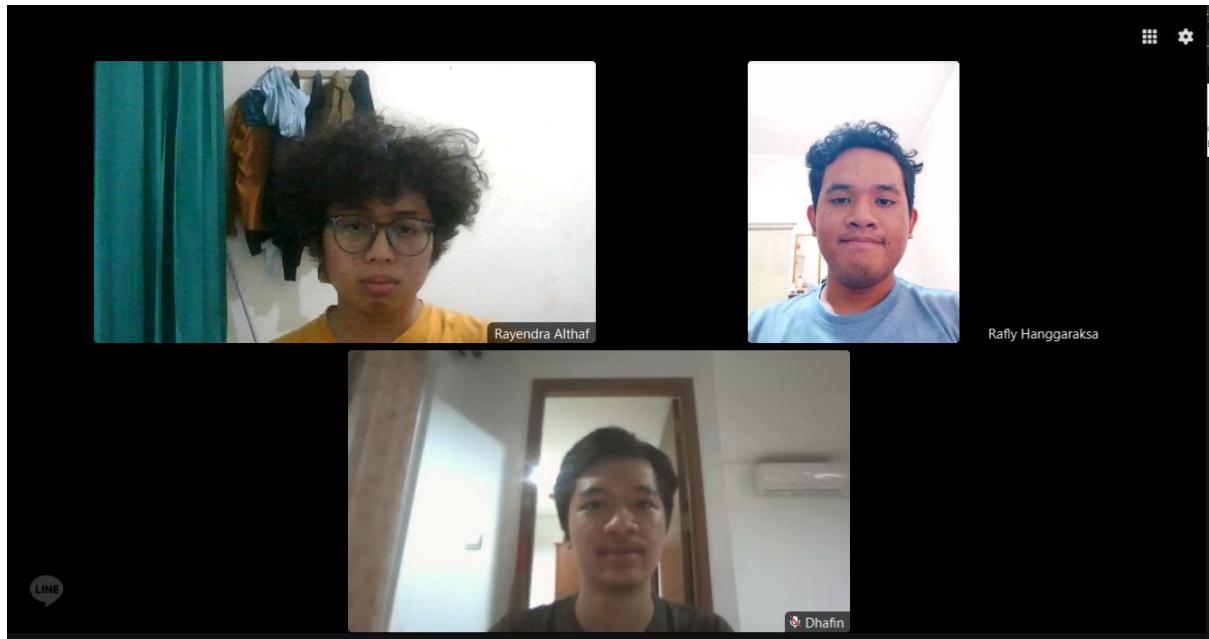


LAPORAN TUGAS BESAR
STRATEGI ALGORITMA
IF 2211



Disusun Oleh:

Raden Rafly Hanggaraksa Budiarto – 13522014

Dhafin Fawwaz Ikramullah – 13522084

Rayendra Althaf Taraka Noor – 13522107

DAFTAR ISI

I. Daftar Isi	2
II. Deskripsi Tugas.....	3
III. Landasan teori.....	4
Boyer-Moore.....	4
Knuth-Morris-Pratt	4
Regex	6
Teknik Pengukuran Presentase Kemiripan	6
Aplikasi Desktop	7
IV. Analisis Pemecahan Masalah.....	8
Langkah Pemecahan Masalah.....	8
Proses penyelesaian dengan KMP dan BM	10
Fitur Fungsional dan Arsitektur Aplikasi.....	13
Contoh Ilustrasi Kasus	13
1. Ilustrasi KMP	13
2. Ilustrasi BM	14
3. Ilustrasi Penggunaan Aplikasi.....	15
V. Implementasi dan Pengujian.....	18
Spesifikasi Teknis Program	18
Tata Cara Penggunaan Program.....	29
Hasil Pengujian	31
Analisis Hasil Pengujian.....	34
VI. Kesimpulan, Saran, Tanggapan, dan Refleksi	35
VII. Lampiran.....	36
VIII. Daftar Pustaka.....	37

I. DESKRIPSI TUGAS

Pada tugas ini, diminta untuk mengimplementasikan sebuah program yang dapat melakukan identifikasi biometrik berbasis sidik jari. Proses implementasi dilakukan dengan menggunakan algoritma Boyer-Moore dan Knuth-Morris-Pratt.



Gambar 1 Langkah Pemecahan Masalah

Gambar yang digunakan pada proses pattern matching kedua algoritma tersebut adalah gambar sidik jari penuh berukuran $m \times n$ pixel yang diambil sebesar 30 pixel setiap kali proses pencocokan data. Untuk tugas ini, Anda dibebaskan untuk mengambil jumlah pixel asalkan didasarkan pada alasan yang masuk akal (dijelaskan pada laporan) dan penanganan kasus ujung yang baik (misal jika ternyata ukuran citra sidik jari tidak habis dibagi dengan ukuran pixel yang dipilih). Selanjutnya, data pixel tersebut akan dikonversi menjadi binary. Karena binary hanya memiliki variasi karakter satu atau nol, maka proses pattern matching akan membuat proses pencocokan karakter menjadi lambat karena harus sering mengulangi proses pencocokan pattern. Cara yang dapat dilakukan untuk mengatasi hal tersebut adalah dengan mengelompokkan setiap baris kode biner per 8 bit sehingga membentuk karakter ASCII. Karakter ASCII 8-bit ini yang akan mewakili proses pencocokan dengan string data.

II. LANDASAN TEORI

BOYER-MOORE

Algoritma Boyer-Moore adalah salah satu algoritma pencocokan pola yang sangat efisien, khususnya untuk kasus di mana pola yang akan dicocokkan lebih pendek dibandingkan dengan teksnya. Algoritma ini diperkenalkan oleh Robert S. Boyer dan J Strother Moore pada tahun 1977. Keunggulan utama dari Boyer-Moore adalah kemampuannya untuk melompati beberapa karakter pada teks sehingga tidak perlu memeriksa setiap karakter satu per satu.

Boyer-Moore bekerja dengan menggunakan dua heuristik utama:

1. Bad Character Heuristic

Heuristik ini bekerja dengan cara:

- Memeriksa apakah karakter pada teks tidak cocok dengan karakter pada pola.
- Jika terjadi ketidakcocokan, maka pola dapat digeser ke kanan hingga posisi terakhir dari karakter yang tidak cocok dalam pola atau menggeser pola melewati karakter yang tidak cocok jika karakter tersebut tidak ada dalam pola.

Langkah-langkah membangun tabel karakter buruk:

- Buat tabel untuk semua karakter yang mungkin muncul dalam teks dan pola.
- Untuk setiap karakter dalam pola, catat posisi terakhir kemunculannya.

KNUTH-MORRIS-PRATT

Algoritma Knuth-Morris-Pratt (KMP) adalah algoritma efisien untuk menemukan posisi pertama kali suatu "pattern" (pola) dalam sebuah "text" (teks). Algoritma ini diperkenalkan oleh Donald Knuth, Vaughan Pratt, dan James H. Morris pada tahun 1977. Kelebihan utama algoritma KMP adalah mampu melakukan pencocokan pola dalam waktu linear $O(n + m)$, di mana n adalah panjang teks dan m adalah panjang pola.

Berikut adalah deskripsi langkah-langkah algoritma KMP:

1. Preproses Pola: Membangun Tabel LPS

Langkah pertama adalah membuat tabel "Longest Prefix Suffix" (LPS) untuk pola. Tabel LPS digunakan untuk menghindari perbandingan karakter yang tidak perlu dengan mengindikasikan panjang proper prefix dari pola yang juga merupakan suffix.

Langkah-langkah membangun tabel LPS:

- Inisialisasi array LPS dengan panjang yang sama dengan pola.
- LPS[0] diinisialisasi dengan 0 karena proper prefix tidak bisa ada di indeks pertama.
- Iterasi mulai dari indeks 1 hingga akhir pola untuk mengisi tabel LPS:
 - Jika karakter pada indeks saat ini cocok dengan karakter pada indeks yang diindikasikan oleh nilai LPS sebelumnya, maka nilai LPS saat ini ditingkatkan dengan 1 dari nilai sebelumnya.
 - Jika tidak cocok, dan nilai LPS bukan nol, kembalikan indeks LPS ke nilai sebelumnya hingga cocok atau mencapai nol.
 - Jika nilai LPS adalah nol dan tidak ada kecocokan, LPS saat ini tetap nol.

Contoh: Untuk pola "ABABCABAB", tabel LPS adalah [0, 0, 1, 2, 0, 1, 2, 3, 4].

2. Pencocokan Pola dalam Teks

Setelah tabel LPS selesai, langkah berikutnya adalah melakukan pencocokan pola dalam teks.

Langkah-langkah pencocokan:

- Iterasi melalui teks dengan dua indeks: satu untuk teks (**i**) dan satu untuk pola (**j**).
- Jika karakter pada **i** (teks) cocok dengan **j** (pola), tingkatkan keduanya.
 - Jika **j** mencapai panjang pola, maka pola telah ditemukan pada indeks **i - j** di teks. Atur ulang **j** dengan nilai dari tabel LPS untuk menghindari perbandingan yang tidak perlu.
 - Jika karakter tidak cocok dan **j** bukan nol, gunakan tabel LPS untuk mengatur ulang **j** tanpa perlu kembali ke awal pola.
 - Jika **j** adalah nol, cukup tingkatkan **i** untuk terus mencari kecocokan.

