

# **LAPORAN TUGAS BESAR 3**

## **IF2211 STRATEGI ALGORITMA**



Dipersiapkan oleh:

**Kelompok 43**

Jovandra Otniel P.S. 13523141

Natalia Desiany Nursimin 13523157

Anas Ghazi Al Ghifari 13523159

**Program Studi Teknik Informatika**

**Sekolah Teknik Elektro dan Informatika**

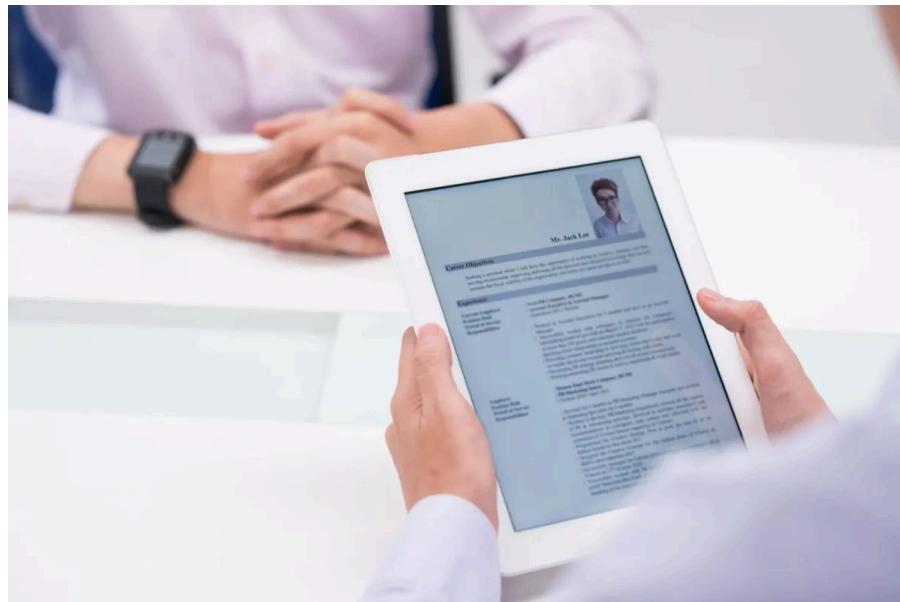
**Institut Teknologi Bandung**

**2025**

## DAFTAR ISI

<b>DAFTAR ISI.....</b>	<b>2</b>
<b>BAB 1 DESKRIPSI TUGAS.....</b>	<b>4</b>
Penjelasan Implementasi.....	5
Penggunaan Program.....	7
<b>BAB 2 LANDASAN TEORI.....</b>	<b>9</b>
2.1. Dasar Teori.....	9
2.2. Penjelasan singkat mengenai aplikasi web yang dibangun.....	9
<b>BAB 3 ANALISIS PEMECAHAN MASALAH.....</b>	<b>10</b>
3.1. Langkah-Langkah Pemecahan Masalah.....	10
3.2. Proses Pemetaan Masalah Menjadi Elemen-Elemen Algoritma KMP dan BM.....	10
3.3 Fitur Fungsional dan Arsitektur Aplikasi Web yang Dibangun.....	10
<b>BAB 4 IMPLEMENTASI DAN PENGUJIAN.....</b>	<b>11</b>
4.1. Spesifikasi Teknis Program.....	11
4.1.1 Struktur Direktori.....	11
4.1.2 Source Code.....	11
4.1.3. Fungsi dan Prosedur yang Dibangun.....	11
4.2. Penjelasan Tata Cara Penggunaan Program.....	11
4.3. Hasil Pengujian Minimal 4 Pencarian dengan Variasi Keyword, Algoritma, dan Panjang Keyword yang Berbeda.....	11
4.4. Analisis Hasil Pengujian.....	11
<b>BAB 5 KESIMPULAN, SARAN, DAN REFLEKSI TENTANG TUGAS BESAR 3.....</b>	<b>12</b>
5.1. Kesimpulan.....	12
5.2. Saran.....	12
5.3. Refleksi.....	12
<b>LAMPIRAN.....</b>	<b>13</b>
Tautan Repository GitHub:.....	13
Tautan Video YouTube:.....	13
Tabel Pengecekan Fitur:.....	13

