

Laporan Tugas Besar 3
IF2211 Strategi Algoritma
Pemanfaatan *Pattern Matching* dalam Membangun
Sistem Deteksi Individu Berbasis Biometrik Melalui
Citra Sidik Jari



Disusun oleh:
Yudi Kurniawan (10023634)
Maximilian Sulistiyo (13522061)
Marvel Pangondian (13522075)
Steven Tjhia (13522103)

Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung
2023

Daftar Isi

| | |
|--|-----------|
| Program Studi Teknik Informatika | |
| Sekolah Teknik Elektro dan Informatika | |
| Institut Teknologi Bandung | |
| 2023..... | 1 |
| Daftar Isi..... | 2 |
| Bab I | |
| Deskripsi Masalah..... | 3 |
| Bab II | |
| Landasan Teori..... | 5 |
| Bab III | |
| Analisis Pemecahan Masalah..... | 11 |
| 3.1 Ekstraksi Keunikan Sidik Jari dalam Bentuk String..... | 11 |
| 3.2 Algoritma Boyer-Moore (BM)..... | 13 |
| 3.3 Algoritma Knuth-Morris-Pratt (KMP)..... | 18 |
| 3.4 Algoritma Longest Common Subsequence (LCS)..... | 24 |
| Bab IV | |
| Implementasi dan Pengujian..... | 27 |
| 4.1 Spesifikasi Teknis Program..... | 27 |
| 4.1.1 Kelas Utama..... | 27 |
| 4.1.2 Metode Kelas..... | 29 |
| 4.2 Tata Cara Penggunaan..... | 35 |
| 4.3 Pengujian..... | 36 |
| 4.4 Analisis Hasil Pengujian..... | 39 |
| Bab V | |
| Kesimpulan dan Saran..... | 40 |
| 5.1 Kesimpulan..... | 40 |
| 5.2 Saran..... | 40 |
| Lampiran..... | 42 |
| Github Repository..... | 42 |
| Video..... | 42 |
| Daftar Pustaka..... | 43 |

Bab I

Deskripsi Masalah



Gambar 1.1 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, Anda diminta untuk mengimplementasikan sistem yang dapat melakukan identifikasi individu berbasis biometrik dengan menggunakan sidik jari. Metode yang akan digunakan untuk melakukan deteksi sidik jari adalah Boyer-Moore dan Knuth-Morris-Pratt. Selain itu, sistem ini akan dihubungkan dengan identitas sebuah individu melalui basis data sehingga harapannya terbentuk sebuah sistem yang dapat mengenali identitas seseorang secara lengkap hanya dengan menggunakan sidik jari.

Bab II

Landasan Teori

Pattern matching adalah proses pencarian pola atau *substring* dalam sebuah string yang lebih besar. Ini merupakan masalah fundamental dalam pengolahan teks dan memiliki banyak aplikasi praktis, seperti pengeditan teks, pemrograman *compiler*, pencarian dalam basis data, dan analisis sekuensial DNA. Terdapat beberapa algoritma yang dirancang untuk menangani masalah ini, di antaranya adalah algoritma Boyer-Moore dan Knuth-Morris-Pratt.

Algoritma Boyer-Moore, yang dikembangkan oleh Robert S. Boyer dan J. Strother Moore pada tahun 1977, adalah algoritma pencocokan pola yang sangat efisien untuk kasus di mana ragam alfabet yang digunakan relatif banyak, sehingga akan lebih cocok jika diterapkan pada pencarian text dengan ASCII daripada *binary*. Algoritma ini dirancang untuk meminimalkan jumlah perbandingan karakter yang diperlukan dengan memanfaatkan informasi yang diperoleh dari pola itu sendiri.

Konsep utama algoritma Boyer-Moore adalah sebagai berikut:

1. Pra Pemrosesan Pola: Sebelum pencarian, algoritma melakukan pra pemrosesan pada pola untuk membangun dua tabel: tabel pencarian balik (bad character shift) dan tabel good suffix shift. Tabel pencarian balik menyimpan informasi tentang kemunculan terakhir setiap karakter dalam pola (*last occurrence function*), sementara tabel good suffix menyimpan informasi tentang pergeseran maksimum yang dapat dilakukan saat terjadi ketidakcocokan dengan substring dalam string.
2. Pencarian dari Kanan ke Kiri (*looking glass*): Berbeda dengan algoritma pencocokan pola sederhana yang mencari dari kiri ke kanan, algoritma Boyer-Moore mencari dari kanan ke kiri. Ini memungkinkan algoritma untuk memanfaatkan informasi dari tabel pencarian balik dan tabel good suffix shift untuk menentukan berapa banyak string yang dapat digeser saat terjadi ketidakcocokan.
3. Pergeseran Cerdas (*character jump*): Jika terjadi ketidakcocokan antara pola dan substring dalam string, algoritma menggunakan informasi dari tabel pencarian balik dan tabel good suffix shift untuk menentukan seberapa jauh pola harus digeser ke kanan.

Pergeseran ini dilakukan dengan cara yang cerdas untuk meminimalkan jumlah perbandingan karakter yang diperlukan.

4. Penanganan Kasus Khusus: Algoritma Boyer-Moore juga menangani kasus-kasus khusus seperti pola yang mengandung karakter yang berulang dan pola yang memiliki prefiks yang sama dengan sufiks.

Kompleksitas waktu terburuk algoritma Boyer-Moore adalah $O(nm + A)$, di mana n adalah panjang string, m adalah panjang pola, dan A adalah ragam alfabet.

Algoritma Knuth-Morris-Pratt (KMP) dikembangkan oleh Donald Knuth, James H. Morris, dan Vaughan Pratt pada tahun 1977. Algoritma ini merupakan salah satu algoritma pencarian pola yang paling efisien dan banyak digunakan, terutama dalam kasus di mana pola memiliki banyak pengulangan *substring*.

Algoritma KMP bekerja dengan cara membandingkan pola dengan teks dari kiri ke kanan. Jika terjadi ketidakcocokan, algoritma ini menggunakan informasi yang telah diperoleh sebelumnya dari pola untuk menghindari pengulangan pengecekan karakter yang tidak perlu. Untuk mencapai hal ini, algoritma KMP menggunakan konsep "*Longest Proper Prefix that is also Suffix*" (LPS) atau "Awalan Terpanjang yang juga Akhiran" dari pola. LPS menyimpan informasi tentang berapa banyak karakter yang harus dilompati jika terjadi ketidakcocokan antara pola dan teks.

Langkah-langkah Algoritma KMP:

1. Preprocessing Pola (*border function*): Sebelum melakukan pencarian pola pada teks, algoritma KMP melakukan preprocessing pada pola untuk membangun tabel LPS. Tabel ini akan menyimpan nilai LPS untuk setiap posisi pada pola.
2. Pencarian Pola pada Teks: Setelah tabel LPS dibangun, algoritma KMP melakukan pencarian pola pada teks. Jika terjadi ketidakcocokan antara pola dan teks, algoritma akan menggunakan informasi dari tabel LPS untuk melompati sebagian karakter pada pola dan teks yang tidak perlu diperiksa kembali.

Algoritma Knuth-Morris-Pratt memiliki beberapa keunggulan, antara lain:

1. Efisiensi yang Tinggi: Dengan menggunakan informasi dari tabel LPS, algoritma KMP dapat melompati banyak karakter yang tidak perlu diperiksa, sehingga meningkatkan efisiensi pencarian.
2. Kinerja yang Baik pada Pola dengan Banyak Pengulangan: Algoritma ini bekerja dengan sangat baik pada pola yang memiliki banyak pengulangan substring karena dapat memanfaatkan informasi dari tabel LPS untuk melompati bagian yang tidak cocok.
3. Penggunaan Memori yang Efisien: Algoritma KMP hanya membutuhkan ruang memori yang konstan, tidak bergantung pada panjang pola atau teks.
4. Aplikasi yang Luas: Algoritma ini banyak digunakan dalam berbagai aplikasi, seperti pemrosesan teks, pencarian kata pada mesin pencari, dan analisis DNA.