REGEX

Algoritma Regular Expressions (Regex) adalah sekumpulan karakter yang mendefinisikan pola pencarian dalam teks. Regex sangat fleksibel dan kuat untuk mencocokkan pola yang kompleks. Komponen utama regex meliputi literal characters (karakter biasa), metakarakter (karakter khusus seperti `.`, `*`, `+`, `?`, `|`, `[]`, `()`, `^`, `$`), kuantifikasi (`*` untuk 0 atau lebih kali, `+` untuk 1 atau lebih kali, `?` untuk 0 atau 1 kali, `{n}` untuk tepat n kali, `{n,}` untuk n atau lebih kali, `{n,m}` untuk antara n dan m kali), karakter kelas (seperti `[a-z]` untuk huruf kecil), dan anchors (seperti `^` untuk awal baris dan `$` untuk akhir baris). Grup dan penangkapan digunakan untuk mengelompokkan bagian dari pola.

Misalkan kita ingin mencocokkan semua kata dalam teks "The quick brown fox jumps over the lazy dog" yang dimulai dengan 'b' atau 'f'. Regex yang digunakan adalah `\b[b|f]\w*\b`, di mana `\b` adalah batas kata, `[b|f]` adalah huruf 'b' atau 'f', dan `\w*` adalah karakter kata 0 atau lebih kali.

TEKNIK PENGUKURAN PRESENTASE KEMIRIPAN

Dalam mengukur tingkat kemiripan dari data, program menggunakan **Levenshtein Distance**. Levenshtein distance, juga dikenal sebagai edit distance, adalah sebuah metrik untuk mengukur perbedaan antara dua string. Metrik ini menghitung jumlah operasi penyuntingan yang diperlukan untuk mengubah satu string menjadi string lainnya. Operasi yang dimaksud terdiri dari tiga jenis: penyisipan karakter, penghapusan karakter, dan substitusi karakter.

Levenshtein distance antara dua string aa dan bb didefinisikan sebagai jumlah minimum operasi yang diperlukan untuk mengubah aa menjadi bb . Misalnya, untuk mengubah "kitten" menjadi "sitting":

1. kitten \rightarrow sitten (substitusi 'k' dengan 's')
2. sitten \rightarrow sittin (substitusi 'e' dengan 'i')
3. sittin \rightarrow sitting (penyisipan 'g')

Jadi, Levenshtein distance antara "kitten" dan "sitting" adalah 3.

Untuk menghitung Levenshtein distance, kita dapat menggunakan algoritma dinamis dengan matriks dua dimensi. Matriks DD berukuran $(m+1) \times (n+1)$ di mana m adalah panjang string aa dan n adalah panjang string bb . Elemen $D[i][j]$ menyimpan Levenshtein distance antara substring $a[0 \dots i-1]$ dan $b[0 \dots j-1]$.

Langkah-langkah:

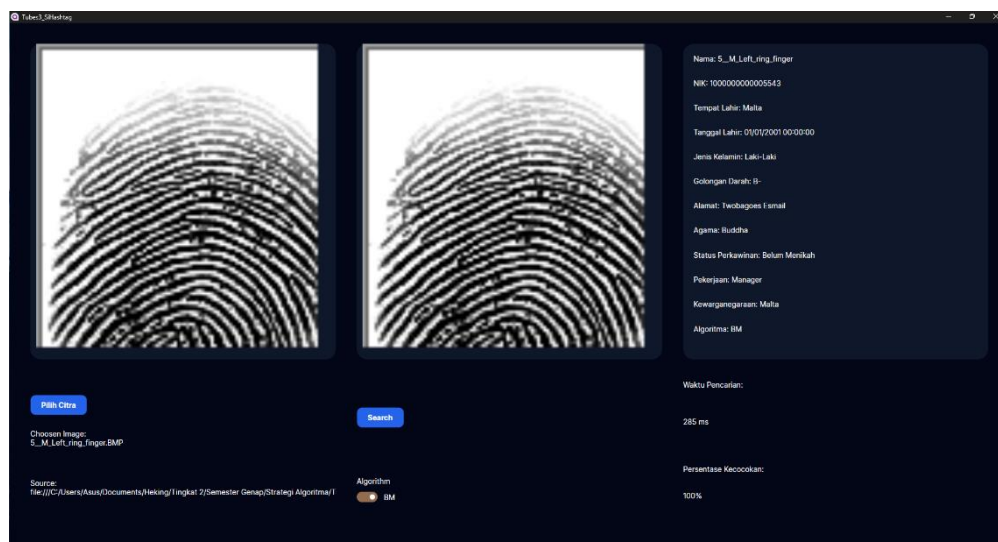
1. Inisialisasi: $D[i][0]=iD[i][0]=i$ untuk ii dari 0 hingga mm dan $D[0][j]=jD[0][j]=j$ untuk jj dari 0 hingga nn .
2. Rekurensi:

$$D[i][j] = \min \begin{cases} D[i-1][j] + 1, \\ D[i][j-1] + 1, \\ D[i-1][j-1] + \text{cost} \end{cases}$$

Dimana $\text{cost} = 0$ jika $a[i-1] = b[j-1]$, dan 1 jika $a[i-1] \neq b[j-1]$

3. Hasil akhir: $D[m][n]$ adalah Levenshtein distance antara a dan b .

APLIKASI DESKTOP



Gambar 2 Tampilan Utama Aplikasi Desktop

Projek ini merupakan suatu program natif berbasis GUI. Program menyediakan input berupa gambar yang dapat diakses oleh pengguna untuk menentukan data yang diinginkannya. Selanjutnya, program akan memberikan opsi penggunaan algoitma pencarian KMP atau BM dengan tujuan perbandingan performa. Apabila program tidak dapat menemukan data yang cocok melalui KMP atau BM, program akan mengalihkan pencarian menggunakan Levenstein Distance. Program akan mengembalikan jawaban menggunakan data dengan Levenstein Distance terkecil.

III. ANALISIS PEMECAHAN MASALAH

LANGKAH PEMECAHAN MASALAH

Proses pemrosesan gambar sidik jari dalam program ini terdiri dari beberapa langkah yang terperinci. Pertama, gambar yang dimasukkan oleh pengguna akan dipotong sebanyak 256 pixel. Keputusan ini diambil dikarenakan apabila hanya diambil sebanyak 30 pixel maka gambar tidak cukup unik untuk ditentukan pada database. Hal ini menyebabkan hasil yang tidak akurat. Selanjutnya, setiap piksel pada gambar tersebut akan diiterasi dan diubah menjadi skala abu-abu (greyscale).

```
int middle = midPoint;
int startingIndex = middle / 8*8 - CHOSEN_PIXEL_SIZE/2;
int endingIndex = startingIndex + CHOSEN_PIXEL_SIZE;

int firstY = startingIndex / image.Width;
int firstX = startingIndex % image.Width;

int lastY = endingIndex / image.Width;
int lastX = endingIndex % image.Width;

int currentX = firstX;
int currentY = firstY;

do {
    Color pixelColor = image.GetPixel(currentX, currentY);
    double grayscale = (pixelColor.R * 0.299 + pixelColor.G * 0.587 + pixelColor.B * 0.114) / 255.0;
    int binaryValue = grayscale > THRESHOLD ? 1 : 0;
    binaryStr += binaryValue;

    if (binaryStr.Length == 8) {
        byte binaryByte = Convert.ToByte(binaryStr, 2);
        ascii += (char)binaryByte;
        binaryStr = "";
    }

    currentX++;
    if(currentX == image.Width){
        currentX = 0;
        currentY++;
    }
}while(currentX != lastX || currentY != lastY);

// Dispose of the image
image.Dispose();
```

Untuk mengecek nilai biner dari skala abu-abu tersebut, program menetapkan suatu ambang batas (threshold). Apabila nilai skala abu-abu lebih besar dibandingkan threshold tersebut, nilai biner dari piksel tersebut akan bernilai satu, dan sebaliknya, jika lebih kecil atau sama dengan threshold, nilai biner akan bernilai nol.

```
Color pixelColor = image.GetPixel(currentX, currentY);
double grayscale = (pixelColor.R * 0.299 + pixelColor.G * 0.587 + pixelColor.B * 0.114) / 255.0;
int binaryValue = grayscale > THRESHOLD ? 1 : 0;
```


Seluruh nilai biner yang dihasilkan dari gambar tersebut kemudian akan diubah menjadi format ASCII 8-bit. Nilai ASCII 8-bit ini akan diproses menggunakan algoritma pattern matching terhadap tabel sidik jari yang berisi data ASCII 8-bit lainnya untuk mencari kecocokan.

```
byte binaryByte = Convert.ToByte(binaryStr, 2);
ascii += (char)binaryByte;
binaryStr = "";
```

Jika tidak ditemukan hasil melalui algoritma pattern matching, program secara otomatis akan mengalihkan pencarian menggunakan metode Levenshtein Distance. Pencarian menggunakan Levenshtein Distance tidak menggunakan gambar yang telah dipotong sebanyak 256 pixel, melainkan seluruh piksel gambar diubah menjadi format ASCII 8-bit untuk dibandingkan. Hal ini bertujuan untuk memperoleh nilai kemiripan yang besar. Apabila hanya digunakan sebanyak 256 pixel, persentase kemiripan hanya akan bernilai 0% sampai 40%.

```
public FingerSolution Solve(SidikJari sjInput){
    FingerSolution sol = new FingerSolution().StartTimer();
    ProcessCalculation(sjInput, AllSidikJari, ref sol);
    if(sol.SidikJari != null) sol.PersentaseKecocokan = 1;
    else SolveWithLevenstheinDistance(new SidikJari(sjInput.BerkasCitra, sjInput.Nama),
    AllSidikJari, ref sol);

    sol.Biodata = FindBiodata(sol.SidikJari);
    return sol.StopTimer();
}
```

