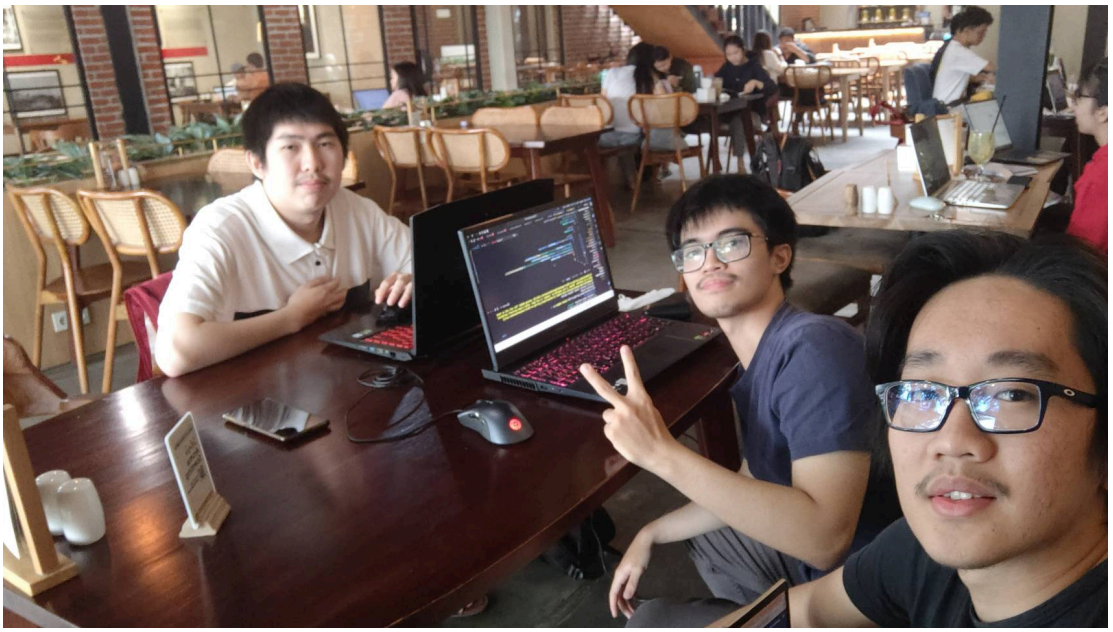
Laporan Tugas Besar 3 IF2211 Strategi Algoritma
Semester II Tahun Akademik 2023/2024

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



Disusun Oleh:

Vanson Kurnialim	13522049
Fedrianz Dharma	13522090
Andhika Tantyo Anugrah	13522094