Kompleksitas waktu untuk algoritma Knuth-Morris-Pratt adalah $O(m+n)$.

Pemilihan algoritma yang tepat tergantung pada kebutuhan aplikasi dan karakteristik data yang akan diproses. Dalam beberapa kasus, kombinasi dari kedua algoritma ini atau modifikasi tertentu dapat digunakan untuk meningkatkan kinerja dan fleksibilitas pencocokan pola.

Konsep *Longest Common Subsequence* (LCS) merupakan salah satu topik fundamental dalam ranah algoritma dan struktur data. Secara sederhana, LCS adalah masalah pencarian urutan terpanjang yang sama antara dua atau lebih string. Masalah ini memiliki aplikasi yang luas dalam berbagai bidang, seperti bioinformatika (untuk menyelaraskan urutan DNA atau protein), pemrosesan sinyal digital, dan kompresi data. Dalam penjelasan ini, kita akan membahas dasar teori LCS secara komprehensif.

Definisi Formal LCS

Secara formal, LCS didefinisikan sebagai berikut: diberikan dua string X dan Y, $LCS(X, Y)$ adalah urutan terpanjang Z sedemikian sehingga Z adalah subsekuensi dari kedua X dan Y. Subsekuensi di sini berarti bahwa urutan karakter Z muncul di X dan Y dengan urutan yang sama, meskipun tidak harus berurutan.

Contoh:

- Jika $X = "ABCBDAB"$ dan $Y = "BDCABA"$, maka $LCS(X, Y) = "BCBA"$ atau $"BDAB"$.
- Jika $X = "ACADEB"$ dan $Y = "CBAGC"$, maka $LCS(X, Y) = "ACB"$.

Pendekatan Pemrograman Dinamis

Salah satu pendekatan yang paling umum untuk menyelesaikan masalah LCS adalah dengan menggunakan teknik pemrograman dinamis. Pemrograman dinamis adalah metode untuk memecahkan masalah kompleks dengan memecahnya menjadi submasalah yang lebih sederhana, dan menyimpan hasil dari sub masalah tersebut untuk digunakan kembali jika diperlukan. Hal ini membantu menghindari perhitungan yang berulang dan meningkatkan efisiensi waktu.

Algoritma LCS menggunakan matriks dua dimensi untuk menyimpan panjang LCS dari setiap sub masalah. Matriks ini diisi secara bertahap dengan menggunakan nilai-nilai yang dihitung dari sub masalah sebelumnya. Proses ini dilakukan dengan membandingkan setiap pasangan karakter dari kedua string dan mengikuti aturan berikut:

- Jika karakter saat ini sama pada kedua string, nilai LCS untuk submasalah tersebut adalah 1 ditambah dengan nilai LCS dari sub masalah sebelumnya.
- Jika karakter saat ini berbeda pada kedua string, nilai LCS untuk submasalah tersebut adalah nilai maksimum antara LCS dari sub masalah sebelumnya dengan mengabaikan karakter pertama dari string pertama atau karakter pertama dari string kedua.

Setelah matriks terisi, nilai LCS terpanjang akan terletak di sudut kanan bawah matriks.

Kompleksitas Waktu dan Ruang

Kompleksitas waktu dari algoritma LCS dengan pemrograman dinamis adalah $O(mn)$, di mana m dan n adalah panjang masing-masing string. Ini cukup efisien untuk kebanyakan kasus, terutama jika kedua string memiliki panjang yang wajar. Kompleksitas ruang dari algoritma ini adalah $O(mn)$, karena kita membutuhkan matriks dua dimensi untuk menyimpan nilai-nilai LCS dari setiap submasalah. Dalam beberapa kasus, ini dapat menjadi masalah jika string yang diberikan sangat panjang dan memori yang tersedia terbatas.

Varian dan Perbaikan

Terdapat beberapa varian dan perbaikan dari algoritma LCS dasar yang dapat meningkatkan efisiensi atau memperluas fungsionalitasnya:

- a. LCS dengan Pencatatan Arah: Dalam implementasi dasar, algoritma hanya menghitung panjang LCS. Dengan sedikit modifikasi, kita dapat mencatat arah (kiri, atas, atau diagonal) di setiap sel matriks, sehingga memungkinkan untuk membangun kembali urutan LCS sebenarnya setelah perhitungan selesai.
- b. Algoritma Hirschberg: Algoritma Hirschberg adalah varian dari algoritma LCS yang menggunakan pendekatan "divide and conquer" untuk mengurangi kebutuhan ruang menjadi $O(\min(m, n))$. Ini sangat berguna saat kita bekerja dengan string yang sangat panjang dan memori yang terbatas.
- c. LCS dengan Penalti Celah: Dalam beberapa kasus, kita mungkin ingin memberikan penalti untuk celah (gap) dalam urutan LCS. Ini dapat dilakukan dengan memodifikasi aturan pengisian matriks untuk memperhitungkan penalti celah.
- d. LCS Berganda: LCS berganda adalah pengembangan dari masalah LCS di mana kita mencari urutan terpanjang yang sama di antara tiga atau lebih string. Ini memiliki aplikasi dalam penyelarasan urutan DNA atau protein dari beberapa spesies.

Aplikasi LCS

Masalah LCS dan variannya memiliki aplikasi yang luas dalam berbagai bidang, seperti:

- a. Bioinformatika: LCS digunakan untuk menyelaraskan urutan DNA atau protein dari spesies yang berbeda, membantu mengidentifikasi kesamaan dan perbedaan evolusioner.
- b. Pemrosesan Sinyal Digital: LCS dapat digunakan untuk mendeteksi dan menghapus redundansi dalam sinyal digital, yang membantu dalam kompresi data.
- c. Analisis Teks: LCS dapat digunakan untuk mendeteksi kesamaan antara dua dokumen teks, yang berguna dalam pendekripsi plagiarisme atau pencarian teks.
- d. Pengenalan Pola: LCS dapat digunakan dalam algoritma pengenalan pola untuk membandingkan pola yang diamati dengan pola referensi.

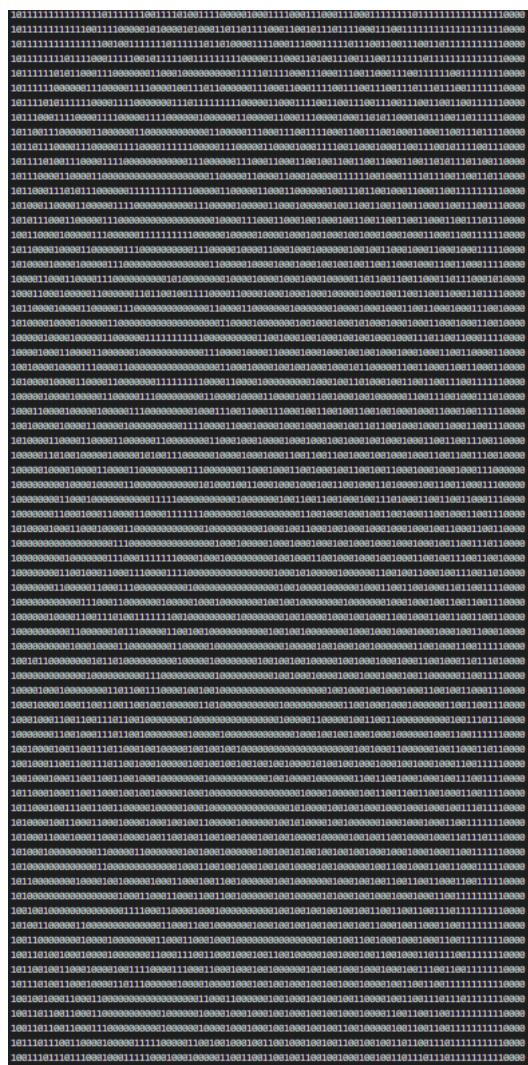
- e. Pemrograman Komputer: LCS digunakan dalam alat bantu pengembangan seperti alat penggabungan (merge) dalam pengendalian versi kode sumber.