Setelah data yang sesuai ditemukan atau ditentukan, program akan kembali menggunakan metode Levenshtein Distance untuk menentukan primary key dari atribut "nama" terdekat. Hal ini dilakukan untuk mencari biodata dari pemilik sidik jari tersebut, sehingga identifikasi dapat dilakukan dengan lebih akurat. Dalam menentukan nilai Levenshtein Distance pada bahasa alay, program membandingkan data nama asli dengan data nama yang telah diubah menjadi bahasa alay. Kemudian program akan mencari distance terkecil diantara keduanya yang akan dijadikan suatu jarak dari pasangan key.

```
private Biodata FindBiodata(SidikJari sj) {
    List<Biodata> biodatalist = Biodata.GetAll();
    Biodata result = null;
    double _smallestDistance = double.MaxValue;
    foreach(Biodata biodata in biodatalist){

        string bioName = biodata.Nama.ToCompareAlay();
        string sjName = sj.Nama.ToCompareAlay();

        // pure string compute
        double pureLevenshteinDistance = LevenshteinDistance.Solve(biodata.Nama, sj.Nama);
        double pureNormalized = pureLevenshteinDistance/double.Max(biodata.Nama.Length,
sj.Nama.Length);

        // corrupted + pure string compute
        double corruptedLevenshteinDistance = LevenshteinDistance.Solve(bioName, sjName);
        double corruptedNormalized = (corruptedLevenshteinDistance/double.Max(bioName.Length,
sjName.Length) + pureNormalized) / 2;

        // result
```

```

        double minResult = double.Min(pureLevenshteinDistance, corruptedNormalized);
        if(_smallestDistance > minResult){
            _smallestDistance = minResult;
            result = biodata;
            if(_smallestDistance == 0){
                return result;
            }
        }
    }
    return result;
}

```

PROSES PENYELESAIAN DENGAN KMP DAN BM

Kode yang berikut mengimplementasikan algoritma pencarian Boyer-Moore. Algoritma ini mengurangi jumlah perbandingan karakter selama pencarian dengan menggunakan informasi yang diperoleh selama tahap prapemrosesan. Berikut adalah penjelasan rinci mengenai fungsi-fungsi dalam kode tersebut.

Fungsi 'LastOccurenceCalculation' bertugas melakukan prapemrosesan terhadap pola (pattern) untuk menghitung tabel kemunculan terakhir (last occurrence table). Tabel ini menyimpan indeks terakhir di mana setiap karakter muncul dalam pola. Pada awalnya, tabel 'lastOc' berukuran 256 (sesuai jumlah karakter ASCII) diinisialisasi dengan nilai -1. Kemudian, untuk setiap karakter dalam pola, tabel diisi dengan indeks terakhir di mana karakter tersebut muncul dalam pola.

```

static void LastOccurenceCalculation(string pat, int[] lastOc)
{
    int size = pat.Length;
    for (int i = 0; i < 256; i++)
        lastOc[i] = -1;
    for (int i = 0; i < size; i++)
        lastOc[(int)pat[i]] = i;
}

```

Selanjutnya, fungsi 'BMSearch' mengimplementasikan algoritma Boyer-Moore untuk mencari pola dalam teks. Fungsi ini dimulai dengan mendefinisikan panjang teks ('n') dan panjang pola ('m'). Kemudian, tabel kemunculan terakhir ('lastOc') dihitung menggunakan fungsi 'LastOccurenceCalculation'. Indeks 'i' diinisialisasi pada posisi akhir pola dalam teks, dan indeks 'j' diinisialisasi pada posisi akhir pola.

```

static int BMSearch(string pat, string txt)
{
    int n = txt.Length;
    int m = pat.Length;
    // Preprocessing pattern
    int[] lastOc = new int[256];
    LastOccurenceCalculation(pat, lastOc);

    int i = m - 1;
    int j = m - 1;
    while (i <= n - 1) {
        if(txt[i] == pat[j]){
            if(j == 0){
                return i;
            }
        }
        else{

```

```

        i--;
        j--;
    }
    } else{
        // Lookup to last occurrence table
        i = i + m - int.Min(j, 1 + lastOc[(int)txt[i]]);
        j = m - 1;
    }
    }
    return -1;
}

```

Dalam proses pencarian, jika karakter teks dan pola pada indeks yang sesuai cocok, fungsi akan memeriksa apakah `j` (indeks dalam pola) adalah 0. Jika ya, ini menunjukkan bahwa seluruh pola telah ditemukan dalam teks, dan indeks awal pencocokan dikembalikan. Jika tidak, `i` dan `j` dikurangi masing-masing satu untuk membandingkan karakter sebelumnya. Namun, jika karakter tidak cocok, indeks `i` disesuaikan menggunakan tabel kemunculan terakhir dan algoritma Boyer-Moore, yang memungkinkan `i` maju beberapa posisi untuk menghindari perbandingan yang tidak perlu. Setelah itu, `j` direset ke posisi terakhir dalam pola. Jika pencarian selesai tanpa menemukan pola, fungsi mengembalikan nilai -1, menunjukkan bahwa pola tidak ditemukan dalam teks.

Kode berikut mengimplementasikan algoritma Knuth-Morris-Pratt (KMP) untuk pencarian pola dalam teks. Berikut adalah penjelasan rinci mengenai fungsi-fungsi dalam kode tersebut.

Fungsi `KMPSearch` bertugas untuk mencari pola (pattern) dalam teks (text). Pertama, panjang pola (`M`) dan panjang teks (`N`) ditentukan. Kemudian, sebuah array `lps` (longest prefix suffix) dibuat untuk menyimpan nilai-nilai prefix suffix terpanjang dari pola. Indeks `j` diinisialisasi pada posisi 0 untuk pola.

```

int KMPSearch(string pattern, string txt) {
    // longest prefix suffix values
    int[] lps = new int[pattern.Length];
    int j = 0;

    computeLPSArray(pattern, lps);

    int i = 0;
    while (i < txt.Length) {
        if (pattern[j] == txt[i]) {
            j++;
            i++;
        }
        if (j == pattern.Length) {
            return i-j;
        }

        // mismatch after j matches
        else if (i < txt.Length && pattern[j] != txt[i]) {
            if (j != 0) j = lps[j - 1];
            else i = i + 1;
        }
    }

    // Case not found
    return -1;
}

```

Tahap pertama dari proses ini adalah prapemrosesan pola untuk menghitung array `lps`, yang dilakukan dengan memanggil fungsi `computeLPSArray`. Setelah array `lps` dihitung, pencarian pola dalam teks dimulai. Indeks `i` diinisialisasi pada posisi 0 untuk teks.

Dalam proses pencarian, jika karakter pola dan teks pada indeks yang sesuai cocok, indeks `j` dan `i` keduanya ditingkatkan. Jika `j` mencapai panjang pola (`M`), ini menunjukkan bahwa seluruh pola telah ditemukan dalam teks, dan fungsi mengembalikan indeks awal pencocokan (`i - j`). Jika terjadi ketidakcocokan setelah beberapa kecocokan (`j` tidak sama dengan 0), indeks `j` disesuaikan menggunakan nilai dari array `lps`. Jika `j` adalah 0, indeks `i` ditingkatkan satu.

Fungsi `computeLPSArray` bertugas menghitung array `lps` untuk pola yang diberikan. Array `lps` menyimpan panjang dari prefix suffix terpanjang untuk setiap posisi dalam pola. Pada awalnya, panjang prefix suffix (`len`) diinisialisasi pada 0, dan `lps[0]` diatur menjadi 0. Indeks `i` diinisialisasi pada posisi 1.

```
void computeLPSArray(string pattern, int[] lps) {
    // length of the previous longest prefix suffix
    int len = 0;
    int i = 1;
    lps[0] = 0;

    while (i < pattern.Length) {
        if (pattern[i] == pattern[len]) {
            len++;
            lps[i] = len;
            i++;
        }
        else {
            if (len != 0) len = lps[len - 1];
            else {
                lps[i] = len;
                i++;
            }
        }
    }
}
```

Dalam proses perhitungan `lps`, jika karakter pada indeks `i` dalam pola cocok dengan karakter pada indeks `len`, nilai `len` ditingkatkan dan nilai `lps[i]` diatur menjadi `len`, kemudian `i` ditingkatkan. Jika terjadi ketidakcocokan dan `len` tidak sama dengan 0, nilai `len` disesuaikan menggunakan nilai dari array `lps` pada posisi `len - 1`. Jika `len` adalah 0, `lps[i]` diatur menjadi 0, dan `i` ditingkatkan. Dengan menggunakan array `lps`, algoritma KMP dapat menghindari perbandingan ulang yang tidak perlu, sehingga meningkatkan efisiensi pencarian pola dalam teks.

FITUR FUNGSIONAL DAN ARSITEKTUR APLIKASI

Solusi yang dibuat berupa program yang dikembangkan menggunakan bahasa pemrograman C#. Framework yang digunakan untuk mengembangkan Graphical User Interface (GUI) menggunakan AvaloniaUI. Database yang digunakan adalah MariaDB (khususnya versi 11.3.2-MariaDB-1:11.3.2+maria~ubu2204). Fitur yang ada pada program diantaranya:

- Memasukkan gambar sidik jari untuk dicari pemiliknya
- Memilih algoritma yang diinginkan antara KMP atau BM
- Mencari pemilih dari sidik jari yang diinginkan

CONTOH ILUSTRASI KASUS

1. Ilustrasi KMP

Misalkan kita ingin mencari pola "ABABCABAB" dalam teks "ABABDABACDABABCABAB".