Sekolah Teknik Elektro dan Informatika

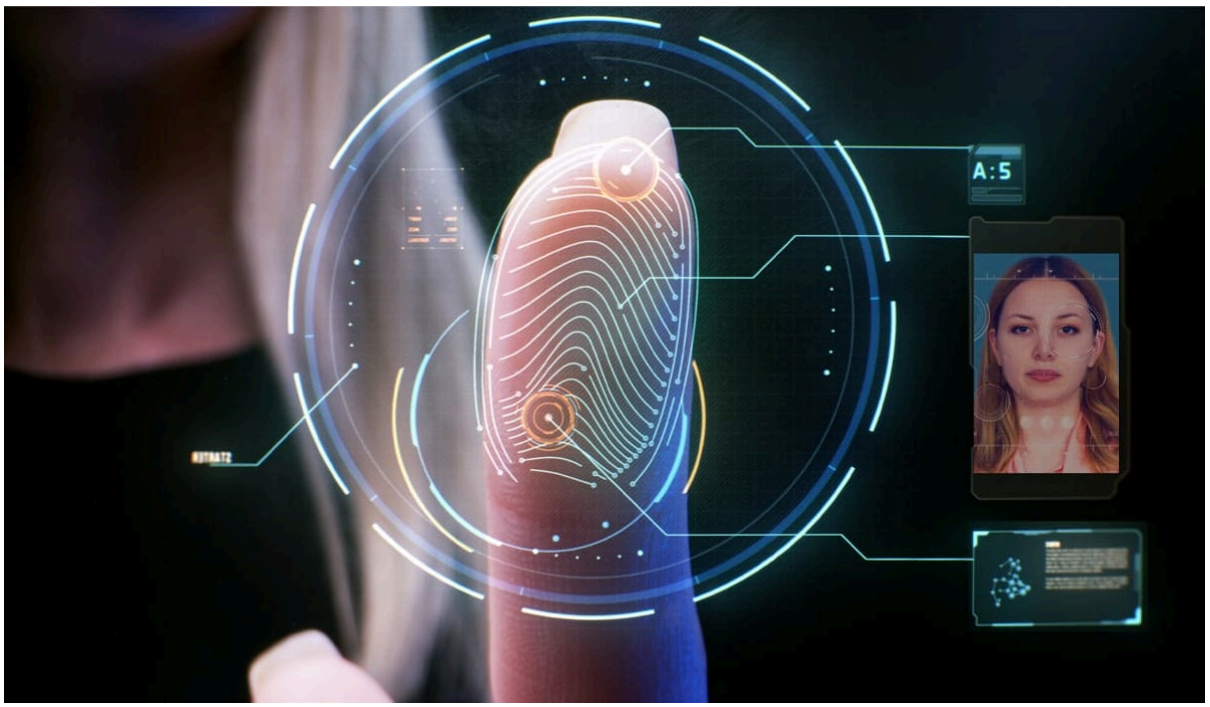
Institut Teknologi Bandung

2024

DAFTAR ISI

DAFTAR ISI..... 2
BAB I
DESKRIPSI TUGAS..... 3
BAB II
LANDASAN TEORI..... 5
A. Algoritma..... 5
B. Persentase Kemiripan..... 5
C. Aplikasi Desktop..... 6
BAB III
ANALISIS PEMECAHAN MASALAH..... 7
A. Langkah-Langkah Pemecahan Masalah..... 7
B. Penyelesaian Solusi dengan Algoritma KMP dan BM..... 8
C. Fitur Fungsional dan Arsitektur Aplikasi Desktop..... 9
D. Ilustrasi Kasus..... 9
BAB IV
IMPLEMENTASI DAN PENGUJIAN..... 11
A. Backend..... 11
B. Frontend..... 18
C. Docker MySQL..... 19
D. Tata Cara Penggunaan Program..... 19
E. Hasil Pengujian..... 19
F. Analisis Hasil Pengujian..... 21
BAB V
KESIMPULAN DAN SARAN..... 23
LAMPIRAN..... 24
Bukti Penggunaan Kakas AI..... 24
Pranala GitHub..... 25
Pranala Video YouTube..... 25

BAB I
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 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 II
LANDASAN TEORI

A. Algoritma

Algoritma Knuth-Morris-Pratt (KMP), merupakan metode pencarian kata atau *string matching* yang efisien untuk menemukan suatu substring (pola) di dalam string dengan variasi karakter sedikit. KMP menghindari pencocokan karakter yang sudah diketahui akan gagal, sehingga mengurangi jumlah perbandingan yang diperlukan. Memanfaatkan properti dari prefiks dan sufiks dari pola yang sedang dicari, algoritma ini dapat menentukan banyak karakter yang dilewati menggunakan *border function*.

Algoritma Boyer-Moore (BM), merupakan metode pencarian kata atau *string matching* yang efisien dengan jumlah variasi karakter cukup banyak, misalnya pada bahasa Inggris. Algoritma ini menggunakan fungsi heuristic karakter baik dan fungsi heuristic karakter buruk. Ketika terjadi ketidakcocokan karakter, algoritma Boyer-Moore menggunakan informasi dari karakter yang tidak cocok untuk menentukan seberapa jauh *pattern* dapat digeser ke kanan. Jika terjadi ketidakcocokan setelah beberapa karakter cocok, pola dapat digeser ke kanan berdasarkan bagian dari pola yang sudah cocok. Algoritma ini melakukan pergeseran yang memastikan bahwa sufiks yang cocok terus digunakan untuk pencocokan berikutnya.

Regular expression merupakan kumpulan karakter berurutan yang membentuk suatu pola pencarian. Setiap karakter pada pola pencarian merepresentasikan suatu arti atau fungsi. Salah satu contoh penggunaan *regex* adalah karakter '.' yang melambangkan *any* karakter. Maka untuk karakter yang dilambangkan '.' pada pola pencarian, karakter apapun akan dianggap benar pada teks. Dengan mengetahui pola pencarian, maka seluruh kemungkinan *string* yang mungkin dari pola pencarian dapat ditemukan.

B. Persentase Kemiripan

Algoritma KMP dan BM hanya berguna untuk menentukan apakah sebuah *string* mengandung keseluruhan *pattern* atau memiliki 100% *pattern* di dalamnya. Pada kenyataanya, pencocokan sidik jari sangat jarang mendapatkan kecocokan 100%, walaupun menggunakan *pattern* atau sebagian dari satu sidik jari. Maka diperlukan algoritma lain untuk menghitung persentase kemiripan sebuah *pattern* dengan sebuah *string* untuk menentukan sidik jari mana yang paling mirip. Digunakan algoritma Longest Common Subsequence yang bekerja dengan menghitung banyak karakter *pattern* yang sama dengan karakter *string* sesuai

dengan urutan. Banyak karakter yang sama ini lalu diubah menjadi persentase berdasarkan panjang *pattern* awal.

C. Aplikasi Desktop

Aplikasi desktop dibuat dengan membaginya menjadi 2 komponen, yaitu *frontend* dan *backend*. *Graphical User Interface* atau GUI aplikasi desktop (bagian *frontend*) dibangun dengan kakas WinForm yang disediakan oleh .NET menggunakan bahasa pemrograman C#. Proses komputasi, pembandingan, dan pengaksesan basis data semua dilakukan di bagian *backend* juga dengan bahasa pemrograman C#.

BAB III
ANALISIS PEMECAHAN MASALAH

A. Langkah-Langkah Pemecahan Masalah

Secara garis besar, aplikasi desktop yang kita bangun, akan menerima sebuah citra sidik jari dari pengguna. Aplikasi akan mencocokkan citra *input* dari pengguna dengan keseluruhan citra pada basis data untuk mencari citra yang paling sesuai jika ada. Informasi terkait citra tersebut akan ditampilkan pada layar.

Seluruh proses pencarian dibagi menjadi beberapa langkah sebagai berikut :

1. Konversi citra sidik jari

   Untuk memanfaatkan algoritma *string matching*, diperlukan format informasi citra sidik jari dalam bentuk *string*. Oleh karena itu, citra sidik jari perlu dikonversi menjadi informasi yang sesuai. Tahap pertama, diambil data *grayscale* dalam biner untuk tiap *pixel* dari citra sidik jari, 1 berarti hitam dan 0 berarti putih. Setelah didapatkan matrix biner tersebut, tiap baris pada matrix akan diubah menjadi kumpulan karakter yang membentuk suatu *string*. Perubahan dilakukan dengan cara mengambil 7 biner di bawah biner target dan menggabungkannya menjadi sebuah karakter 8 bit yang akan diubah menjadi representasinya dalam ASCII. Dengan demikian, di dalam satu baris terdapat kumpulan karakter yang akan diubah menjadi sebuah *string*.