Bab III

Analisis Pemecahan Masalah

3.1 Ekstraksi Keunikan Sidik Jari dalam Bentuk String

Proses ini dimulai dengan mengubah masukan sidik jari dari file BMP menjadi format biner. Biner tersebut didasarkan pada pola terang-gelap sidik jari, dengan nilai 0 menandakan warna hitam dan 1 menandakan warna putih. Berikut adalah contoh hasil dari konversi tersebut:

A large block of binary code representing a fingerprint image. The code consists of a grid of 0s and 1s, where 0 represents black pixels and 1 represents white pixels. The pattern is highly complex and organic, characteristic of a fingerprint.

Gambar 3.1.1 Contoh hasil biner sidik jari

Setelah file BMP diproses menjadi format biner, langkah selanjutnya adalah memilih 30×8 bit yang akan dijadikan representasi keunikan sidik jari. 30×8 bit ini berada antara 3 lokasi yakni di $(\frac{1}{4} * \text{tinggi_BMP})$, $(\frac{1}{2} * \text{tinggi_BMP})$, atau $(\frac{3}{4} * \text{tinggi_BMP})$. Dari antara 3 lokasi tersebut, akan dipilih lokasi yang menghasilkan pola dengan homogenitas terendah

"Homogenitas terendah" berarti memilih blok dengan jumlah bit 1 dan bit 0 yang paling seimbang, hal ini untuk menghindari pemilihan blok dengan nilai 1 atau 0 semua. Berikut contoh blok yang dipilih:

```
011000000010111100011100111111011011111111111111111111  
0001111111000011110011110111110111110011110111111111111  
111111111111111111111100011111011101001110111111111111111  
110100111100000100011110001110001110001111111011111111111  
000101000110110111100011001011101111100011100111111111111  
011111101101000111100011100011111011100110011100110111111  
111001111111110000011100011010011100111001111110111111111  
001000000000111101111000111000111001110011111100111111001111
```

Gambar 3.1.2 Contoh blok pola sidik jari.

Representasi ini akan dijadikan sebagai pola dalam proses pencocokan string. Blok tersebut kemudian diubah menjadi sebuah string ASCII berukuran 30 karakter. Cara transformasi sebuah blok binary menjadi sebuah string ASCII adalah dengan merepresentasikan setiap bit pada koordinat (i,j) sebagai karakter tersendiri. Untuk bit (i,j) dapat dijadikan karakter dengan menggabungkan bit tersebut bersama dengan bit $(i,j + 1)$ sampai $(i, j+ 7)$, hasilnya ada sebuah string 8 bit yang dapat diubah menjadi satu karakter ASCII. Pada gambar 3.1.2, berarti karakter pertama adalah karakter dengan representasi bit 00110010 ("2"). Transformasi blok tersebut adalah sebagai berikut:

```
2¶§ | dnvrv~j| *«|ùíílw3?UIäf;ñûI
```

Gambar 3.1.3 Hasil ASCII pola sidik jari

Langkah berikutnya adalah mencari sidik jari pada database yang cocok dengan masukan sidik jari pengguna. Proses ini dimulai dengan mengubah gambar sidik jari dalam database menjadi format biner. Hasil biner tersebut kemudian diproses menjadi format ASCII sesuai dengan proses pengubahan blok ke string ASCII di atas, hanya saja proses ini dilakukan untuk setiap baris sampai baris - 7 sebab untuk baris -7 dan baris setelahnya tidak memiliki tinggi yang cukup untuk membentuk karakter ASCII (sebuah karakter ASCII terdiri dari 8 bit).

Gambar 3.1.4 Hasil Potongan ASCII sidik jari

Dengan mengubah representasi biner ke ASCII, kita dapat memanfaatkan algoritma pencocokan string yang lebih cepat dalam menemukan kecocokan antara sidik jari pengguna dan sidik jari dalam database. Ini karena representasi ASCII memungkinkan penggunaan algoritma pencocokan teks yang optimal dan telah teruji, seperti Boyer-Moore atau Knuth-Morris-Pratt, yang dapat memproses data dengan cepat dan efisien.

3.2 Algoritma Boyer-Moore (BM)

Algoritma Boyer-Moore (BM) adalah algoritma pencarian *string* yang efisien dan sering digunakan dalam berbagai aplikasi, termasuk identifikasi sidik jari. Algoritma ini bekerja dengan membandingkan pola (sidik jari yang ingin diidentifikasi) yang sudah diubah menjadi ASCII dengan teks (basis data sidik jari) dari kanan ke kiri. Konsep utama di balik algoritma BM adalah melakukan lompatan (*character jump*) yang besar saat terjadi ketidakcocokan antara pola dan teks, sehingga dapat meningkatkan efisiensi pencarian.

Cara Kerja Algoritma Boyer-Moore

1. Inisialisasi Tabel Preprocessing

Algoritma BM membutuhkan dua tabel *preprocessing*: tabel *bad character* dan tabel *good suffix*.

- Tabel *Bad Character*: Tabel *bad character* menyimpan informasi tentang posisi terjauh yang dapat dilompati jika ada ketidakcocokan karakter pada pola dengan teks. Untuk setiap karakter dalam alfabet, tabel ini menyimpan jarak maksimum yang dapat dilompati jika karakter tersebut tidak cocok dengan karakter pada pola.
- Tabel *Good Suffix*: Tabel *good suffix* menyimpan informasi tentang kemunculan suffix terpanjang dari pola yang juga merupakan prefix dari pola tersebut. Ini memungkinkan algoritma untuk melompat lebih jauh saat terjadi ketidakcocokan. Tabel ini menyimpan informasi tentang jarak yang dapat dilompati jika terjadi ketidakcocokan dengan suffix tertentu dari pola.

2. Pencocokan Pola dengan Teks

Setelah tabel preprocessing diinisialisasi, algoritma BM mulai mencocokan pola dengan teks dari kanan ke kiri (*looking glass*).

- Bandingkan setiap karakter pada pola dengan karakter yang sesuai pada teks, dari kanan ke kiri.
- Jika semua karakter cocok, pola ditemukan dalam teks.
- Jika terjadi ketidakcocokan:
 - I. Gunakan tabel bad character untuk menentukan jarak lompatan berdasarkan karakter yang tidak cocok.
 - II. Gunakan tabel good suffix untuk menentukan jarak lompatan berdasarkan suffix terpanjang dari pola yang juga merupakan prefix.

III. Pilih jarak lompatan terbesar dari tabel bad character dan tabel good suffix.

IV. Lompat ke posisi baru di teks sesuai dengan jarak lompatan yang dipilih.

3. Implementasi dalam Sistem Identifikasi Sidik Jari

Dalam sistem identifikasi sidik jari, algoritma BM dapat diimplementasikan dengan langkah-langkah berikut:

- a. Representasikan pola sidik jari sebagai *string*.
- b. Inisialisasi tabel bad character dan tabel good suffix untuk pola sidik jari.
- c. Gunakan algoritma BM untuk mencari pola sidik jari dalam basis data sidik jari (teks).
- d. Jika pola ditemukan, kembalikan identitas individu yang sesuai dengan pola sidik jari tersebut.

Kelebihan Algoritma Boyer-Moore

1. Efisien untuk Pola yang Panjang

Algoritma BM sangat efisien untuk pola yang panjang, karena dapat melakukan lompatan yang besar saat terjadi ketidakcocokan. Semakin panjang pola, semakin besar kemungkinan untuk melakukan lompatan yang signifikan, sehingga meningkatkan kecepatan pencarian.

2. Pencarian Cepat dengan Langkah Melompat (*character jump*)

Algoritma BM memanfaatkan konsep lompatan untuk meminimalkan jumlah perbandingan yang dilakukan. Dengan melompat ke posisi yang lebih jauh di teks, algoritma dapat menghindari perbandingan yang tidak diperlukan, sehingga mempercepat proses pencarian.