## **BAB 1 DESKRIPSI TUGAS**



**Gambar 1. CV ATS dalam Dunia Kerja**

(Sumber: <https://www.antaranews.com/> )

Di era digital ini, keamanan data dan akses menjadi semakin penting. Perkembangan proses rekrutmen tenaga kerja telah mengalami perubahan signifikan dengan memanfaatkan teknologi untuk meningkatkan efisiensi dan akurasi. Salah satu inovasi yang menjadi solusi utama adalah Applicant Tracking System (ATS), yang dirancang untuk mempermudah perusahaan dalam menyaring dan mencocokkan informasi kandidat dari berkas lamaran, khususnya Curriculum Vitae (CV). ATS memungkinkan perusahaan untuk mengelola ribuan dokumen lamaran secara otomatis dan memastikan kandidat yang relevan dapat ditemukan dengan cepat.

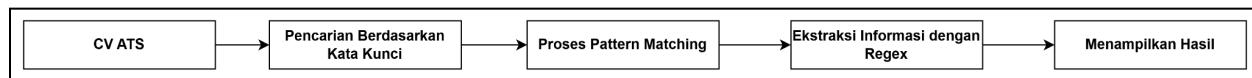
Meskipun demikian, salah satu tantangan besar dalam pengembangan sistem ATS adalah kemampuan untuk memproses dokumen CV dalam format PDF yang tidak selalu terstruktur. Dokumen seperti ini memerlukan metode canggih untuk mengekstrak informasi penting seperti identitas, pengalaman kerja, keahlian, dan riwayat pendidikan secara efisien. Pattern matching menjadi solusi ideal dalam menghadapi tantangan ini.

Pattern matching adalah teknik untuk menemukan dan mencocokkan pola tertentu dalam teks. Dalam konteks ini, algoritma Boyer-Moore dan Knuth-Morris-Pratt (KMP) sering digunakan karena keduanya menawarkan efisiensi tinggi untuk pencarian teks di dokumen besar. Algoritma ini memungkinkan sistem ATS untuk mengidentifikasi informasi penting dari CV pelamar dengan kecepatan dan akurasi yang optimal.

Di dalam Tugas Besar 3 ini, Anda diminta untuk mengimplementasikan sistem yang dapat melakukan deteksi informasi pelamar berbasis dokumen CV digital. Metode yang akan digunakan untuk melakukan deteksi pola dalam CV adalah Boyer-Moore dan Knuth-Morris-Pratt. Selain itu, sistem ini akan dihubungkan dengan identitas kandidat melalui basis data sehingga harapannya terbentuk sebuah sistem yang dapat mengenali profil pelamar secara lengkap hanya dengan menggunakan CV digital.

## Penjelasan Implementasi

Dalam tugas ini, Anda akan mengembangkan sebuah sistem ATS (Applicant Tracking System) berbasis CV Digital dengan memanfaatkan teknik Pattern Matching. Implementasi sistem ini akan menggunakan algoritma Boyer-Moore dan Knuth-Morris-Pratt (*Aho-Corasick* apabila mengerjakan bonus) untuk menganalisis dan mencocokkan pola dalam dokumen CV digital, sesuai dengan konsep yang telah dipelajari dalam materi dan slide perkuliahan.



**Gambar 2.** Skema Implementasi *Applicant Tracking System*

Sistem ini bertujuan untuk mencocokkan kata kunci dari user terhadap isi CV pelamar kerja dengan pendekatan pattern matching menggunakan algoritma KMP (Knuth-Morris-Pratt) atau BM (Boyer-Moore). Semua proses dilakukan secara in-memory, tanpa menyimpan hasil