   Tata cara konversi di atas digunakan untuk citra sidik jari pada basis data. Citra sidik jari *input* dari pengguna memiliki pengaturan perubahan yang berbeda karena tidak semua *pixel* akan diubah menjadi *string* melainkan *pattern* atau sebagian kecil saja. Setelah citra *input* diproses seperti citra sidik jari pada basis data, ukuran *pixel* yang kami gunakan untuk menjadi *pattern* adalah 30 pixel atau 30 karakter ASCII saja karena kami rasa sudah cukup panjang namun tidak terlalu pendek untuk merepresentasikan sebuah citra sidik jari. Keputusan ini diambil dengan konsiderasi dataset citra sidik jari yang digunakan berukuran ± 100x100 *pixel*. 30 *pixel* yang digunakan berada pada baris tengah bawah matrik dan juga berada pada *range* kolom paling tengah.

2. Pencocokkan citra

   Aplikasi menyediakan 2 algoritma pencocokkan citra yang dapat dipilih pengguna, yaitu algoritma *string matching* KMP atau BM. Algoritma yang dipilih akan mencocokkan *string pattern* dari input dengan tiap baris *string* dari setiap citra yang

ada di database hingga ditemukan. Jika ditemukan yang citra dengan kecocokan 100%, program akan langsung berhenti dan mengembalikan hasilnya.

3. Persentase kemiripan

   Selama pencarian dengan menggunakan algoritma KMP atau BM, dilakukan juga perhitungan dengan menggunakan algoritma LCS untuk mencari tingkat kemiripan antara *pattern* dengan *string* dari citra pada database. Jika saat proses pencocokkan citra menggunakan algoritma KMP dan BM tidak menghasilkan hasil yang *match*, maka persentase terbesar hasil LCS akan digunakan.

4. Pengambilan informasi biodata

   Setelah mendapatkan nama pemilik sidik jari berdasarkan citra yang *matched* atau yang paling mendekati sesuai dengan persentase kemiripan, akan dicari informasi biodata dari pemilik citra. Praktisnya, kita hanya tinggal melakukan *query* pada basis data yang memiliki nama yang sama dengan nama yang kita sudah dapatkan. Namun pada implementasi aplikasi ini, atribut nama pada basis data yang digunakan bukanlah nama asli yang sesuai, namun telah mengalami perubahan atau *corrupted*. Sehingga akan dimanfaatkan konsep *regular expression* (*regex)* untuk mencocokkan nama pada biodata. Nama asli pada citra akan diubah menjadi sebuah pola pencarian menggunakan *regex*, barulah akan dicocokkan dengan nama pada basis data. Segala kemungkinan perubahan pada nama telah diperhitungkan oleh pola pencarian hasil *regex* sehingga jika memang ada data yang sesuai dalam basis data, informasi biodata pasti bisa didapatkan.

B. Penyelesaian Solusi dengan Algoritma KMP dan BM

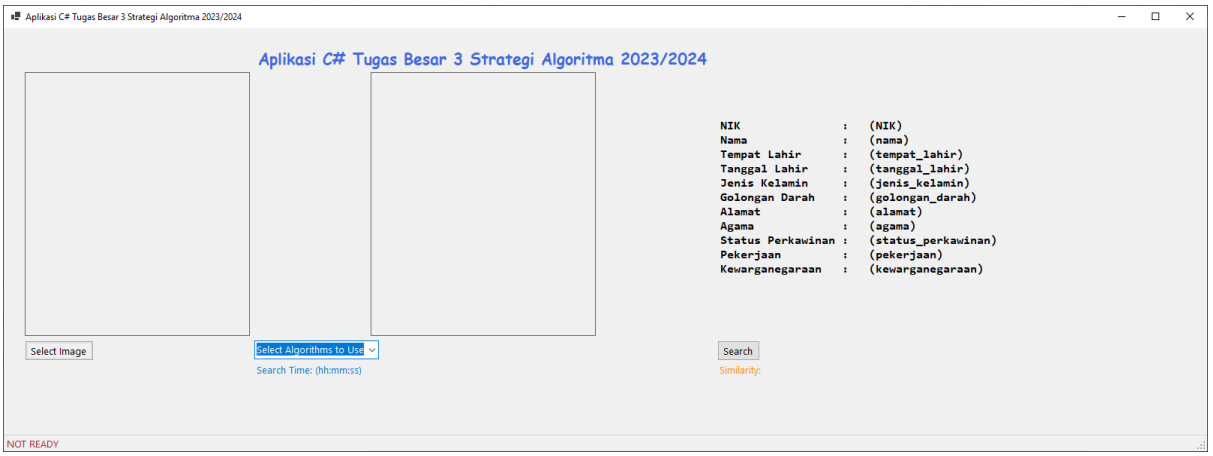
Algoritma *string matching* Knuth-Morris-Pratt (KMP) memeriksa *string* citra pada basis data dengan *pattern* hasil dari *regex* mulai dari karakter kiri menuju kanan. Jika karakter yang diperiksa *match*, algoritma akan memeriksa karakter selanjutnya sampai akhir *string*. Jika tidak, akan dicari panjang *prefix* yang merupakan *suffix* dari *substring* kumpulan karakter *pattern* yang telah diperiksa. Lalu pattern akan digeser satu kali ke kanan beserta tambahan pergeseran sejumlah panjang *prefix* jika ada. Pemeriksaan selanjutnya akan dimulai dari karakter setelah *prefix*. Perhitungan panjang *prefix* memanfaatkan *border function*.