3. Fleksibilitas dalam Pencarian

Algoritma BM dapat digunakan untuk mencari pola dalam teks dengan ukuran yang besar atau bahkan dalam aliran data yang terus berkembang. Ini memungkinkan algoritma untuk digunakan dalam berbagai aplikasi, termasuk pemrosesan data real-time dan analisis data besar.

Kekurangan Algoritma Boyer-Moore

1. Membutuhkan Ruang Tambahan untuk Tabel Preprocessing

Algoritma BM membutuhkan ruang tambahan untuk menyimpan tabel *bad character* dan tabel *good suffix*. Meskipun ukuran tabel ini tergantung pada panjang pola dan ukuran alfabet, namun dalam kasus tertentu, seperti pola yang sangat panjang atau alfabet yang besar, kebutuhan ruang tambahan dapat menjadi masalah.

2. Kurang Efisien untuk Pola yang Pendek atau Banyak Karakter yang Berulang

Algoritma BM kurang efisien untuk pola yang pendek atau pola yang memiliki banyak karakter yang berulang. Dalam kasus ini, lompatan yang dilakukan mungkin tidak terlalu besar, sehingga mengurangi keunggulan utama dari algoritma ini.

3. Waktu Preprocessing yang Lebih Lama

Inisialisasi tabel *bad character* dan tabel *good suffix* membutuhkan waktu *preprocessing* yang lebih lama dibandingkan dengan algoritma pencarian string lainnya, seperti Knuth-Morris-Pratt (KMP). Meskipun waktu preprocessing ini hanya dilakukan sekali, namun dapat menjadi masalah jika pola sering berubah atau diperbarui.

Analisis Kompleksitas

1. Kompleksitas Waktu

Kasus Terburuk: Kompleksitas waktu algoritma BM dalam kasus terburuk adalah $O(nm + A)$, di mana n adalah panjang teks, m adalah panjang pola, dan A adalah ragam alfabet pada pola. Ini terjadi ketika pola tidak ditemukan dalam teks dan algoritma harus membandingkan setiap karakter dalam teks.

2. Kompleksitas Ruang

Kompleksitas ruang algoritma BM adalah $O(m + A)$, di mana m adalah panjang pola dan A adalah ukuran alfabet (misalnya, 256 untuk karakter ASCII). Ini dibutuhkan untuk menyimpan tabel bad character dan tabel good suffix. Dalam kasus pola yang sangat panjang atau alfabet yang besar, kebutuhan ruang tambahan ini dapat menjadi masalah.

Optimasi Algoritma Boyer-Moore

Meskipun algoritma Boyer-Moore (BM) sudah cukup efisien, ada beberapa teknik optimasi yang dapat diterapkan untuk meningkatkan kinerjanya lebih lanjut:

1. Penyederhanaan Tabel Good Suffix

Dalam beberapa kasus, tabel good suffix dapat disederhanakan dengan hanya menyimpan informasi suffix terpanjang yang juga merupakan prefix. Dengan demikian, kita dapat mengurangi ukuran tabel good suffix dan mempercepat proses pencarian dalam tabel tersebut.

Implementasi dalam Sistem Identifikasi Sidik Jari

Dalam konteks sistem identifikasi sidik jari, algoritma Boyer-Moore dapat diimplementasikan dengan langkah-langkah berikut:

1. Representasi Pola Sidik Jari sebagai String.

Gambar sidik jari diekstraksi menjadi *binary* dan direpresentasikan dalam bentuk ASCII.

2. Inisialisasi Tabel Preprocessing

Sebelum menjalankan algoritma BM, kita perlu menginisialisasi tabel *bad character* dan tabel *good suffix* untuk pola sidik jari yang direpresentasikan sebagai *string*.

3. Pencarian Pola dalam Basis Data.

Dengan tabel *preprocessing* yang telah diinisialisasi, kita dapat menerapkan algoritma BM untuk mencari pola sidik jari dalam basis data sidik jari (teks). Basis data ini juga harus direpresentasikan dalam bentuk string dengan cara yang sama seperti pola sidik jari.

4. Pencocokan dan Identifikasi Individu.

Jika pola sidik jari ditemukan dalam basis data, algoritma BM akan memberikan lokasi pola tersebut dalam teks. Kemudian, kita dapat menggunakan informasi ini untuk mengidentifikasi individu yang sesuai dengan pola sidik jari tersebut dalam basis data.

5. Penanganan Kasus Khusus.

Dalam sistem identifikasi sidik jari, kita mungkin tidak mendapatkan *exact match* dengan algoritma BM. Dalam situasi seperti ini, kita dapat menggunakan teknik tambahan seperti pencocokan pola yang lebih fleksibel atau penggunaan algoritma lain seperti *Longest Common Subsequence* (LCS) untuk mengukur kesamaan antara pola sidik jari.

Dengan mengimplementasikan algoritma Boyer-Moore dalam sistem identifikasi sidik jari, kita dapat memanfaatkan keunggulan utama algoritma ini, yaitu efisiensi pencarian untuk pola yang panjang dan kemampuan melakukan lompatan yang signifikan saat terjadi ketidakcocokan. Hal ini dapat membantu meningkatkan kecepatan dan akurasi proses identifikasi individu berdasarkan sidik jari mereka.

3.3 Algoritma Knuth-Morris-Pratt (KMP)

Algoritma Knuth-Morris-Pratt (KMP) adalah algoritma pencarian string yang efisien dan sering digunakan dalam berbagai aplikasi, termasuk identifikasi sidik jari. Algoritma ini bekerja dengan membandingkan pola (sidik jari yang ingin diidentifikasi) dengan teks (basis data sidik jari) dari kiri ke kanan. Konsep utama di balik algoritma KMP adalah memanfaatkan informasi tentang prefix dan suffix yang sama dalam pola untuk melakukan lompatan yang efisien saat terjadi ketidakcocokan.

Cara Kerja Algoritma KMP

1. Inisialisasi Tabel Lompatan (Jump Table).

Sebelum memulai pencarian, algoritma KMP membutuhkan inisialisasi tabel lompatan (jump table) dengan menggunakan fungsi pencarian nilai awalan (prefix) terpanjang.

- Pencarian Prefix Terpanjang

Fungsi pencarian prefix terpanjang digunakan untuk menghitung panjang prefix terpanjang yang juga merupakan suffix dari pola hingga posisi tertentu. Ini dilakukan dengan membandingkan setiap karakter pada pola dengan karakter-karakter sebelumnya.

- Tabel Lompatan

Tabel lompatan menyimpan informasi tentang seberapa jauh algoritma dapat melompat jika terjadi ketidakcocokan antara pola dengan teks pada suatu posisi tertentu. Nilai dalam tabel lompatan adalah panjang prefix terpanjang yang juga merupakan suffix dari pola hingga posisi tersebut.

2. Pencocokan Pola dengan Teks.

Setelah tabel lompatan diinisialisasi, algoritma KMP mulai mencocokan pola dengan teks dari kiri ke kanan.

- a. Mulai pencocokan pola dari kiri teks.
- b. Bandingkan setiap karakter pada pola dengan karakter yang sesuai pada teks.
- c. Jika semua karakter cocok, pola ditemukan dalam teks.
- d. Jika terjadi ketidakcocokan:

I. Gunakan tabel lompatan untuk menentukan jarak lompatan berdasarkan prefix terpanjang yang juga merupakan suffix dari pola hingga posisi ketidakcocokan.

II. Lompat ke posisi baru di teks sesuai dengan jarak lompatan yang diperoleh dari tabel lompatan.

- e. Ulangi langkah b hingga d hingga pola cocok atau teks habis.
3. Implementasi dalam Sistem Identifikasi Sidik Jari.
- Dalam sistem identifikasi sidik jari, algoritma KMP dapat diimplementasikan dengan langkah-langkah berikut:

- a. Representasikan pola sidik jari sebagai *string* ASCII.
- b. Inisialisasi tabel lompatan untuk pola sidik jari.
- c. Gunakan algoritma KMP untuk mencari pola sidik jari dalam basis data sidik jari (teks).
- d. Jika pola ditemukan, kembalikan identitas individu yang sesuai dengan pola sidik jari tersebut.