Langkah-langkah KMP:

Prapemrosesan Pola: Hitung Array LPS

Pola: "ABABCABAB"

LPS Array:

$lps[0] = 0$

$lps[1] = 0$

$lps[2] = 1$

$lps[3] = 2$

$lps[4] = 0$

$lps[5] = 1$

$lps[6] = 2$

$lps[7] = 3$

$lps[8] = 4$

Pencarian dalam Teks

Mulai dengan $i = 0$ (teks) dan $j = 0$ (pola).

Cocokkan karakter satu per satu.

Jika karakter cocok, tingkatkan i dan j.

Jika j mencapai panjang pola, pola ditemukan.

Jika terjadi ketidakcocokan dan j bukan 0, gunakan nilai dari array LPS untuk mengatur ulang j.

Jika j adalah 0, tingkatkan i.

Proses Pencarian:

i=0, j=0: teks[0]='A', pola[0]='A' (cocok)

i=1, j=1: teks[1]='B', pola[1]='B' (cocok)

i=2, j=2: teks[2]='A', pola[2]='A' (cocok)

i=3, j=3: teks[3]='B', pola[3]='B' (cocok)

i=4, j=4: teks[4]='D', pola[4]='C' (tidak cocok, set j=lps[3]=2)

...

Proses berlanjut hingga i=15, j=9: teks[15]='B', pola[8]='B' (cocok)

Pola ditemukan di indeks 10 dalam teks.

2. Ilustrasi BM

Kasus:

Misalkan kita ingin mencari pola "EXAMPLE" dalam teks "HERE IS A SIMPLE EXAMPLE".

Langkah-langkah Boyer-Moore:

Prapemrosesan Pola: Hitung Tabel Kemunculan Terakhir

Pola: "EXAMPLE"

Tabel Kemunculan Terakhir:

E: 6

X: 1

A: 4

M: 3

P: 2

L: 5

Semua karakter lainnya diset -1.

Pencarian dalam Teks

Mulai dengan $i = 6$ (posisi akhir pola pada teks) dan $j = 6$ (posisi akhir pola).

Cocokkan karakter pola dari akhir ke awal.

Jika karakter tidak cocok, geser pola ke kanan berdasarkan tabel kemunculan terakhir.

Ulangi hingga pola ditemukan atau teks habis.

Proses Pencarian:

$i=6, j=6$: teks[6]=' ', pola[6]='E' (tidak cocok, geser $i += \max(1, j - \text{last_occ}[\text{teks}[i]])$, $i=13$)

$i=13, j=6$: teks[13]='P', pola[6]='E' (tidak cocok, geser $i += \max(1, j - \text{last_occ}[\text{teks}[i]])$, $i=18$)

$i=18, j=6$: teks[18]='E', pola[6]='E' (cocok)

$i=17, j=5$: teks[17]='L', pola[5]='L' (cocok)

$i=16, j=4$: teks[16]='P', pola[4]='P' (cocok)

$i=15, j=3$: teks[15]='M', pola[3]='M' (cocok)

$i=14, j=2$: teks[14]='A', pola[2]='A' (cocok)

$i=13, j=1$: teks[13]=' ', pola[1]='X' (tidak cocok, geser $i += \max(1, j - \text{last_occ}[\text{teks}[i]])$, $i=20$)

$i=20, j=6$: teks[20]='E', pola[6]='E' (cocok)

...

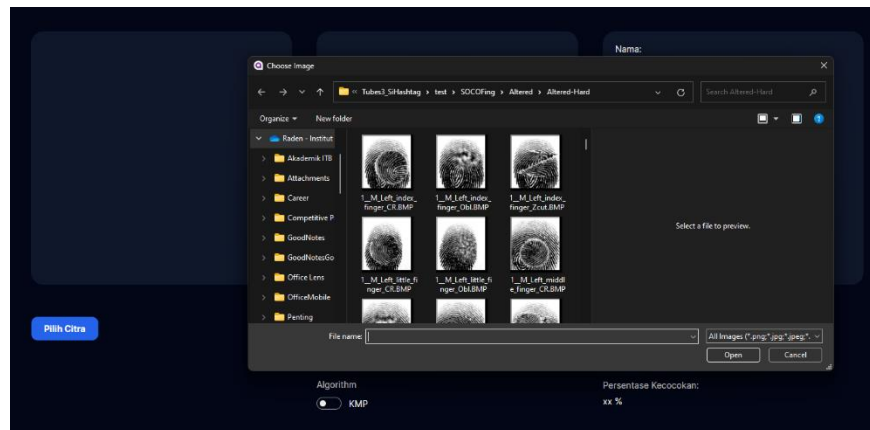
Proses berlanjut hingga $i=24, j=6$: teks[24]='E', pola[0]='E' (cocok)

Pola ditemukan di indeks 17 dalam teks.

3. Ilustrasi Penggunaan Aplikasi

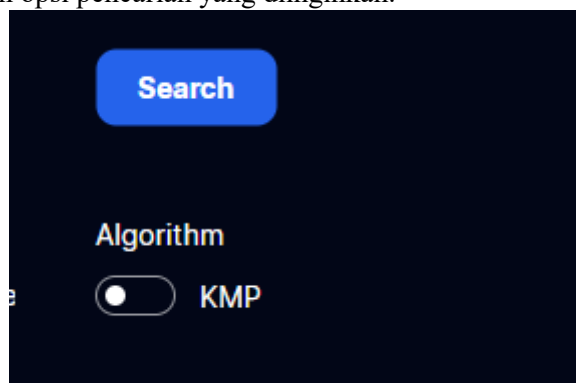
Langkah – langkah menggunakan aplikasi:

1. Pengguna memilih gambar yang ingin dimasukkan.

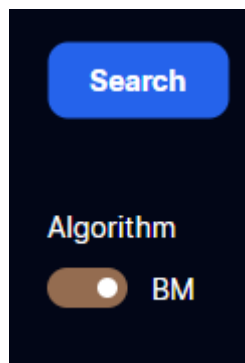


Gambar 3 Prompt untuk Input Gambar

2. Pengguna memilih opsi pencarian yang diinginkan.

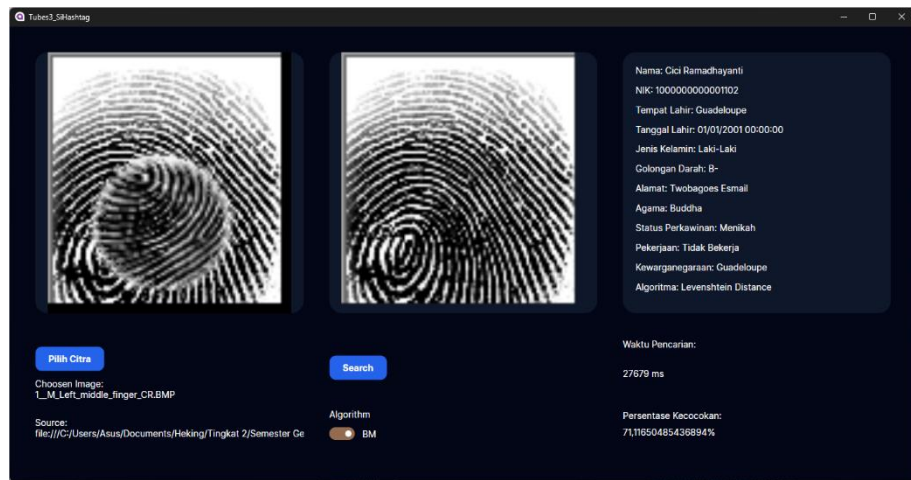


Gambar 4 Toggle KMP



Gambar 5 Toggle BM

3. Pengguna mengeksekusi pencarian.



Gambar 6 Hasil Eksekusi Pencarian

IV. IMPLEMENTASI DAN PENGUJIAN

SPEKIFIKASI TEKNIS PROGRAM

Dalam program ini, terdapat Kelas 'FingerSolver' yang berperan sebagai entry point untuk data yang dimasukkan oleh pengguna. Kelas ini bersifat abstract agar kelas 'BMSolver' serta 'KMPSolver' dapat menggunakan algoritmanya masing – masing.

```
public abstract class FingerSolver{

    /// <summary>
    /// Return the solution depending on the algorithm in FingerSolution
    /// </summary>
    /// <param name="sjInput">Input from user</param>
    /// <returns>FingerSolution</returns>
    public FingerSolution Solve(SidikJari sjInput){
        FingerSolution sol = new FingerSolution().StartTimer();
        ProcessCalculation(sjInput, AllSidikJari, ref sol);
        if(sol.SidikJari != null) sol.PersentaseKecocokan = 1;
        else SolveWithLevenstheinDistance(new SidikJari(sjInput.BerkasCitra, sjInput>Nama),
AllSidikJari, ref sol);

        sol.Biodata = FindBiodata(sol.SidikJari);
        return sol.StopTimer();
    }

    // Case not found
    void SolveWithLevenstheinDistance(SidikJari sj, List<SidikJari> listSj, ref FingerSolution
sol){
        double percentage = 0;
        int smallest = int.MaxValue;

        // Paralel version
        int unsafeCounter = 0;
        int[] distances = new int[listSj.Count];
        Parallel.For(0, listSj.Count, new ParallelOptions { MaxDegreeOfParallelism =
Environment.ProcessorCount }, i => {
            distances[i] = LevenshteinDistance.Solve(sj.Ascii, listSj[i].Ascii);
            if(++unsafeCounter % 1000 == 0) Console.WriteLine("LevenshteinDistance: " +
unsafeCounter + " / " + listSj.Count);
        });

        for(int i = 0; i < listSj.Count; i++) {
            if(distances[i] < smallest) {
                smallest = distances[i];
                sol.SidikJari = listSj[i];
                // percentage = ((double)distances[i]) / double.Max(sj.Ascii.Length,
listSj[i].Ascii.Length); // case all pixel
                double bigger = double.Max(sj.Ascii.Length, listSj[i].Ascii.Length);
                double smaller = double.Min(sj.Ascii.Length, listSj[i].Ascii.Length);
                percentage = Mathf.InverseLerp(bigger - smaller, bigger, smallest);
            }
        }

        sol.PersentaseKecocokan = 1-percentage;
    }

    /// <summary>
    /// // Populate solution in FingerSolution
    /// </summary>
    /// <param name="sj">Input from user</param>
    /// <param name="listSj">List to search in database</param>
    /// <param name="sol">Solution itself</param>
```

```

        protected abstract void ProcessCalculation(SidikJari sj, List<SidikJari> listSj, ref
FingerSolution sol);

        /// <summary>
        /// return all sidik jari
        /// </summary>
        /// <returns></returns>
        private static List<SidikJari> cachedSidikJariList = null;
        private static List<SidikJari> AllSidikJari => cachedSidikJariList ?? (cachedSidikJariList =
SidikJari.GetAll());
        public static void Initialize() => cachedSidikJariList = SidikJari.GetAll();

        /// <summary>
        /// Find user data with regex because of bahasa alay stuff
        /// </summary>
        /// <param name="sj">The SidikJari found on database with ProcessCalculation</param>
        /// <returns></returns>
        private Biodata FindBiodata(SidikJari sj) {
            List<Biodata> biodataList = Biodata.GetAll();
            Biodata result = null;
            double _smallestDistance = double.MaxValue;
            foreach(Biodata biodata in biodataList){

                string bioName = biodata>Nama.ToCompareAlay();
                string sjName = sj>Nama.ToCompareAlay();

                // pure string compute
                double pureLevenshteinDistance = LevenshteinDistance.Solve(biodata>Nama, sj>Nama);
                double pureNormalized = pureLevenshteinDistance/double.Max(biodata>Nama.Length,
sj>Nama.Length);

                // corrupted + pure string compute
                double corruptedLevenshteinDistance = LevenshteinDistance.Solve(bioName, sjName);
                double corruptedNormalized =
(corruptedLevenshteinDistance/double.Max(bioName.Length, sjName.Length) + pureNormalized) / 2;

                // result
                double minResult = double.Min(pureLevenshteinDistance, corruptedNormalized);
                if(_smallestDistance > minResult){
                    _smallestDistance = minResult;
                    result = biodata;
                    if(_smallestDistance == 0){
                        return result;
                    }
                }
            }
            return result;
        }
    }
}

```

```

public class BMSolver : FingerSolver{
    protected override void ProcessCalculation(SidikJari sj, List<SidikJari> listSj, ref
FingerSolution sol){
        SidikJari result = null;
        for(int i = 0; i < listSj.Count; i++) {
            int res = BMSearch(sj.Ascii, listSj[i].Ascii);
            if(res != -1) {
                result = listSj[i];
                break;
            }
        }
        sol.SidikJari = result;
    }

    // Last Occurence table preprocessing
    static void LastOccurenceCalculation(string pat, int[] lastOc)
    {
        int size = pat.Length;
    }
}

```

```

        for (int i = 0; i < 256; i++)
            lastOc[i] = -1;
        for (int i = 0; i < size; i++)
            lastOc[(int)pat[i]] = i;
    }

    static int BMSearch(string pat, string txt)
    {
        int n = txt.Length;
        int m = pat.Length;

        // Preprocessing pattern
        int[] lastOc = new int[256];
        LastOccurenceCalculation(pat, lastOc);

        int i = m - 1;
        int j = m - 1;
        while (i <= n - 1) {
            if(txt[i] == pat[j]){
                if(j == 0){
                    return i;
                }
                else{
                    i--;
                    j--;
                }
            } else{
                // Lookup to last occurrence table
                i = i + m - int.Min(j, 1 + lastOc[(int)txt[i]]);
                j = m - 1;
            }
        }
        return -1;
    }

    public override string ToString() => "BM";
}

public class KMPSolver : FingerSolver{
    protected override void ProcessCalculation(SidikJari sj, List<SidikJari> listSj, ref
FingerSolution sol){
        SidikJari result = null;
        for(int i = 0; i < listSj.Count; i++) {
            int res = KMPSearch(sj.Ascii, listSj[i].Ascii);
            if(res != -1) {
                result = listSj[i];
                break;
            }
        }
        sol.SidikJari = result;
    }

    int KMPSearch(string pattern, string txt) {
        // longest prefix suffix values
        int[] lps = new int[pattern.Length];
        int j = 0;

        computeLPSArray(pattern, lps);

        int i = 0;
        while (i < txt.Length) {
            if (pattern[j] == txt[i]) {
                j++;
                i++;
            }
        }
    }
}

```

```

        if (j == pattern.Length) {
            return i-j;
        }

        // mismatch after j matches
        else if (i < txt.Length && pattern[j] != txt[i]) {
            if (j != 0) j = lps[j - 1];
            else i = i + 1;
        }
    }

    // Case not found
    return -1;
}

void computeLPSArray(string pattern, int[] lps) {
    // length of the previous longest prefix suffix
    int len = 0;
    int i = 1;
    lps[0] = 0;

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

public override string ToString() => "KMP";
}

```

Selanjutnya terdapat kelas `Levenshtein Distance` yang memiliki method static `Solve` yang berperan untuk melakukan perhitungan menggunakan Levenshtein Distance.

```

public class LevenshteinDistance {
    public static int Solve(string A, string B) {
        int ALength = A.Length;
        int BLength = B.Length;

        int[,] mat = new int[ALength + 1, BLength + 1];

        // Return full length if any empty
        if (ALength == 0) return BLength;
        if (BLength == 0) return ALength;

        // Init matrix
        for (int i = 0; i <= ALength; i++) {
            mat[i, 0] = i;
        }
        for (int j = 0; j <= BLength; j++) {
            mat[0, j] = j;
        }

        // rows, columns distances
        for (int i = 1; i <= ALength; i++) {
            for (int j = 1; j <= BLength; j++) {

```

```

        int cost = (B[j - 1] == A[i - 1]) ? 0 : 1;

        mat[i, j] = Math.Min(
            Math.Min(mat[i - 1, j] + 1, mat[i, j - 1] + 1),
            mat[i - 1, j - 1] + cost
        );
    }
}
return mat[A.Length, B.Length];
}
}

```

Untuk menyimpan data dari sidik jari, digunakan Kelas `SidikJari` yang berperan sebagai struktur data untuk menyimpan data filepath, nama dari pemilik sidik jari tersebut, dan nilai ascii hasil pengolahan.

```

public class SidikJari{
    const int CHOSEN_PIXEL_SIZE = 256;
    public const double THRESHOLD = 0.2;
    // Path to image file
    public string BerkasCitra => _berkasCitra;

    public string Nama => _nama;
    public string Ascii => _ascii; // generated later

    string _berkasCitra; // path to image file
    string _nama;
    string _ascii;

    public SidikJari(string berkasCitra, string nama){
        _berkasCitra = berkasCitra;
        _nama = nama;
        _ascii = ReadImageASCII();
    }

    private SidikJari(string berkasCitra){
        _berkasCitra = berkasCitra;
        _nama = "";
    }

    public static SidikJari GetSidikJariInChosenPixelSize(string berkasCitra){
        SidikJari sj = new SidikJari(berkasCitra);
        sj._ascii = sj.ReadImageASCIIChosenPixelSize();
        return sj;
    }

    public static SidikJari GetSidikJariFull(string berkasCitra){
        SidikJari sj = new SidikJari(berkasCitra);
        sj._ascii = sj.ReadImageASCII();
        return sj;
    }

    public SidikJari(string berkasCitra, string nama, string ascii){
        _berkasCitra = berkasCitra;
        _nama = nama;
        _ascii = ascii;
    }

    public static List<SidikJari> GetAll(){
        MySqlDataReader reader = Database.Execute("SELECT * FROM sidik_jari");
        List<SidikJari> list = new List<SidikJari>();
        while(reader.Read()){
            SidikJari sidikJari = new SidikJari(
                reader.GetString("berkas_citra"),
                reader.GetString("nama"),
                reader.GetString("ascii")
            );
        }
    }
}

```

```

        list.Add(sidikJari);
    }
    reader.Close();
    return list;
}

private string cleanPrefix(string path){
    return path.Replace("file:///", "");
}

// Return string representation of image by path
public string ReadImageASCII(){
    _berkasCitra = cleanPrefix(_berkasCitra);
    Bitmap image = new Bitmap(_berkasCitra);

    string binaryStr = "";
    string ascii = "";
    for (int y = 0; y < image.Height; y++) {
        for (int x = 0; x < image.Width; x++) {
            Color pixelColor = image.GetPixel(x, y);
            double grayscale = (pixelColor.R * 0.299 + pixelColor.G * 0.587 + pixelColor.B *
0.114) / 255.0;
            int binaryValue = grayscale > THRESHOLD ? 1 : 0;
            binaryStr += binaryValue;

            if (binaryStr.Length == 8) {
                byte binaryByte = Convert.ToByte(binaryStr, 2);
                ascii += (char)binaryByte;
                binaryStr = "";
            }
        }
    }

    // Handle if any remaining bits
    if (binaryStr.Length > 0) {
        // add 0 until 8 bits
        while (binaryStr.Length < 8) {
            binaryStr += "0";
        }
        byte binaryByte = Convert.ToByte(binaryStr, 2);
        ascii += (char)binaryByte;
    }

    // Dispose of the image
    image.Dispose();
    return ascii;
}

public void PopulateImageAscii(){
    _ascii = ReadImageASCII();
}

// Read only CHOSEN_PIXEL_SIZE middle binary
public string ReadImageASCIICHosenPixelSize(){
    _berkasCitra = cleanPrefix(_berkasCitra);
    Bitmap image = new Bitmap(_berkasCitra);

    string binaryStr = "";
    string ascii = "";

    int binaryLength = image.Height * image.Width;

    int midPoint = image.Width*(image.Height/2) + image.Width/2;

    // Edge case if the image is too small
    if(binaryLength < CHOSEN_PIXEL_SIZE){
        // Add everything until binaryLength but ensure it can be divided by 8