Algoritma *string matching* Boyer-Moore (BM) berbeda dengan KMP. Algoritma ini memulai pemeriksaan dari karakter paling kanan *pattern* menuju kiri. Jika karakter yang diperiksa *match*, maka akan dilanjutkan dengan memeriksa karakter sebelah kirinya. Jika tidak, akan dicari karakter pertama di sebelah kiri sisa *pattern,* yang *match* dengan karakter

pada *string* yang menyebabkan tidak *match,* lalu menggeser *pattern* hingga kedua huruf bertepatan. Jika *pattern* memiliki karakter tersebut pada bagian yang sudah diperiksa atau di sebelah kanan, geser *pattern* satu kali. Jika *pattern* tidak mengandung karakter tersebut, geser seluruh *pattern* hingga karakter awal berada di setelah huruf yang tidak *match* pada *string*.

Kedua implementasi algoritma di atas dilakukan untuk tiap baris pada matrix. Kondisi *matched* didapatkan jika salah satu baris *match* dan kondisi tidak *match* didapatkan jika seluruh baris tidak *match*, maka pencarian akan dilanjutkan ke matrix citra selanjutnya hingga ditemukan atau data habis.

C. Fitur Fungsional dan Arsitektur Aplikasi Desktop



Gambar 3.1. Tampilan Aplikasi

Seperti yang bisa dilihat pada gambar di atas ini, aplikasi hanya memiliki sedikit fungsionalitas atau tombol yang dapat digunakan. Tombol ‘Select image’ berfungsi untuk mengunggah berkas citra dari desktop pengguna. Tombol *drop-down* bertuliskan ‘Select Algorithm’ akan memberikan pilihan algoritma yang dapat dipilih yaitu KMP dan BM. Tombol ‘Search’ akan memproses seluruh informasi yang diberikan pengguna jika sudah lengkap dan menampilkan hasilnya pada layar di sebelah kanan sesuai dengan bagian yang telah tersedia. Untuk mengantisipasi informasi yang panjang, terdapat fitur *scroll* pada laman aplikasi yang dapat digunakan oleh pengguna.

Dikarenakan aplikasi *desktop* dibangun menggunakan kakas WinForm, aplikasi hanya dapat dijalankan pada platform *desktop* atau platform berbasis arsitektur x86.

D. Ilustrasi Kasus

Pada basis data terdapat informasi berkas citra dan atribut nama “Budi” pada tabel sidik_jari dan informasi berikut ('1', 'bd bd1man', 'KABUPATEN PURWOREJO' ,'1994-03-20', 'Perempuan', 'AB', 'Jl. Dahlia 19', 'Buddha', 'Cerai', 'Software Consultant',

'Indonesia') pada tabel biodata. Selain data yang disebutkan terdapat pula data lainnya namun tidak relevan pada ilustrasi kasus ini.

Pengguna memasukkan sebuah berkas citra milik Budi pada aplikasi lalu menekan tombol *search*. Aplikasi akan langsung memproses berkas citra *input* dan membandingkannya dengan seluruh berkas citra pada basis data. Hasil pencocokkan akan mendapatkan nama asli pemilik citra yaitu “Budi”. Nama tersebut akan dibuat menjadi pola pencarian menggunakan *regex*. Setelah dilakukan iterasi seluruh basis data, didapatkan nama *corrupted* yaitu “bd bd1man” dengan informasi biodata lainnya.

berkas citra yang cocok dengan berkas citra *input* akan ditampilkan pada layar beserta informasi biodata pemilik citra.

BAB IV
IMPLEMENTASI DAN PENGUJIAN

Program dibagi menjadi 2 projek, backend dan frontend. Pada bagian backend terdapat fungsi-fungsi yang digunakan untuk menyelesaikan masalah yang berhubungan *string matching* dan database. Pada bagian frontend terdapat kode-kode yang digunakan untuk membuat GUI yang dibutuhkan.

A. Backend

Pada bagian backend terdapat 2 bagian, yaitu folder algorithm yang berisi algoritma dan kode, serta folder database yang berisi file sql dari database.

1. KMP.cs

   File KMP.cs berisi kode implementasi dari algoritma KMP yang digunakan untuk melakukan *string matching*.

```
public static bool KMPSolve(string source, string subset) {
    int count = 0 ;
    StringBuilder src = new(source) ;
    StringBuilder sub = new(subset) ;
    int len_src = src.Length ;
    int len_sub = sub.Length ;

    if (len_src <= len_sub) {
        return false ;
    }

    int t_index = 0 ;
    int size = 0 ;
    while (t_index <= len_src-len_sub) {
        // Console.WriteLine("a") ;
        int p_index ;
        for (p_index = size ; p_index < len_sub ; p_index++) {
            if (p_index == len_sub - 1 && sub[p_index] == src[t_index+p_index]) {
                count += 1 ;
                return true ;
            }
            else {
                count += 1 ;
                if (sub[p_index] == src[t_index+p_index]) {
                    continue ;
                }
                else {
                    int border = BorderFunction(sub.ToString(0, p_index)) ;
                    if (p_index == 0) {
                        t_index += 1 ;
                    }
                    else {
                        t_index += p_index - border ;
                        size = border ;
                    }
                    break ;
                }
            }
        }
    }
    return false ;
}
```

Fungsi KMPSolve menerima 2 argumen, yaitu string source dan string subset. Fungsi ini akan mengembalikan true apabila string source mengandung string subset.

```
static int BorderFunction(string word) {
    if (word.Length <= 1) {
        return 0 ;
    }
    else {
        StringBuilder prefix = new(word) ;
        prefix.Remove(prefix.Length - 1 , 1) ;
        StringBuilder sufix = new(word) ;
        sufix.Remove(0,1) ;
        while (prefix.Length > 0) {
            if (prefix.Equals(sufix)) {
                return prefix.Length ;
            }
            else {
                prefix.Remove(prefix.Length - 1 , 1) ;
                sufix.Remove(0,1) ;
            }
        }
        return 0 ;
    }
}
```

Fungsi BorderFunction digunakan untuk mendapatkan nilai dari border function dari suatu *string pattern*.