Kelebihan Algoritma Knuth-Morris-Pratt

1. Efisien untuk Pola yang Pendek atau Banyak Karakter yang Berulang.

Algoritma KMP sangat efisien untuk pola yang pendek atau pola yang memiliki banyak karakter yang berulang. Dalam kasus ini, algoritma dapat memanfaatkan informasi tentang prefix dan suffix yang sama dalam pola untuk melakukan lompatan yang signifikan saat terjadi ketidakcocokan.

2. Tidak Membutuhkan Ruang Tambahan yang Besar.

Algoritma KMP hanya membutuhkan tabel lompatan dengan ukuran yang sama dengan panjang pola. Ini membuat algoritma KMP lebih efisien dalam penggunaan memori dibandingkan dengan algoritma pencarian string lainnya yang membutuhkan ruang tambahan yang lebih besar.

3. Kinerja yang Konsisten.

Algoritma KMP memiliki kompleksitas waktu yang konsisten, yaitu $O(n)$ dalam kasus terburuk, di mana n adalah panjang teks. Ini membuatnya menjadi pilihan yang stabil dan dapat diandalkan untuk pencarian string dalam berbagai situasi.

Kekurangan Algoritma Knuth-Morris-Pratt.

1. Kurang Efisien untuk Pola yang Panjang dan Tidak Memiliki Banyak Karakter yang Berulang.

Meskipun algoritma KMP efisien untuk pola yang pendek atau banyak karakter yang berulang, namun algoritma ini kurang efisien untuk pola yang panjang dan tidak memiliki banyak karakter yang berulang. Dalam kasus ini, lompatan yang dilakukan mungkin tidak terlalu besar, sehingga mengurangi keunggulan utama dari algoritma ini.

2. Membutuhkan Perhitungan Awal untuk Tabel Lompatan.

Algoritma KMP membutuhkan perhitungan awal untuk membuat tabel lompatan, yang dapat memakan waktu jika pola sangat panjang. Meskipun perhitungan ini hanya dilakukan sekali, namun dapat menjadi masalah jika pola sering berubah atau diperbarui.

3. Tidak Optimal untuk Kasus Pencarian Berulang pada Teks yang Sama.

Jika kita perlu melakukan pencarian berulang pada teks yang sama dengan pola yang berbeda, algoritma KMP mungkin tidak optimal. Dalam situasi seperti ini, algoritma lain seperti Boyer-Moore (BM) yang memiliki tabel preprocessing yang terpisah untuk setiap pola mungkin lebih efisien.

Analisis Kompleksitas.

1. Kompleksitas Waktu.

- a. Kasus Terburuk

Kompleksitas waktu algoritma KMP dalam kasus terburuk adalah $O(m + n)$, di mana m adalah panjang pola dan n adalah panjang teks. Ini terjadi ketika pola tidak ditemukan dalam teks atau ketika algoritma harus membandingkan setiap

karakter dalam teks. Dalam kasus ini, algoritma KMP tidak dapat melakukan lompatan yang signifikan, sehingga kompleksitas waktunya menjadi linear terhadap panjang teks.

b. Kasus Terbaik.

Kompleksitas waktu algoritma KMP dalam kasus terbaik adalah $O(n/m)$, di mana n adalah panjang teks dan m adalah panjang pola. Ini terjadi ketika pola ditemukan pada awal teks atau ketika algoritma dapat melakukan lompatan yang besar setiap kali terjadi ketidakcocokan. Dalam kasus ini, algoritma KMP dapat memanfaatkan lompatan yang besar untuk meminimalkan jumlah perbandingan yang dilakukan.

2. Kompleksitas Ruang.

Kompleksitas ruang algoritma KMP adalah $O(m)$, di mana m adalah panjang pola. Ini dibutuhkan untuk menyimpan tabel lompatan. Dibandingkan dengan algoritma lain seperti Boyer-Moore yang membutuhkan ruang tambahan untuk tabel preprocessing lainnya, kebutuhan ruang algoritma KMP relatif lebih kecil.

Optimasi Algoritma Knuth-Morris-Pratt.

Meskipun algoritma Knuth-Morris-Pratt (KMP) sudah cukup efisien, ada beberapa teknik optimasi yang dapat diterapkan untuk meningkatkan kinerjanya lebih lanjut:

1. Optimasi Perhitungan Tabel Lompatan.

Proses perhitungan tabel lompatan dapat dioptimalkan dengan menggunakan teknik seperti pemrograman dinamis atau pendekatan iteratif yang lebih efisien. Dengan mengoptimalkan perhitungan ini, kita dapat mengurangi waktu yang dibutuhkan untuk inisialisasi tabel lompatan, terutama untuk pola yang sangat panjang.

Implementasi dalam Sistem Identifikasi Sidik Jari.

Dalam konteks sistem identifikasi sidik jari, algoritma Knuth-Morris-Pratt dapat diimplementasikan dengan langkah-langkah berikut:

1. Representasi Pola Sidik Jari sebagai String.

Gambar sidik jari diekstraksi menjadi *binary* dan direpresentasikan dalam bentuk ASCII.

2. Inisialisasi Tabel Lompatan.

Sebelum menjalankan algoritma KMP, kita perlu menginisialisasi tabel lompatan untuk pola sidik jari yang direpresentasikan sebagai string. Ini dilakukan dengan menghitung prefix terpanjang yang juga merupakan suffix dari pola untuk setiap posisi dalam pola.

3. Pencarian Pola dalam Basis Data.

Dengan tabel lompatan yang telah diinisialisasi, kita dapat menerapkan algoritma KMP untuk mencari pola sidik jari dalam basis data sidik jari (teks). Basis data ini juga harus direpresentasikan dalam bentuk string dengan cara yang sama seperti pola sidik jari.

4. Pencocokan dan Identifikasi Individu.

Jika pola sidik jari ditemukan dalam basis data, algoritma KMP akan memberikan lokasi pola tersebut dalam teks. Kemudian, kita dapat menggunakan informasi ini untuk mengidentifikasi individu yang sesuai dengan pola sidik jari tersebut dalam basis data.

5. Penanganan Kasus Khusus.

Dalam sistem identifikasi sidik jari, kita mungkin tidak mendapatkan *exact match* dengan algoritma KMP. Dalam situasi seperti ini, kita dapat menggunakan teknik tambahan seperti pencocokan pola yang lebih fleksibel atau penggunaan algoritma lain seperti *Longest Common Subsequence* (LCS) untuk mengukur kesamaan antara pola sidik jari.

Dengan mengimplementasikan algoritma Knuth-Morris-Pratt dalam sistem identifikasi sidik jari, kita dapat memanfaatkan keunggulan utama algoritma ini, yaitu efisiensi pencarian untuk pola yang pendek atau banyak karakter yang berulang, serta kebutuhan ruang yang relatif

kecil. Hal ini dapat membantu meningkatkan kecepatan dan akurasi proses identifikasi individu berdasarkan sidik jari mereka.

Namun, perlu diingat bahwa algoritma KMP mungkin tidak optimal untuk pola yang panjang dan tidak memiliki banyak karakter yang berulang, atau dalam kasus pencarian berulang pada teks yang sama dengan pola yang berbeda. Dalam situasi seperti itu, kita mungkin perlu mempertimbangkan penggunaan algoritma lain seperti Boyer-Moore atau mengombinasikan beberapa algoritma untuk mendapatkan kinerja yang optimal.

3.4 Algoritma *Longest Common Subsequence* (LCS).

Algoritma *Longest Common Subsequence* digunakan jika dan hanya jika tidak ditemukan hasil yang *exact match* pada algoritma KMP dan BM.