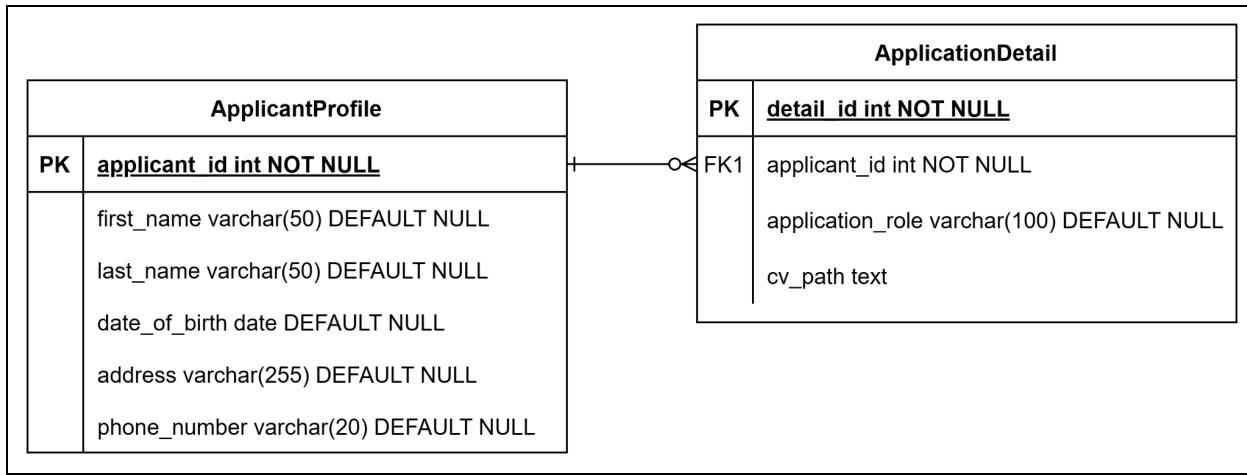
pencarian—hanya data mentah (raw) CV yang disimpan. Pengguna (HR atau rekruter) akan memberikan input berupa daftar kata kunci yang ingin dicari (misalnya: "python", "react", dan "sql") serta jumlah CV yang ingin ditampilkan (misalnya Top 10 matches). Setiap file CV dalam format PDF akan dikonversi menjadi satu string panjang yang memuat seluruh teks dari dokumen tersebut. Proses konversi ini bertujuan untuk mempermudah pencocokan pola menggunakan algoritma string matching, sehingga setiap keyword dapat dicari secara efisien dalam satu representasi data linear.

Untuk memberikan pemahaman yang lebih konkret, berikut disajikan contoh kasus penerapan sistem CV ATS beserta prosesnya dan contoh output yang dihasilkan. Dataset yang digunakan dalam contoh ini merupakan dataset CV ATS yang tercantum pada bagian referensi.

**Tabel 1.** Hasil ekstraksi teks dari CV ATS

CV ATS	Ekstraksi Text untuk Regex	Ekstraksi Text untuk <i>Pattern Matching</i> (KMP & BM)
 10276858.pdf	<a href="#">Ekstraksi Text Regex.txt</a>	<a href="#">Ekstraksi Text Pattern Matching.txt</a>

Pada tahap implementasi ini, setiap CV yang telah dikonversi menjadi string panjang untuk mempermudah proses pencocokan. Representasi ini menjadi dasar dalam mencari CV yang paling relevan dengan kata kunci yang dimasukkan oleh pengguna. Proses pencarian dilakukan dengan menggunakan algoritma pencocokan string Knuth-Morris-Pratt (KMP) dan Boyer-Moore (BM) untuk menemukan CV yang memiliki kemiripan tertinggi dengan kebutuhan yang ditentukan. Apabila tidak ditemukan satupun CV dalam basis data yang memiliki kecocokan kata kunci secara exact match menggunakan algoritma KMP maupun Boyer-Moore, maka sistem akan mencari CV yang paling mirip berdasarkan tingkat kemiripan di atas ambang batas tertentu (threshold). Hal ini mempertimbangkan kemungkinan adanya kesalahan pengetikan (typo) oleh pengguna atau HR saat memasukkan kata kunci. Anda diberikan **keleluasaan untuk menentukan nilai ambang batas persentase** kemiripan tersebut, dengan syarat dilakukan pengujian terlebih dahulu untuk menemukan nilai tuning yang optimal dan **dijelaskan secara rinci dalam laporan**. Metode perhitungan tingkat kemiripan harus diterapkan menggunakan algoritma **Levenshtein Distance**.