2. BM.cs

File BM.cs berisi kode implementasi dari algoritma BM yang digunakan untuk melakukan *string matching*.

```
public static Dictionary<char, int> BuildLast(string pattern)
{
    var last = new Dictionary<char, int>();
    for (int i = 0; i < pattern.Length; i++)
    {
        last[pattern[i]] = pattern.Length - i - 1;
    }
    last[pattern[pattern.Length-1]] = pattern.Length;

    return last;
}
```

Fungsi BuildLast digunakan untuk mendapatkan Dictionary yang berisi index terakhir kemunculan tiap karakter pada *string pattern*.

```
public static int BMmatch(string text, string pattern)
{
    Dictionary<char, int> last = BuildLast(pattern);
    if (pattern.Length > text.Length)
    {
        return -1;
    }
    int i = pattern.Length - 1; // text index
    int j = pattern.Length - 1; // pattern index
    int posMarker = 0;
    do
    {

        if (pattern[j] == text[i])
        {
            if (j == 0)
            {
                return i;
            }
            else
            {
                i--;
                j--;
            }
        }
        else
        {
            if (!last.TryGetValue(text[i], out int l))
            {
                l = pattern.Length;
            }
            posMarker += l;
            i = posMarker + pattern.Length - 1;
            j = pattern.Length - 1;
        }
    } while (i <= text.Length - 1);
    return -1;
}
```

Fungsi BMmatch akan mengembalikan index dari lokasi kemunculan *string pattern* pada *string text*. Fungsi akan mengembalikan -1 bila tidak ditemukan *pattern* pada *text*.

3. Converter.cs

File Converter.cs berisi algoritma untuk mengubah citra sidik jari menjadi List yang berisi ASCII.

```
static int[,] ConvertToBnW(Bitmap bitmap)
{
    int[,] matrix = new int[bitmap.Height, bitmap.Width];
    for (int y = 0; y < bitmap.Height; y++)
    {
        for (int x = 0; x < bitmap.Width; x++)
        {
            Color originalColor = bitmap.GetPixel(x, y);
            int grayScale = (int)(originalColor.R * 0.21 + originalColor.G * 0.72 + originalColor.B * 0.07);
            int newColor = (grayScale >= 128) ? 0 : 1;
            matrix[y, x] = newColor;
        }
    }
    return matrix;
}
```

Fungsi ConvertToBnW akan mengubah gambar input menjadi grayscale dan mengembalikan matrix yang berisi angka 1 atau 0. Nilai grayscale diatas 127 akan diubah menjadi angka 0 dan dibawah 128 menjadi angka 1.

```
static List<string> ConvertToAscii(int[,] matrix)
{
    List<string> arrStr = [];
    StringBuilder sb = new StringBuilder();
    int val = 0;
    for (int row = 0; row < matrix.GetLength(0) - 7; row++)
    {
        sb.Clear();
        for (int col = 0; col < matrix.GetLength(1); col++)
        {
            val = 0;
            for (int itr = 0; itr < 8; itr++)
            {
                val = val << 1 | matrix[row + itr, col];
            }
            sb.Append((char)val);
        }
        arrStr.Add(sb.ToString());
    }
    return arrStr;
}
```

Fungsi ConvertToAscii akan mengubah matrix yang berisi 0 atau 1 menjadi List of string yang merupakan representasi ASCII dari 8 bit biner pada matrix. 8 bit biner tersebut didapatkan dengan melakukan looping pada tiap baris dan mengambil nilai pada tiap kolom dan 7 baris dibawahnya.

```
1 reference
public static string getPattern(List<string> arr)
{

    int midpoint = arr.Count * 2 / 3;
    string Original = arr.ElementAt(midpoint);
    // StringBuilder sb = new StringBuilder();
    int midpoint2 = (Original.Length - 30) / 2;

    return Original.Substring(midpoint2, 30);

}
```

Fungsi getPattern digunakan untuk mengambil string sampel ASCII dari citra sidik jadi yang sudah di convert menjadi List of string.

4. Database.cs

```
public static (string NIK, string nama, string tempat_lahir, string tanggal_lahir,
string jenis_kelamin, string golongan_darah, string alamat, string agama, string
status_perkawinan, string pekerjaan, string kewarganegaraan) SearchDatabaseWithName
(string name)
```

Fungsi SeachDatabaseWithName digunakan untuk mencari database berdasarkan nama yang diberikan. Fungsi ini akan mengembalikan semua biodata yang didapat.

```
public static void EncryptDatabaseBiodata()
```

Prosedur EncryptDatabaseBiodata digunakan untuk melakukan enkripsi pada tabel biodata di database.

```
public static void EncryptDatabaseSidikJari() ...
```

Prosedur EncryptDatabaseSidikJari digunakan untuk melakukan enkripsi pada tabel sidik_jari di database.

```
public static List<KeyValuePair<string, string>> ReadDatabaseSidikJari() ...
```

Fungsi ReadDatabaseSidikJari digunakan untuk mendapatkan semua data dari tabel sidik_jari pada database.

```
public static List<string> ReadDatabaseName() ···
```

Fungsi ReadDatabaseName digunakan untuk mendapatkan semua nama yang ada di database.

```
public static (List<string> NIK, List<string> nama, List<string> tempat_lahir,
List<string> tanggal_lahir, List<string> jenis_kelamin, List<string> golongan_darah,
List<string> alamat, List<string> agama, List<string> status_perkawinan, List<string>
pekerjaan, List<string> kewarganegaraan) ReadDatabaseBiodataAll() ···
```

Fungsi ReadDatabaseBiodataAll digunakan untuk mendapatkan semua data di tabel biodata pada database.