- Jika $X = \text{"ABCBDAB"}$ dan $Y = \text{"BDCABA"}$, maka $\text{LCS}(X, Y) = \text{"BCBA"}$ atau "BDAB" , dengan tingkat kemiripan = $\text{panjang_hasil_LCS} / \text{panjang_Y} = 4 / 6 = 0.667 = 67\%$.
- Jika $X = \text{"ACADEB"}$ dan $Y = \text{"CBAGC"}$, maka $\text{LCS}(X, Y) = \text{"ACB"}$, dengan tingkat kemiripan = $\text{panjang_hasil_LCS} / \text{panjang_Y} = 3 / 5 = 0.6 = 60\%$.

Algoritma LCS digunakan dengan pendekatan pemrograman dinamis:

1. Inisiasi tabel awal (misal kita membandingkan 4 huruf : "abcd" dengan "acbe"):

| | | a | b | c | d |
|---|---|---|---|---|---|
| | 0 | 0 | 0 | 0 | 0 |
| a | 0 | | | | |
| c | 0 | | | | |
| b | 0 | | | | |

| | | | | | |
|---|---|--|--|--|--|
| e | 0 | | | | |
|---|---|--|--|--|--|

2. Setelah itu, ketika ditemukan huruf yang sama antara kolom dan baris, maka masukan nilai dari sel kiri atas yang ditambah 1 (contoh pada saat membandingkan kolom “a” dengan baris “a”, karena kolom dan baris sama-sama “a”, maka masukan nilai sel kiri atas (yaitu nol) kemudian ditambahkan 1), kemudian kolom berikutnya akan mengikuti nilai tersebut:

| | | a | b | c | d |
|---|---|---|---|---|---|
| | 0 | 0 | 0 | 0 | 0 |
| a | 0 | 1 | 1 | 1 | 1 |
| c | 0 | | | | |
| b | 0 | | | | |
| e | 0 | | | | |

3. Lakukan langkah 2 pada setiap sel sampai seluruh sel pada tabel terisi:

| | | a | b | c | d |
|---|---|---|---|---|---|
| | 0 | 0 | 0 | 0 | 0 |
| a | 0 | 1 | 1 | 1 | 1 |
| c | 0 | 1 | 1 | 2 | 2 |
| b | 0 | 1 | 2 | 2 | 2 |
| e | 0 | 1 | 2 | 2 | 2 |

4. Didapatkan panjang dari LCS pada sel pojok kanan bawah:

| | | a | b | c | d |
|---|---|---|---|---|---|
| | 0 | 0 | 0 | 0 | 0 |
| a | 0 | 1 | 1 | 1 | 1 |
| c | 0 | 1 | 1 | 2 | 2 |
| b | 0 | 1 | 2 | 2 | 2 |
| e | 0 | 1 | 2 | 2 | 2 |

5. Hitung persentase kemiripan = panjang_LCS / panjang_string = 2 / 4 = 0.5 = 50%.

Kompleksitas dari algoritma LCS ini adalah $O(m * n)$ dengan m adalah panjang string satu dan n adalah panjang string lainnya.

Bab IV

Implementasi dan Pengujian

4.1 Spesifikasi Teknis Program

4.1.1 Kelas Utama

a. BMPToBytes

Kelas bertanggung jawab untuk melakukan pra-pemrosesan gambar sidik jari

b. IBaseAlgorithm

Interface untuk kelas algoritma, mendefinisikan metode yang diperlukan untuk mengimplementasikan berbagai algoritma.

c. KMPAlgorithm

Kelas ini mengimplementasikan algoritma Knuth-Morris-Pratt (KMP) untuk pencocokan pola dalam data sidik jari.

1. *private string pattern* : atribut yang menyimpan pola masukan sidik jari.
2. *private int[] lps* : atribut yang menyimpan tabel *Longest Prefix Suffix* yang akan digunakan dalam algoritma *Knuth – Morris – Pratt*.

d. BoyerMoore

Kelas ini mengimplementasi algoritma *Boyer Moore* untuk pencocokan pola dalam data sidik jari.

1. *private readonly int alphabetSize* : atribut menyimpan besar alfabet, *default* 256.
2. *private readonly int[] badCharacterShift* : atribut menyimpan tabel heuristik *bad character shift* yang akan digunakan dalam algoritma *Boyer Moore*.
3. *private readonly int[] goodSuffixShift* : atribut menyimpan tabel heuristik *good suffix shift* yang akan digunakan dalam algoritma *Boyer Moore*.
4. *private readonly int[] borderPositions* : atribut yang menyimpan *border position* yang akan digunakan saat membuat tabel *good suffix shift*.
5. *private readonly char[] pattern* : atribut yang menyimpan pola masukan sidik jari.

e. LongestCommonSubsequence

Kelas ini mengimplementasi algoritma *Longest Common Subsequence* untuk pencocokan pola dalam data sidik jari.

1. *private string pattern* : atribut yang menyimpan pola masukan sidik jari.

f. AlgoMaster

Kelas ini menentukan algoritma yang digunakan berdasarkan input pengguna. Objek dari kelas ini akan digunakan oleh GUI ketika ingin mendapatkan hasil pencarian sidik jari.

1. *private string pattern* : atribut yang menyimpan pola masukan sidik jari.

g. SQLiteDataAccess

Kelas ini bertanggung jawab dalam mengatur pengambilan data dalam *database*;

1. *private const string ConnectionString* : atribut yang menyimpan lokasi *database*.

h. RegexAlayPattern

Kelas ini bertanggung jawab dalam membuat pattern regex dari nama yang normal menjadi nama yang alay.

1. *private Dictionary<char, string> leetMap* : atribut ini merupakan sebuah map yang memetakkan sebuah huruf menjadi huruf yang “mirip” dengannya dalam bahasa alay, contohnya a menjadi 4, e menjadi 3 dan seterusnya
2. *private Dictionary<char, char> reverseLeetMap* : atribut ini melakukan hal sebaliknya dengan leetMap dimana ia menerima angka yang mirip untuk mendapatkan huruf awal. Hal ini untuk melakukan normalisasi di kasus tidak ada nama yang sesuai dengan pattern yang dihasilkan
3. *private string regexPattern* : atribut ini merupakan pattern yang dihasilkan setelah pemrosesan kata

i. Encrypt

Kelas ini bertanggung jawab untuk melakukan enkripsi terhadap nilai asli. Teknik enkripsi yang dilakukan adalah RSA dengan nilai n = 3337 dan *public key* e = 79.

1. *private static BigInteger n* : nilai dari hasil perkalian dua buah bilangan prima yang disembunyikan (p = 47; q = 71).

2. *private static BigInteger e* : nilai yang relatif prima dengan m, dengan $m = (p-1)(q-1)$, dengan p dan q adalah bilangan prima dan $pq = n$.
- j. EncryptDatabase
- Kelas ini bertanggung jawab untuk melakukan enkripsi pada database terutama pada tabel *biodata*. Kolom yang dienkripsi adalah NIK, tempat_lahir, tanggal_lahir, jenis_kelamin, golongan_darah, alamat, agama, status_perkawinan, pekerjaan, dan kewarganegaraan.
- k. Decrypt
- Kelas ini bertanggung jawab untuk melakukan dekripsi dari data yang ingin ditampilkan dengan nilai *private key d* = 1019 dan nilai *n* = 3337.
1. *private static BigInteger d* : nilai *private key* untuk melakukan dekripsi yang didapatkan dari *public key* dengan menghitung kekongruenan.
 2. *private static BigInteger n* : nilai dari hasil perkalian dua buah bilangan prima yang disembunyikan ($p = 47$; $q = 71$).