```

```

        for (int y = 0; y < image.Height; y++) {
            for (int x = 0; x < image.Width; x++) {
                Color pixelColor = image.GetPixel(x, y);
                double grayscale = (pixelColor.R * 0.299 + pixelColor.G * 0.587 +
pixelColor.B * 0.114) / 255.0;
                int binaryValue = grayscale > THRESHOLD ? 1 : 0;
                binaryStr += binaryValue;

                if (binaryStr.Length == 8) {
                    byte binaryByte = Convert.ToByte(binaryStr, 2);
                    ascii += (char)binaryByte;
                    binaryStr = "";
                }
            }
            return ascii;
        }
    }

    int middle = midPoint;
    int startingIndex = ((int)middle/8)*8 - CHOSEN_PIXEL_SIZE/2;
    int endingIndex = startingIndex + CHOSEN_PIXEL_SIZE;

    int firstY = startingIndex / image.Width;
    int firstX = startingIndex % image.Width;

    int lastY = endingIndex / image.Width;
    int lastX = endingIndex % image.Width;

    int currentX = firstX;
    int currentY = firstY;

    do {
        Color pixelColor = image.GetPixel(currentX, currentY);
        double grayscale = (pixelColor.R * 0.299 + pixelColor.G * 0.587 + pixelColor.B *
0.114) / 255.0;
        int binaryValue = grayscale > THRESHOLD ? 1 : 0;
        binaryStr += binaryValue;

        if (binaryStr.Length == 8) {
            byte binaryByte = Convert.ToByte(binaryStr, 2);
            ascii += (char)binaryByte;
            binaryStr = "";
        }

        currentX++;
        if(currentX == image.Width){
            currentX = 0;
            currentY++;
        }
    }while(currentX != lastX || currentY != lastY);

    // Dispose of the image
    image.Dispose();

    return ascii;
}
}

```

Untuk menyimpan Biodata yang diperoleh dari SQL, program menggunakan Kelas `Biodata` yang memiliki komposisi dari Kelas `Database`.

```

public class Biodata{
    // Getter
    public string NIK => nik;
}

```



```

public string Nama => nama;
public string TempatLahir => tempat_lahir;
public DateTime TanggalLahir => tanggal_lahir;
public string JenisKelamin => jenis_kelamin;
public string GolonganDarah => golongan_darah;
public string Alamat => alamat;
public string Agama => agama;
public string StatusPerkawinan => status_perkawinan;
public string Pekerjaan => pekerjaan;
public string Kewarganegaraan => kewarganegaraan;

string nik;
string nama;
string tempat_lahir;
DateTime tanggal_lahir;
string jenis_kelamin;
string golongan_darah;
string alamat;
string agama;
string status_perkawinan;
string pekerjaan;
string kewarganegaraan;

public static List<Biodata> GetAll(){
    MySqlDataReader reader = Database.Execute("SELECT * FROM biodata");
    List<Biodata> list = new List<Biodata>();
    while(reader.Read()){
        Biodata biodata = new Biodata
        {
            nik = reader.GetString("NIK"),
            nama = reader.GetString("nama"),
            tempat_lahir = reader.GetString("tempat_lahir"),
            tanggal_lahir = reader.GetDateTime("tanggal_lahir"),
            jenis_kelamin = reader.GetString("jenis_kelamin"),
            golongan_darah = reader.GetString("golongan_darah"),
            alamat = reader.GetString("alamat"),
            agama = reader.GetString("agama"),
            status_perkawinan = reader.GetString("status_perkawinan"),
            pekerjaan = reader.GetString("pekerjaan"),
            kewarganegaraan = reader.GetString("kewarganegaraan")
        };
        list.Add(biodata);
    }
    reader.Close();
    return list;
}
}

public class Database{
    static MySqlConnection _connection;
    static bool _initialized = false;
    public static void Initialize(){
        Env.Load();
        string serverIp = Environment.GetEnvironmentVariable("SERVER_IP");        if(serverIp ==
null || serverIp == "") serverIp = "localhost";
        string dbUser = Environment.GetEnvironmentVariable("DB_USER");            if(dbUser ==
null || dbUser == "") dbUser = "root";
        string dbPassword = Environment.GetEnvironmentVariable("DB_PASSWORD");    if(dbPassword ==
null || dbPassword == "") dbPassword = "password";
        string dbName = Environment.GetEnvironmentVariable("DB_NAME");            if(dbName ==
null || dbName == "") dbName = "finger";
        string port = Environment.GetEnvironmentVariable("DB_PORT");              if(port == null
|| port == "") port = "3306";
        string connstring =
"server="+serverIp+";uid="+dbUser+";pwd="+dbPassword+";database="+dbName+";port="+port;

        try {
            _connection = new MySqlConnection(connstring);

```

```

        _connection.Open();
        _initialized = true;
    } catch (MySqlException ex) {
        Console.WriteLine(ex.ToString());
    }
}

public static MySqlDataReader Execute(string query) {
    if(!_initialized) Initialize();
    MySqlCommand cmd = new MySqlCommand(query, _connection);
    return cmd.ExecuteReader();
}

public static MySqlCommand ExecuteCommand(string query) {
    if(!_initialized) Initialize();
    MySqlCommand cmd = new MySqlCommand(query, _connection);
    return cmd;
}

public static MySqlDataReader Execute(string query, params (string, object)[] parameters) {
    if(!_initialized) Initialize();
    MySqlCommand cmd = new MySqlCommand(query, _connection);
    foreach(var tuple in parameters){
        cmd.Parameters.AddWithValue(tuple.Item1, tuple.Item2);
    }
    return cmd.ExecuteReader();
}

public static int ExecuteNonQuery(string query, params (string, object)[] parameters) {
    if(!_initialized) Initialize();
    MySqlCommand cmd = new MySqlCommand(query, _connection);
    foreach(var tuple in parameters){
        cmd.Parameters.AddWithValue(tuple.Item1, tuple.Item2);
    }
    return cmd.ExecuteNonQuery();
}

public static MySqlDataReader Execute(string query, params (string, string[])[] parameters)
{
    if(!_initialized) Initialize();
    MySqlCommand cmd = new MySqlCommand(query, _connection);
    foreach(var tuple in parameters){
        foreach(string val in tuple.Item2){
            cmd.Parameters.AddWithValue(tuple.Item1, val);
        }
    }
    return cmd.ExecuteReader();
}

public static void Execute(string fullQuery, MySqlParameter[] parameters){
    var cmd = new MySqlCommand(fullQuery.ToString(), _connection);
    cmd.Parameters.AddRange(parameters);
    cmd.ExecuteNonQuery();
}
}

```

Terakhir terdapat Kelas 'Seeder' yang digunakan untuk mengisi data pada SQL.

```

public class Seeder{
    public static void PreprocessSidikjari()
    {
        Console.WriteLine("Before:");
        CheckDatabaseContent();

        Database.Initialize();
    }
}

```

```

        Console.WriteLine("Preprocess started...");

        MySqlDataReader reader = Database.Execute("ALTER TABLE sidik_jari ADD COLUMN IF NOT
EXISTS ascii TEXT");
        reader.Close();

        Stopwatch stopwatch = new Stopwatch();
        stopwatch.Start();
        List<SidikJari> sjList = SidikJari.GetAll();
        int counter = 0;
        Parallel.ForEach(sjList, new ParallelOptions { MaxDegreeOfParallelism =
Environment.ProcessorCount }, sj => {
            sj.PopulateImageAscii();
            if(++counter % 1000 == 0) Console.WriteLine(counter+" images converted");
        });
        stopwatch.Stop();
        Console.WriteLine("Ascii conversion finished in "+stopwatch.ElapsedMilliseconds+" ms");

        stopwatch.Restart();
        counter = 0;
        foreach(SidikJari sj in sjList){
            Database.ExecuteNonQuery("INSERT INTO sidik_jari (berkas_citra, nama, ascii) VALUES
(@berkas_citra, @nama, @ascii)",
                ("@berkas_citra", sj.BerkasCitra),
                ("@nama", sj>Nama),
                ("@ascii", sj.Ascii)
            );
            if(++counter % 1000 == 0) Console.WriteLine(counter+" ascii inserted");
        }
        stopwatch.Stop();
        Console.WriteLine("Database update finished in "+stopwatch.ElapsedMilliseconds+" ms");

        Console.WriteLine("Preprocess Finished!");

        Console.WriteLine("After:");
        CheckDatabaseContent();
    }

    public static void CheckDatabaseContent(){
        List<SidikJari> list = SidikJari.GetAll();
        int max = Math.Min(3, list.Count);
        for(int i = 0; i < max; i++){
            Console.WriteLine("==== "+i+" =====");
            Console.WriteLine(list[i].Nama);
            Console.WriteLine(list[i].BerkasCitra);
            Console.WriteLine(list[i].Ascii);
        }
    }

    public static void StartSeeding(string imageFolderPath){
        Database.Initialize();
        Database.ExecuteNonQuery("ALTER TABLE sidik_jari ADD COLUMN IF NOT EXISTS ascii TEXT");
        Database.ExecuteNonQuery("TRUNCATE TABLE biodata");
        Database.ExecuteNonQuery("TRUNCATE TABLE sidik_jari");

        string[] filePathList = Directory.GetFiles(imageFolderPath);

        Console.WriteLine("Seeding started...");
        Stopwatch stopwatch = new Stopwatch();
        stopwatch.Start();

        // Parallel version with max thread = amount of processor
        SidikJari[] sjList = new SidikJari[filePathList.Length];
        int maxThread = Environment.ProcessorCount;
        int unsafeCounter = 0;