5. Encryption.cs

```
8 references
public static string Encrypt(string plaintext, string keyV, int keyC) {···

14 references
public static string Decrypt(string ciphertext, string keyV, int keyC) {···
```

Fungsi Encrypt dan Decrypt digunakan untuk melakukan enkripsi dan dekripsi pada suatu string. Enkripsi yang dilakukan adalah gabungan dari metode Vigenere dan Caesar dengan masing-masing *key* sesuai parameter. Dekripsi yang dilakukan merupakan kebalikan dari fungsi enkripsi yaitu melakukan dekripsi metode Caesar dengan *key* yang dibalik dilanjutkan dengan dekripsi metode Vigenere dengan *key* yang sama.

6. Regex.cs

```
2 references
public static string createPattern(string word) {
    StringBuilder src = new StringBuilder(word.ToLower()) ;
    StringBuilder pattern = new StringBuilder() ;
    for (int i = 0 ; i < src.Length ; i++) {
        if (RegularExpressions.formula.ContainsKey(src[i])) {
            pattern.Append(RegularExpressions.formula[src[i]]) ;
        }
        else {
            pattern.Append(src[i]) ;
        }
    }
    return pattern.ToString() ;
}
```

Fungsi createPattern digunakan untuk membuat suatu *pattern* regex dari masukan string.

```
2 references
public static bool regexMatching(string word, string pattern) {
    return Regex.IsMatch(word, pattern) ;
}
```

Fungsi regexMatching digunakan untuk memeriksa apakah sebuah *string* cocok dengan pola pencarian *pattern* yang telah dibuat.

7. Gabungan.cs

```
public static (string path, string NameAsli, string NamaCorrupt, float percent)
getPathAndName(string pathPattern, int algo) ...
```

Fungsi getPathAndName digunakan untuk mendapatkan path citra pada database, nama asli, nama yang *corrupt*, dan persentase kemiripan dari suatu citra sidik jari yang menjadi masukan.

8. LCS.cs

```
1 reference
public static int LongestCommonSubsequence(string pattern, string subset) ...
```

Fungsi LongestCommonSubsequence digunakan untuk mencari Longest Common Subsequence dari sebuah pattern pada sebuah pattern.

B. Frontend

Frontend aplikasi diimplementasikan di dalam file *MainForm.Designer.cs*. File ini berisi kelas parsial *MainForm*. Kelas ini merupakan pusat dari semua komponen-komponen pada *MainForm*. Kelas ini memiliki atribut-atribut bertipe *TableLayoutPanel*, *Label*, *PictureBox*, *Button*, *ComboBox*, *ToolStripStatusLabel*, *StatusStrip*, dan *tuple* yang berisi tipe data *string* dan *int*. Atribut-atribut ini merupakan komponen-komponen yang sebagian besar terlihat pada tampilan. Kelas ini juga memiliki beberapa fungsi seperti *LoadImage* yang berfungsi untuk menampilkan gambar, *SelectImageButton_Click* untuk menampilkan prompt untuk mengunggah file, *algorithm_SelectedIndexChanged* untuk mengubah pilihan algoritma yang akan digunakan, *StartSearch* untuk memanggil fungsi pada *backend*, dan fungsi *ShowResult* untuk menampilkan hasil luaran dari *backend*.

C. Docker MySQL

Docker digunakan untuk menampung *database* MySQL yang digunakan agar user id dan password yang digunakan untuk connect pada server database sama. Selain itu docker digunakan agar pengguna yang tidak mempunyai MySQL tetap dapat menjalankan programnya.

D. Tata Cara Penggunaan Program

1. **Source**

   Pertama, aktifkan terlebih dahulu docker dengan menggunakan docker-compose up-d agar MySQL dapat digunakan. Setelah docker aktif, jalankan dotnet run --project src/frontend untuk menjalankan aplikasi dengan GUI. Upload citra sidik jari yang ingin digunakan. Pilih algoritma yang ingin digunakan, tersedia algoritma BM dan algoritma KMP. Klik tombol search dan tunggu hasil.

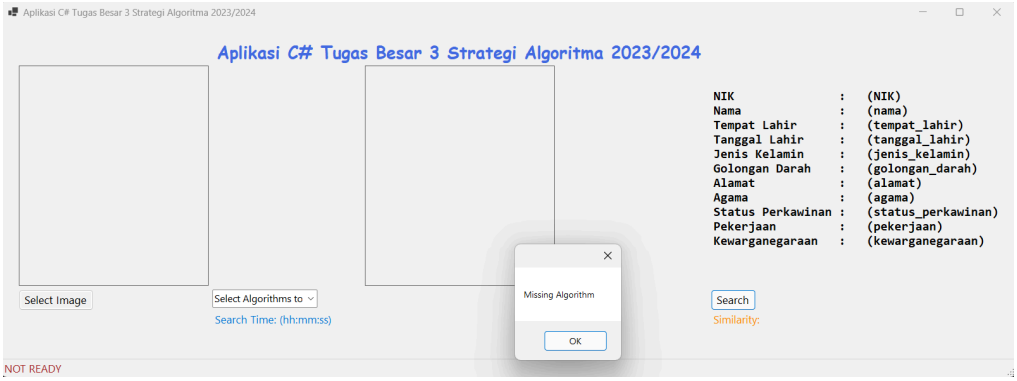
2. **Binary**

   Pada *root folder repository* execute frontend.exe dengan: bin/frontend/net8.0-windows/frontend.exe, kemudian upload citra sidik jari yang ingin dicari biodatanya. Pada tahap ini, silahkan pilih algoritma yang ingin digunakan dan klik tombol search dan tunggu hasil.