4.1.2 Metode Kelas

- a. BMPToBytes

| Nama Metode | Keterangan |
|---|--|
| <i>ConvertToBlackAndWhite(string filePath)</i> → <i>image</i> | Membaca <i>file</i> pada <i>filePath</i> dan memproses <i>file</i> tersebut menjadi objek <i>image</i> (<i>SixLabors.ImageSharp.Image<Rgba32></i>) dalam bentuk hitam putih (0 hitam dan 1 putih). |
| <i>ConvertBMPToASCII(string filePath)</i> → <i>List<string></i> | Membaca <i>file</i> pada <i>filePath</i> , yang memiliki format BMP, dan |

| | |
|--|--|
| | mengembalikan daftar <i>string</i> (<i>List<string></i>) di mana setiap <i>string</i> mewakili bentuk ASCII dari setiap baris dalam <i>file</i> tersebut. |
| <i>ConvertImageToBinary(SixLabors.ImageSharp.Image<Rgba32> image) → List<string></i> | Membaca <i>file</i> pada <i>filePath</i> , yang memiliki format BMP, dan mengembalikan daftar <i>string</i> (<i>List<string></i>) di mana setiap <i>string</i> mewakili bentuk biner dari setiap baris dalam <i>file</i> tersebut. |
| <i>GetImagePattern(string filename) → string</i> | Membaca <i>file</i> pada <i>filename</i> dan mengembalikan pola keunikan gambar tersebut dalam bentuk ASCII. |
| <i>RemoveTrailingZeroRows(List<string> binaryRows) → List<string></i> | Menghapus baris biner yang 0, baris tersebut berada di akhir <i>binaryRows</i> . |

b. IBaseAlgorithm

| Nama Metode | Keterangan |
|---|--|
| <i>Search(string text) → int</i> | Metode pencarian yang harus diimplementasikan oleh kelas algoritma lainnya, mengembalikan -1 jika tidak ada <i>substring</i> cocok dengan pola sidik jari pada <i>text</i> . |
| <i>SearchAllRows(List<string> text) → int</i> | Metode pencarian pada <i>List<String></i> yang harus diimplementasikan oleh kelas algoritma lainnya, mengembalikan -1 jika |

| | |
|--|--|
| | tidak ada <i>string</i> pada <i>text</i> yang memiliki <i>substring</i> yang cocok dengan pola sidik jari. |
|--|--|

c. KMPAlgorithm

| Nama Metode | Keterangan |
|---|--|
| <i>ComputeLPSArray()</i> | <i>Preprocess</i> tabel lps yang akan digunakan untuk algoritma <i>Knuth – Morris – Pratt</i> . |
| <i>Search(string text) → int</i> | Metode pencarian pola sidik jari pada <i>string text</i> dengan algoritma <i>Knuth - Morris - Pratt</i> , mengembalikan -1 jika tidak ada substring cocok. |
| <i>SearchAllRows(List<string> text) → int</i> | Metode pencarian pola sidik jari pada <i>List<string> text</i> dengan algoritma <i>Knuth - Morris - Pratt</i> , mengembalikan -1 jika tidak <i>string</i> pada <i>text</i> yang memiliki <i>substring</i> yang cocok dengan pola sidik jari. |

d. BoyerMoore

| Nama Metode | Keterangan |
|--|---|
| <i>PreprocessBadCharacterHeuristic()</i> | <i>Preprocess</i> tabel heuristik <i>bad character shift</i> yang akan digunakan untuk algoritma <i>Boyer Moore</i> . |

| | |
|---|---|
| <i>PreprocessGoodSuffixHeuristic()</i> | <i>Preprocess</i> tabel heuristik <i>good character shift</i> yang akan digunakan untuk algoritma <i>Boyer Moore</i> . |
| <i>PreprocessBorderPositions()</i> | Memastikan tabel <i>good character shift</i> terisi dengan benar, untuk slot yang belum terisi (bernilai 0) diisikan dengan <i>shift value</i> . |
| <i>Search(string text) → int</i> | Metode pencarian pola sidik jari pada <i>string text</i> dengan algoritma <i>Boyer Moore</i> , mengembalikan -1 jika tidak ada substring cocok. |
| <i>SearchAllRows(List<string> text) → int</i> | Metode pencarian pola sidik jari pada <i>List<string> text</i> dengan algoritma <i>Boyer Moore</i> , mengembalikan -1 jika tidak <i>string</i> pada <i>text</i> yang memiliki <i>substring</i> yang cocok dengan pola sidik jari. |

e. LongestCommonSubsequence

| Nama Metode | Keterangan |
|---|---|
| <i>Search(string text) → int</i> | Metode pencarian pola sidik jari pada <i>string text</i> dengan algoritma <i>Longest Common Subsequence</i> , mengembalikan panjang <i>longest common subsequence</i> |
| <i>SearchAllRows(List<string> text) → int</i> | Metode pencarian pola sidik jari pada <i>List<string> text</i> dengan algoritma <i>Longest Common Subsequence</i> , |

| | |
|--|--|
| | mengembalikan panjang <i>common sequence</i> terbesar di antara <i>string</i> pada <i>text</i> . |
| <i>SearchBestMatch(SQLiteDataAccess sqlData, List<SidikJari> allSidik) → Tuple<Biodata?, SidikJari?></i> | Metode untuk mencari sidik jari dengan kecocokan terbesar dengan masukan sidik jari pengguna, lalu mengembalikan sidik jari dan <i>biodata</i> tersebut. |
| <i>FindLCSLength(string text) → int</i> | Mencari panjang <i>longest common sequence</i> pada <i>text</i> . |

f. AlgoMaster

| Nama Metode | Keterangan |
|---|--|
| <i>Search(string filename, int algorithmType)</i> | Metode yang mengembalikan tuple yang berisi biodata dan gambar sidik jari (<i>Tuple<Biodata?, SidikJari?></i>) yang didapatkan dari proses KMP, BM, ataupun LCS. |

g. SQLiteDataAccess

| Nama Metode | Keterangan |
|-----------------------|--|
| <i>GetSidikJari()</i> | Metode ini akan mengembalikan <i>List<SidikJari></i> dimana <i>SidikJari</i> adalah objek yang berisi data sidik jari dari <i>database</i> . |

| | |
|---|--|
| <code>public List<Biodata> GetBiodata()</code> | Metode ini akan mengembalikan <code>List<Biodata></code> yang berisi kumpulan dari seluruh biodata yang ada. |
| <code>public Biodata? searchForBiodata(SidikJari source)</code> | Metode ini akan mengembalikan biodata yang sesuai dengan masukan sidik jari. |

h. RegexAlayPattern

| Nama Metode | Keterangan |
|---|--|
| <code>GeneratePattern(string baseString)</code> | Metode ini menerima <code>baseString</code> yang merupakan nama normal yang didapatkan dari table sidik jari. Kemudian akan membuat <i>pattern</i> alay regex yang disimpan pada atribut kelas |
| <code>IsMatch(string input) → bool</code> | Metode ini akan menerima input nama dalam bahasa alay dan memeriksa pencocokan dengan <i>pattern</i> regex yang telah dibuat |
| <code>public string GetPattern() → string</code> | Metode ini mengembalikan <i>pattern</i> regex yang telah dibuat |
| <code>public string Normalize(string input) → string</code> | Metode ini akan mencoba menormalisasi bahasa alay menjadi normal dengan menggunakan <i>reverseLeetMap</i> |

i. Encrypt

| Nama Metode | Keterangan |
|--------------------|-------------------|
| | |

| | |
|--|--|
| <pre>public static String encrypt(String originalString)</pre> | <p>Metode ini menerima masukan nilai awal yang ingin dienkripsi kemudian setiap karakter diubah menjadi nilai desimal dan dilakukan enkripsi dengan teknik RSA kemudian mengembalikan seluruh deretan angka desimal hasil enkripsi yang dipisahkan dengan spasi.</p> |
|--|--|

j. EncryptDatabase

| Nama Metode | Keterangan |
|---|---|
| <pre>public static void encryptDatabase()</pre> | Metode ini melakukan enkripsi pada database yang digunakan, terutama pada tabel <i>biodata</i> . |
| <pre>private static String encrypt(String oldValue)</pre> | Metode ini melakukan enkripsi masukan dan mengeluarkan hasil enkripsi dengan memanggil metode dari kelas <i>Encrypt</i> . |

k. Decrypt

| Nama Metode | Keterangan |
|---|---|
| <pre>public static String decrypt(String encryptedString)</pre> | Metode ini menerima masukan hasil enkripsi dan mengembalikan nilai awal setelah dilakukan dekripsi. |