```

```

Parallel.For(0, filePathList.Length, new ParallelOptions { MaxDegreeOfParallelism =
maxThread }, i => {
    string file = filePathList[i];
    string name = Path.GetFileNameWithoutExtension(file);
    SidikJari sj = new SidikJari(file, name);
    sjList[i] = sj;

    // below is not accurate because threading
    if(++unsafeCounter % 1000 == 0) Console.WriteLine(unsafeCounter+" images
converted");
});
stopwatch.Stop();
Console.WriteLine("Ascii conversion finished in "+stopwatch.ElapsedMilliseconds+" ms");

stopwatch.Restart();
long count = (long) Math.Pow(10,15);
// Cannot bulk insert because of max_allowed_packet
Reader rdr = new TextReader();
Stack<string> allNames = getCopyOfAllNames(rdr);
Stack<string> allAddress = getCopyOfAllAddress(rdr);
string[] countries = rdr.getContentList("Assets/country.txt");
unsafeCounter = 0;
foreach(SidikJari sj in sjList){
    string name = (allNames.Count != 0)? allNames.Pop() : sj>Nama;
    string address = (allAddress.Count != 0)? allAddress.Pop() : "Twobagoes Esmail";
    string jenis_kelamin = (string)Random.Shared.Choice("Laki-laki", "Perempuan");
    string gol_darah = (string)Random.Shared.Choice("A+", "A-", "B+", "B-", "O+", "O-
", "AB+", "AB-");
    string agama =
(string)Random.Shared.Choice("Islam", "Kristen", "Protestan", "Hindu", "Buddha");
    string status_menikah = (string)Random.Shared.Choice("Belum menikah", "Menikah");
    string pekerjaan = (string)Random.Shared.Choice("Tidak Bekerja",
"Manager", "Consultant", "Software Engineer");
    string country = (string)Random.Shared.Choice(countries);

    Database.ExecuteNonQuery("INSERT INTO sidik_jari (berkas_citra, nama, ascii) VALUES
(@berkas_citra, @nama, @ascii)",
        ("@berkas_citra", sj.BerkasCitra),
        ("@nama", Random.Shared.NextDouble() >= 0.5 ? name : name.ToAlay()),
        ("@ascii", sj.Ascii)
    );

    Database.ExecuteNonQuery("INSERT INTO biodata (NIK, nama, tempat_lahir,
tanggal_lahir, jenis_kelamin, golongan_darah, alamat, agama, status_perkawinan, pekerjaan,
kewarganegaraan) VALUES (@NIK, @nama, @tempat_lahir, @tanggal_lahir, @jenis_kelamin,
@golongan_darah, @alamat, @agama, @status_perkawinan, @pekerjaan, @kewarganegaraan)",
        ("@NIK", (count++).ToString()),
        ("@nama", name),
        ("@tempat_lahir", country),
        ("@tanggal_lahir", new DateOnly()),
        ("@jenis_kelamin", jenis_kelamin),
        ("@golongan_darah", gol_darah),
        ("@alamat", address),
        ("@agama", agama),
        ("@status_perkawinan", status_menikah),
        ("@pekerjaan", pekerjaan),
        ("@kewarganegaraan", country)
    );

    if(++unsafeCounter % 1000 == 0) Console.WriteLine(unsafeCounter+" images inserted");
}
stopwatch.Stop();
Console.WriteLine("Insert database finished in "+stopwatch.ElapsedMilliseconds+" ms");
}

static Stack<string> getCopyOfAllNames(Reader rdr){

```

```

        Stack<string> copy = new();
        string[] temp = rdr.getContentList("Assets/nama.txt");
        foreach(string str in temp) {
            copy.Push(str);
        }
        return copy;
    }

    static Stack<string> getCopyOfAllAddress(Reader rdr){
        Stack<string> copy = new();
        string[] temp = rdr.getContentList("Assets/alamat.txt");
        foreach(string str in temp) {
            copy.Push(str);
        }
        return copy;
    }

    public static void TestAllAscii(){
        Database.Initialize();
        Database.ExecuteNonQuery("CREATE TABLE IF NOT EXISTS test (value TEXT)");

        string value = "";
        for(int i = 0; i < 256; i++) value += (char)i;

        Database.ExecuteNonQuery("INSERT INTO test (value) VALUES(@value)", ("@value", value));

        MySqlDataReader reader = Database.Execute("SELECT * FROM test");
        reader.Read();
        string valToCheck = reader.GetString("value");
        reader.Close();
        if(valToCheck != value) {
            Console.ForegroundColor = ConsoleColor.Red;
            Console.WriteLine("Failed. Please use another MariaDB/MySQL version or use
docker.");
            Console.WriteLine("Value after database SELECT statement: " + valToCheck);
            Console.WriteLine("Hardcoded value: " + value);
            Console.WriteLine();

            for(int i = 0; i < value.Length; i++){
                if(value[i] != valToCheck[i]){
                    Console.WriteLine("Different value at index "+i);
                    Console.WriteLine(value[i] + ": " + (int)value[i]);
                    Console.WriteLine(valToCheck[i] + ": " + (int)valToCheck[i]);
                    break;
                }
            }
            Console.ResetColor();
        } else {
            Console.ForegroundColor = ConsoleColor.Green;
            Console.WriteLine("Passed, you can use this MariaDB/MySQL version.");
            Console.ResetColor();
        }

        Database.ExecuteNonQuery("DROP TABLE test");
    }
}

```

TATA CARA PENGGUNAAN PROGRAM

Jalankan perintah berikut untuk membangun executable.

```
dotnet publish -r win-x64
```

Setelah proses build selesai, Anda dapat menjalankannya dengan mengklik dua kali `bin/Release/publish/Tubes3_SiHashtag.exe`.

Untuk menjalankan database, build dulu docker image

```
docker-compose up --build
```

Setelah selesai, perintah berikut dapat dijalankan untuk mengaktifkan database tanpa perlu melakukan build lagi.

```
docker-compose up
```

Sebelum memulai aplikasi, kita perlu merestore database dengan dump yang ada. Buka terminal baru. Jalankan perintah berikut dan ubah data yang diperlukan.

```
docker exec -i <container id> mariadb -u root -ppassword tubes3_stima24 < Assets/dump/tubes3_stima24_2.sql
```

contohnya

```
docker exec -i 4b4d7661eb39 mariadb -u root -ppassword tubes3_stima24 < Assets/dump/tubes3_stima24.sql
```

`-ppassword` adalah benar tanpa spasi.

Jika menggunakan MariaDB dan dump-nya adalah MySQL, ganti setiap `utf8mb4_0900_ai_ci` dengan `utf8mb4_general_ci` pada file dump.

Untuk mendapatkan Container Id, jalankan perintah berikut.

```
docker ps
```

Pastikan untuk menjalankan preprocess sebelum memulai aplikasi

```
dotnet run preprocess
```

Jika ingin mencoba seeding, jalankan perintah berikut

```
dotnet run seed
```

Jika Anda tidak ingin menggunakan docker, mungkin bisa dilakukan tergantung pada versi MariaDB/MySQL Anda. Jalankan perintah berikut untuk memeriksa apakah kompatibel atau tidak.



```
dotnet run testascii
```

Kemudian jalankan executable atau cukup jalankan perintah berikut. Pastikan .Net terinstal.

```
dotnet run
```

HASIL PENGUJIAN

[Dataset](#) dapat diunduh pada link berikut.

Nama Gambar	Hasil
263__F_Right_middle_finger (Real)	<div>KMP</div>  <p> Nama: Andryan Dwi Cahyono NIK: 1000000000001807 Tempat Lahir: Mozambique Tanggal Lahir: 01/01/2001 00:00:00 Jenis Kelamin: Laki-Laki Golongan Darah: A- Alamat: Twobagoes Esmail Agama: Protestan Status Perkawinan: Belum Menikah Pekerjaan: Tidak Bekerja Kewarganegaraan: Mozambique Algoritma: KMP </p> <p> Waktu Pencarian: 87 ms </p> <p> Algorithm <input checked="" type="radio"/> KMP </p> <p> Persentase Kecocokan: 100% </p> <div>BM</div>  <p> Nama: Andryan Dwi Cahyono NIK: 1000000000001807 Tempat Lahir: Mozambique Tanggal Lahir: 01/01/2001 00:00:00 Jenis Kelamin: Laki-Laki Golongan Darah: A- Alamat: Twobagoes Esmail Agama: Protestan Status Perkawinan: Belum Menikah Pekerjaan: Tidak Bekerja Kewarganegaraan: Mozambique Algoritma: BM </p> <p> Waktu Pencarian: 76 ms </p> <p> Algorithm <input checked="" type="radio"/> BM </p> <p> Persentase Kecocokan: 100% </p>

247_M_Left_index_finger (Real)

KMP



Nama: Ageng Kurniawan
NIK: 1000000000001620
Tempat Lahir: Turkmenistan
Tanggal Lahir: 01/01/2001 00:00:00
Jenis Kelamin: Laki-Laki
Golongan Darah: A-
Alamat: Twobagoes Esmail
Agama: Protestan
Status Perkawinan: Belum Menikah
Pekerjaan: Consultant
Kewarganegaraan: Turkmenistan
Algoritma: KMP