E. Hasil Pengujian

1. Pengujian 1

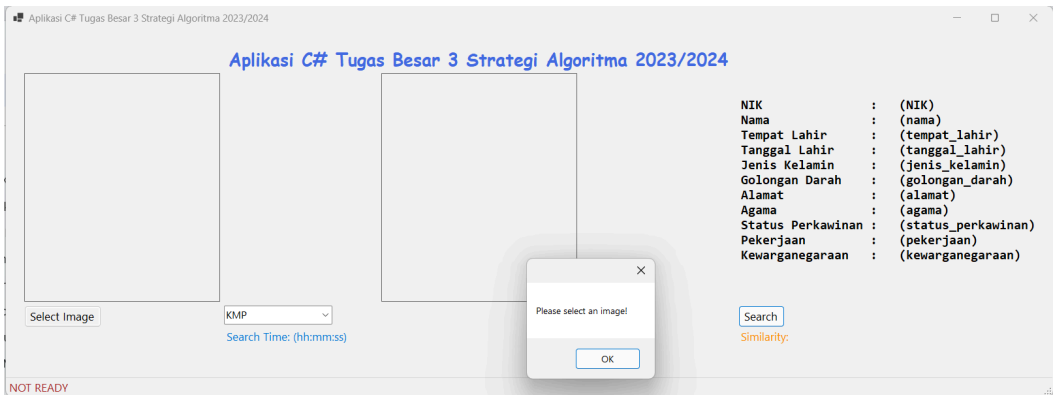
   Berikut adalah tampilan aplikasi ketika menekan tombol 'Search' tanpa mengisi berkas citra dan pilihan algoritma :



Gambar 4.1. Tampilan aplikasi pengujian 1

2. Pengujian 2

Berikut adalah tampilan aplikasi ketika menekan tombol ‘Search’ tanpa mengisi berkas citra :



Gambar 4.2. Tampilan aplikasi pengujian 2

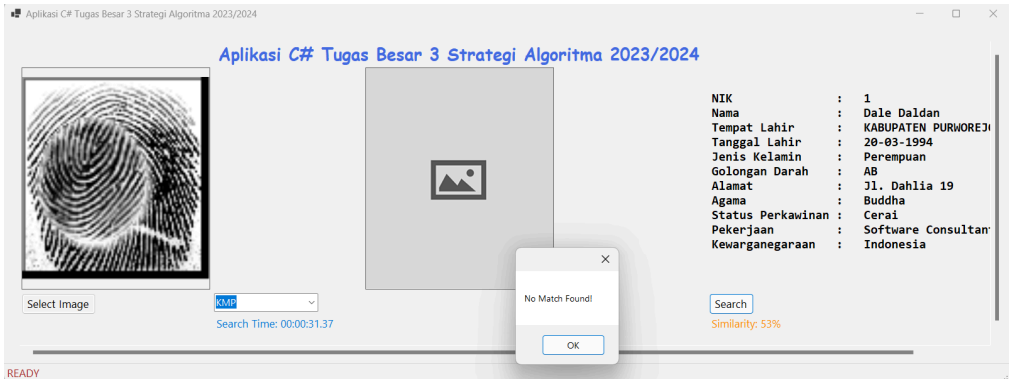
3. Pengujian 3

   Berikut adalah tampilan aplikasi ketika berkas citra unggahan berada dalam basis data aplikasi :

Gambar 4.3. Tampilan aplikasi pengujian 3

4. Pengujian 4

   Berikut adalah tampilan aplikasi ketika berkas citra unggahan tidak berada dalam batas kemiripan dalam basis data aplikasi :



Gambar 4.4. Tampilan aplikasi pengujian 4

Similarity tetap ditampilkan untuk menunjukkan similarity yang paling besar yang ditemukan namun kurang dari batas.

5. Pengujian 5

   Berikut adalah tampilan aplikasi ketika berkas citra unggahan merupakan berkas citra dalam basis data yang telah dilakukan perubahan :

Gambar 4.5. Tampilan aplikasi pengujian 5

F. Analisis Hasil Pengujian

Berdasarkan pengujian 1 dan pengujian 2, aplikasi tidak akan melanjutkan proses jika seluruh informasi yang diperlukan belum dipenuhi oleh pengguna. Pada pengujian 3, hasil yang didapat sesuai dengan ekspektasi dengan *similarity* 100% karena berkas citra memang sama dengan berkas citra pada basis data. Jika saat pencarian ditemukan berkas dengan tingkat *similarity* 100%, maka pencarian akan dihentikan dan langsung menampilkan hasil.

Pada pengujian 4, unggahan berkas citra merupakan citra pada basis data yang telah dimodifikasi secara berat sehingga hanya memiliki kemiripan sebesar 54%. Oleh karena batas kemiripan yang kami tentukan sebesar 60%, hasil pencarian tidak ditampilkan pada layar karena dianggap tidak akurat.

Berbeda dengan hasil pada pengujian 5, hasil modifikasi citra masih berada dalam tingkat kemiripan sehingga hasil pencarian dapat ditampilkan. Aplikasi yang kami buat masih tidak dapat mengatasi perbedaan sidik jari yang terlalu besar. Dikarenakan area pengambilan *pattern* dan metode pencocokkan data yang digunakan, teknik yang kami gunakan tidak sesuai untuk membuat sebuah metode pencocokkan sidik jari yang andal yang telah mengalami modifikasi yang berat. Pada

praktik nyata, dengan citra sidik jari yang bervariasi dan tidak terkontrol, hasil yang didapatkan tidak dapat diprediksi menyebabkan aplikasi ini *unreliable*.