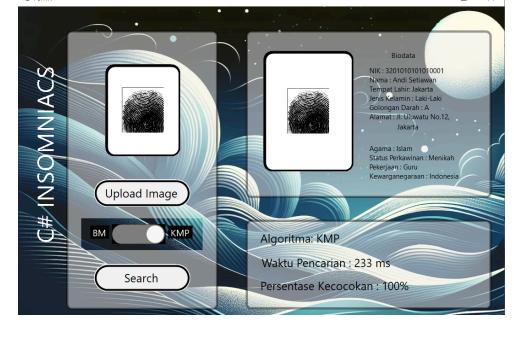
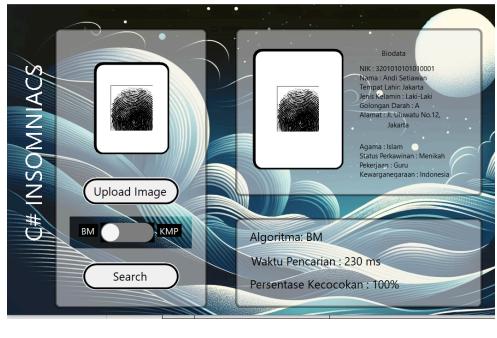
4.2 Tata Cara Penggunaan

Berikut adalah tata cara penggunaan program kami:

1. Penggunaan program ini dimulai dari pengguna memasukkan gambar sidik jari yang ingin dicari dengan menekan tombol “Upload Image”.
2. Selanjutnya atur *toggle* untuk memilih algoritma yang ingin digunakan (KMP untuk Knuth-Morris-Pratt dan BM untuk Boyer-Moore).

3. Tekan tombol “Search” untuk memulai proses pencarian.
4. Tunggu beberapa saat sampai hasil keluar, jika terdapat sidik jari pada basis data dengan tingkat kemiripan diatas 55%, maka hasil akan ditampilkan, jika tingkap kemiripan berada dibawah 55% maka akan muncul *pop-up* yang memberitahu tidak terdapat sidik jari tersebut pada basis data.

4.3 Pengujian

| No | Knuth-Morris-Pratt | Boyer-Moore |
|----|---|--|
| 1 |  |  |
| 2 |  |  |

| | | |
|---|--|--|
| 3 | | |
| 4 | | |
| 5 | | |

| | | |
|---|--|--|
| 6 | | |
| 7 | | |
| 8 | | |

| | | |
|----|--|--|
| 9 | | |
| 10 | | |

4.4 Analisis Hasil Pengujian

Berdasarkan hasil pengujian, didapatkan bahwa 9 dari 10 percobaan (percobaan ke - 2,3,4,5,6,7,8,9,10) algoritma Boyer-Moore lebih cepat dibandingkan dengan algoritma Knuth-Morris-Pratt. Hal ini dapat dikarenakan algoritma KMP cenderung lebih lambat untuk pola yang tidak memiliki karakter berulang sebab KMP bergantung kepada *suffix* dan *prefix* dari pola, sedangkan algoritma BM semakin efektif jika variasi karakter pada pola semakin banyak (ukuran alfabet besar).

Bab V

Kesimpulan dan Saran

5.1 Kesimpulan

Setelah melakukan berbagai pengujian, kami menemukan bahwa algoritma Knuth-Morris-Pratt sangat efektif dalam melakukan pencocokan pola dengan efisiensi yang tinggi. Algoritma ini memiliki keunggulan dalam menemukan pola dengan cepat tanpa perlu kembali ke awal teks saat terjadi ketidakcocokan, sehingga menghemat waktu dalam proses pencarian. Di sisi lain, algoritma Boyer-Moore dikenal karena kecepatannya dan efisiensinya dalam pencocokan pola. Algoritma ini bekerja sangat cepat, terutama dalam basis data besar, karena mampu melompati sejumlah karakter atau posisi dengan cepat menggunakan pendekatan heuristiknya. Berdasarkan hasil pengujian, didapatkan bahwa algoritma Boyer-Moore lebih cepat daripada algoritma Knuth-Morris-Pratt pada persoalan pencocokan sidik jari dengan representasi karakter ASCII.

Dengan pemahaman mendalam tentang teori dan penerapan kedua algoritma ini, sistem identifikasi sidik jari yang kami kembangkan dapat memberikan keamanan dan keandalan yang tinggi. Desain sistem yang baik, mulai dari pencocokan pola, hingga penerapan langkah-langkah keamanan, memastikan kinerja yang optimal dan melindungi data biometrik yang sensitif.

5.2 Saran

Saran untuk kelompok

1. Mengoptimalkan Struktur Pembagian Tugas dan Kerjasama Tim:
 - Tingkatkan koordinasi dan komunikasi dalam tim untuk memastikan bahwa setiap anggota memahami peran dan tanggung jawab masing-masing.
 - Tetapkan jadwal pertemuan rutin untuk membahas perkembangan proyek dan mengatasi hambatan yang mungkin muncul.
 - Gunakan alat kolaborasi seperti Trello atau Slack untuk memantau progres dan mengelola tugas secara efektif.
2. Perluasan Pengembangan Algoritma:

- Lakukan penelitian lebih lanjut untuk mengevaluasi dan mengimplementasikan algoritma tambahan yang dapat meningkatkan akurasi dan kecepatan sistem identifikasi sidik jari, seperti Support Vector Machine (SVM) atau Deep Learning.

3. Peningkatan Penulisan Komentar dan Dokumentasi:

- Pastikan setiap bagian kode disertai dengan komentar yang jelas dan informatif untuk memudahkan pemahaman dan pemeliharaan di masa mendatang.
- Buat dokumentasi proyek yang lengkap, termasuk penjelasan tentang arsitektur sistem, flowchart proses, dan panduan penggunaan.

4. Peningkatan Antarmuka Pengguna (UI/UX):

- Fokus pada pengembangan antarmuka pengguna yang lebih intuitif dan user-friendly untuk memudahkan interaksi dengan sistem.
- Kumpulkan umpan balik dari pengguna untuk memahami kekurangan dan area yang perlu diperbaiki dalam antarmuka pengguna.

5. Keamanan dan Privasi Data:

- Tingkatkan langkah-langkah keamanan dalam penyimpanan dan transmisi data sidik jari untuk mencegah akses yang tidak sah.
- Pastikan bahwa semua data biometrik dienkripsi dan dilindungi sesuai dengan standar keamanan yang berlaku.

Lampiran

Github Repository

https://github.com/Zechtro/Tubes3_C-Insomniacs

Video

https://youtu.be/G4Y_T3k8KEA?si=NqjjQ1udw1EgB6Eq

Daftar Pustaka

<https://iopscience.iop.org/article/10.1088/1757-899X/662/2/022040>

<https://www.kaggle.com/datasets/ruizagara/socofing>

<https://learn.microsoft.com/en-us/shows/on-net/how-do-i-use-sql-server-with-csharp-and-dotnet>

<https://learn.microsoft.com/en-us/visualstudio/ide/create-csharp-winform-visual-studio?view=vs-2022>

<https://medium.com/@neethamadhu.ma/good-suffix-rule-in-boyer-moore-algorithm-explained-simplly-9d9b6d20a773>

<https://www.youtube.com/watch?v=NnD96abizww>

<https://docs.sixlabors.com/articles/imagesharp/gettingstarted.html>

<https://stackoverflow.com/questions/62017557/create-new-instance-of-image-in-c-sharp-using-sixlabors-imagesharp>

<https://informatika.stei.itb.ac.id/~rinaldi.munir/TA/Makalah-KNIF-2010.pdf>