Search

Waktu Pencarian:

87 ms

Algorithm

☒ KMP

Persentase Kecocokan:

100%

BM



Nama: Ageng Kurniawan
NIK: 1000000000001620
Tempat Lahir: Turkmenistan
Tanggal Lahir: 01/01/2001 00:00:00
Jenis Kelamin: Laki-Laki
Golongan Darah: A-
Alamat: Twobagoes Esmail
Agama: Protestan
Status Perkawinan: Belum Menikah
Pekerjaan: Consultant
Kewarganegaraan: Turkmenistan
Algoritma: BM

Search

Waktu Pencarian:

75 ms

Algorithm


☒ BM

Persentase Kecocokan:

100%

234_M_Right_little_finger (Real)

KMP



Nama: Ahmad Sapudin
NIK: 100000000001486
Tempat Lahir: Virgin Islands
Tanggal Lahir: 01/01/2001 00:00:00
Jenis Kelamin: Perempuan
Golongan Darah: O+
Alamat: Twobagoes Esmail
Agama: Kristen
Status Perkawinan: Belum Menikah
Pekerjaan: Manager
Kewarganegaraan: Virgin Islands
Algoritma: KMP


Search

Waktu Pencarian:
90 ms

Algorithm
☒ KMP

Persentase Kecocokan:
100%

BM



Nama: Ahmad Sapudin
NIK: 100000000001486
Tempat Lahir: Virgin Islands
Tanggal Lahir: 01/01/2001 00:00:00
Jenis Kelamin: Perempuan
Golongan Darah: O+
Alamat: Twobagoes Esmail
Agama: Kristen
Status Perkawinan: Belum Menikah
Pekerjaan: Manager
Kewarganegaraan: Virgin Islands
Algoritma: BM

Search

Waktu Pencarian:
75 ms

Algorithm
☒ BM

Persentase Kecocokan:
100%



100_M_Left_little_finger_Zcut
(Altered)

Levenstein

Pilih Citra

Chosen Image:
100_M_Left_little_finger_Zcut.BMP

Source:
file:///C:/Users/Asus/Documents/Heking/Tingkat 2/Semester Ge



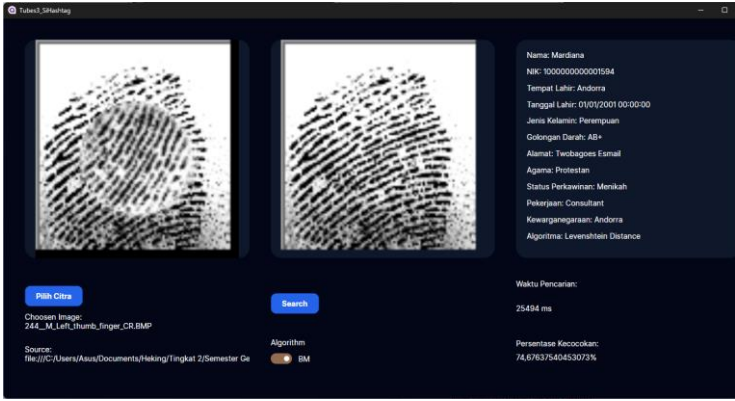
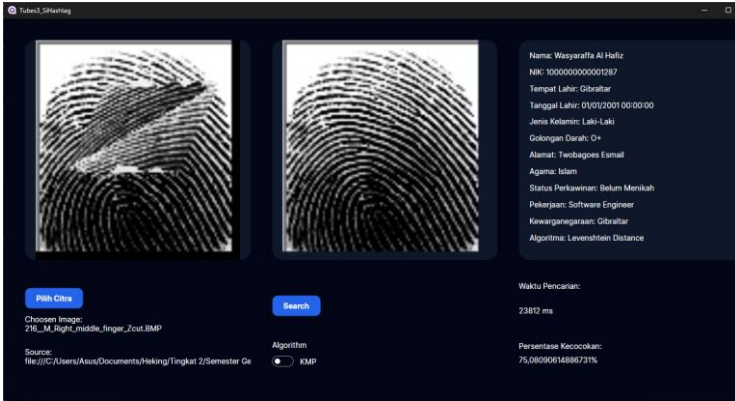
Search

Waktu Pencarian:
25909 ms

Algorithm
☒ BM

Persentase Kecocokan:
78,64077669802912%

Nama: Sukandi
NIK: 1000000000000001
Tempat Lahir: Saint Martin (French part)
Tanggal Lahir: 01/01/2001 00:00:00
Jenis Kelamin: Perempuan
Golongan Darah: A+
Alamat: 4538 Us Hwy 231
Agama: Hindu
Status Perkawinan: Belum Menikah
Pekerjaan: Consultant
Kewarganegaraan: Saint Martin (French part)
Algoritma: Levenshtein Distance

<p>244_M_Left_little_finger_CR (Altered)</p>	<p>Levenstein</p> 
<p>2166_M_Right_middle_finger_Z cut (Altered)</p>	<p>Levenstein</p> 

ANALISIS HASIL PENGUJIAN

Berdasarkan hasil pengujian, dapat terlihat bahwa proses membandingkan sidik jari dengan menggunakan algoritma KMP dan BM sangatlah cepat, sekitar 70 sampai 100 ms pada pengujian-pengujian yang dilakukan. Waktu pemrosesan yang cepat ini sejalan dengan kompleksitas waktu keduanya yakni $O(m+n)$ pada KMP dan $O(n)$ untuk *average case* BM. Selain itu, perbandingan KMP dan BM pada algoritma ini juga hanya dilakukan dengan membandingkan potongan kecil sidik jari dan satu sidik jari utuh.

Walaupun begitu terdapat kasus lain yang akan memakan waktu cukup lama yaitu ketika tidak ditemukan sidik jari yang sepenuhnya sama. Jika kasus ini terjadi, akan dilakukan proses membandingkan kembali dengan algoritma *Levenshtein Distance*. Proses algoritma ini memakan waktu yang cukup lama sekitar 20 sampai 30 detik. Hal ini disebabkan karena perbandingan pada *Levenshtein Distance* akan melakukan perbandingan pada gambar utuh dan kompleksitas waktunya cukup besar yakni, $O(mn)$.

V. KESIMPULAN, SARAN, TANGGAPAN, DAN REFLEKSI

Algoritma Knuth-Morris-Pratt (KMP) dan Boyer-Moore adalah dua teknik pencarian string yang sangat efisien dan telah terbukti efektif dalam mengatasi masalah pencarian pola dalam teks. Algoritma KMP menghindari perbandingan ulang yang tidak perlu dengan memanfaatkan array lps (longest prefix suffix), yang memungkinkan algoritma untuk melompati sejumlah karakter dalam teks. Di sisi lain, algoritma Boyer-Moore menggunakan informasi dari pola untuk melompati sejumlah karakter dalam teks, mengurangi jumlah perbandingan yang diperlukan.

Untuk aplikasi praktis, sangat disarankan untuk memilih algoritma pencarian string berdasarkan karakteristik data dan kebutuhan spesifik dari tugas pencarian. Algoritma KMP sangat cocok untuk situasi di mana pola memiliki banyak pengulangan, sedangkan algoritma Boyer-Moore lebih efisien dalam kasus di mana pola relatif lebih panjang dan teks yang sedang dicari memiliki banyak karakter unik. Selain itu, mengkombinasikan kedua algoritma ini atau menggunakan variasi lain seperti algoritma Rabin-Karp dapat memberikan hasil yang lebih optimal dalam berbagai konteks pencarian.

Tanggapan terhadap implementasi algoritma KMP dan Boyer-Moore menunjukkan bahwa kedua algoritma ini sangat dihargai dalam komunitas pengembangan perangkat lunak dan penelitian komputer. Keduanya telah menjadi dasar dalam pengembangan berbagai aplikasi, mulai dari pencarian teks sederhana hingga analisis data yang kompleks. Implementasi yang tepat dari algoritma ini dapat menghemat waktu dan sumber daya komputasi secara signifikan, yang merupakan kebutuhan penting dalam era data besar dan komputasi cepat saat ini.

Dalam mengerjakan tugas besar ini, kami mendapatkan banyak pelajaran yang berharga. Pelajaran – pelajaran tersebut, kami yakin, dapat membantu kami untuk menempuh perkuliahan di Teknik Informatika Institut Teknologi Bandung. Pelajaran yang kami dapatkan adalah kami dapat memahami bagaimana algoritma KMP serta Boyer-Moore bekerja alih – alih hanya dengan menggunakannya saja. Pemahaman tersebut dapat kami gunakan sebagai landasan untuk pengetahuan serta keterampilan kami. Kami juga dapat memahami bagaimana untuk membangun aplikasi desktop menggunakan bahasa C#.

Masih ada pelajaran – pelajaran lainnya yang tidak bisa kami sebutkan satu – satu. Kesalahan yang kami lakukan murni datang dari kekurangan dan kemampuan kami. Oleh karena itu, kami akan terus berusaha hingga segala kekurangan yang kami miliki tidak akan menjadi hambatan untuk hari yang akan datang.

VI. LAMPIRAN

Repository:

https://github.com/raflyhanga/Tubes3_SiHashtag

Video:

https://youtu.be/D8pgiHEY_s4?si=bnjgZ7ZQjvJq4uCP

VII. DAFTAR PUSTAKA

Munir, R. (t.thn.). *Strategi Algoritma*. Diambil kembali dari Homepage Rinaldi Munir:
<https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2023-2024/stima23-24.htm>