Walaupun demikian, aplikasi ini sangatlah cocok untuk pengetesan pencocokkan citra yang terkontrol dan sebagai metode pembelajaran terkait *string matching* dan aspek pemrosesan lainnya. Hal yang dapat dikembangkan untuk memperbesar akurasi mungkin adalah metode penentuan *pattern* ataupun metode pencocokkan.

BAB V
KESIMPULAN DAN SARAN

Dari tugas besar ketiga IF2211 Strategi Algoritma ini, kami berhasil mengimplementasikan sistem yang dapat mengidentifikasi individu berbasis biometrik dengan menggunakan sidik jari menggunakan metode Boyer-Moore dan Knuth-Morris-Pratt. Sistem ini juga berhasil dihubungkan dengan identitas sebuah individu melalui basis data sehingga menjadi sistem yang dapat mengenali identitas seseorang secara lengkap hanya dengan menggunakan sidik jari.

Saran untuk diri kami sendiri adalah mempercantik GUI menambahkan *multi-threading* agar GUI tidak dalam keadaan *not responding* saat *frontend* menunggu hasil dari *backend* sekaligus memangkas waktu yang dibutuhkan oleh *backend* jika *backend* juga memanfaatkan *multi-threading*.

LAMPIRAN

Bukti Penggunaan Kakas AI

c# read database and put it inside list of pair

To read data from a database in C# and store it inside a list of pairs (using the `KeyValuePair` type), you can follow these steps:

1. Set up your database connection.
2. Execute a query to retrieve data.
3. Read the data and store it in a list of `KeyValuePair`.

Below is an example demonstrating how to achieve this using a SQL Server database. The example assumes you have a table called `ExampleTable` with columns `KeyColumn` and `ValueColumn`.

First, you'll need to include the necessary namespaces:

csharp Copy code

```csharp
using System;
using System.Collections.Generic;
using System.Data.SqlClient;
```

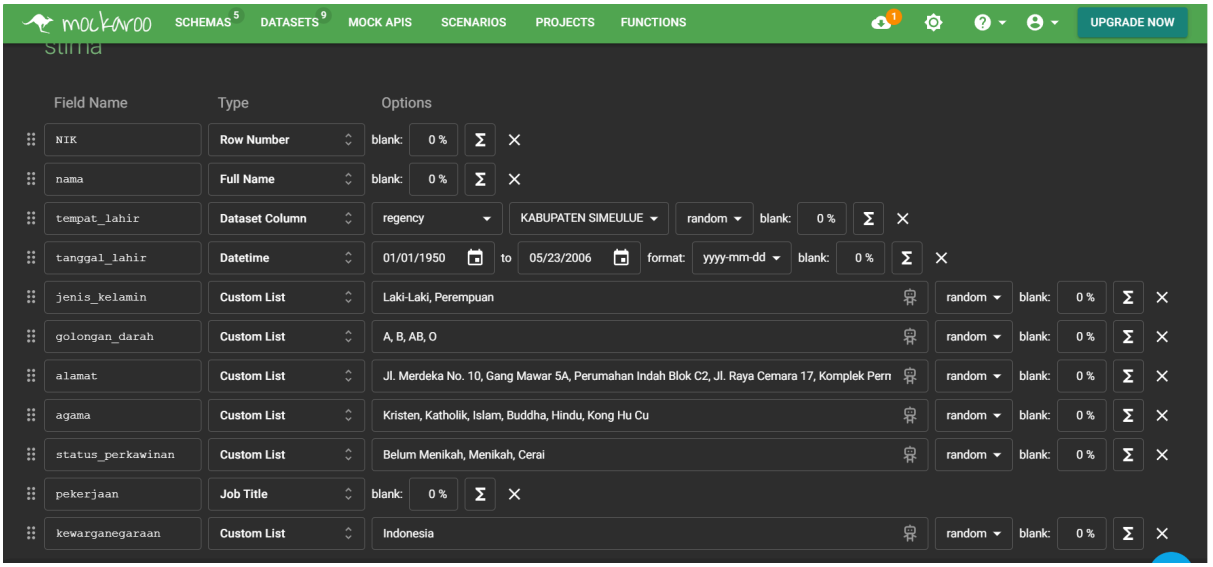
Here's the complete code to read from the database and store the data in a list of `KeyValuePair<string, string>`:

csharp Copy code

```csharp
public class DatabaseReader
{
    private string connectionString;

    public DatabaseReader(string connectionString)
    {
        this.connectionString = connectionString;
    }

    public List<KeyValuePair<string, string>> ReadData()
    {
        var dataList = new List<KeyValuePair<string, string>>();

        using (SqlConnection connection = new SqlConnection(connectionString))
        {
            connection.Open();

            string query = "SELECT KeyColumn, ValueColumn FROM ExampleTable";
            using (SqlCommand command = new SqlCommand(query, connection))
            {
                using (SqlDataReader reader = command.ExecuteReader())
                {
                    while (reader.Read())
                    {
```

Penggunaan ChatGPT untuk menghubungkan C# dengan database MySQL

Mockaroo



Mockaroo digunakan untuk *seeding* database

Pranala GitHub

Kode dapat diakses pada repository GitHub:

https://github.com/FedrianzD/Tubes3_Last-Day-on-ITB

Pranala Video YouTube

Video dapat diakses pada platform video YouTube:

https://youtu.be/vmCpb-Dik7k

Jika terdapat masalah, dapat menghubungi ID Line: vansonk

DAFTAR PUSTAKA

1. Spesifikasi Tugas Besar 3 Stima 2023/2024.docx - Google Docs
2. C# language reference | Microsoft Learn
3. https://alaygenerator.blogspot.com/
4. https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Pencocokan-string-2021.pdf