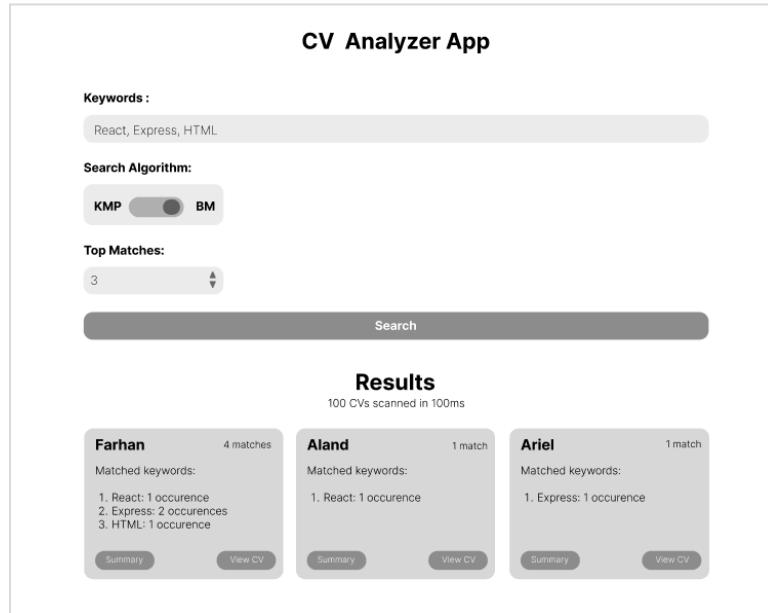
**Gambar 3.** Skema basis data CV ATS

Dalam skema basis data ini, tabel **ApplicantProfile** menyimpan informasi pribadi pelamar, sedangkan tabel **ApplicationDetail** menyimpan detail aplikasi yang diajukan oleh pelamar tersebut. Relasi antara tabel **ApplicantProfile** dan **ApplicationDetail** adalah one-to-many, karena seorang pelamar dapat mengajukan lamaran untuk beberapa posisi dalam perusahaan yang sama, atau bahkan perusahaan yang berbeda. Setiap lamaran mungkin memerlukan dokumen yang berbeda, seperti CV yang telah disesuaikan untuk peran tertentu.

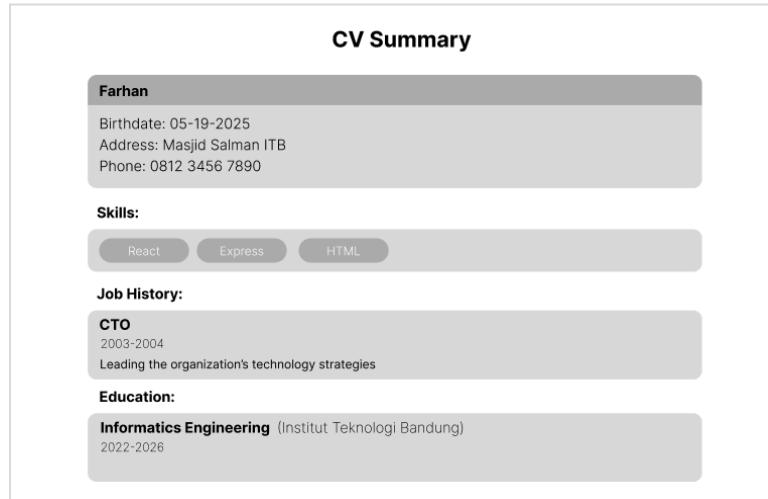
Untuk keperluan pengembangan awal, basis data silahkan **di-seeding secara mandiri** menggunakan data simulasi. Mendekati tenggat waktu pengumpulan tugas, asisten akan menyediakan seeding resmi yang akan digunakan untuk Demo Tugas Besar.

Atribut **cv\_path** pada tabel **ApplicationDetail** digunakan untuk menyimpan lokasi berkas CV digital pelamar di dalam repositori sistem. Lokasi penyimpanan mengikuti struktur folder di direktori **data/**, sebagaimana dijelaskan dalam struktur *repository* pada bagian [pengumpulan tugas](#). Berkas CV yang tersimpan akan dianalisis oleh sistem ATS (Applicant Tracking System) yang dikembangkan dalam Tugas Besar ini.

## Penggunaan Program



Gambar 4. Contoh Antarmuka Program (Halaman Home)



Gambar 5. Contoh Antarmuka Program (Halaman Summary)

Anda diperbolehkan menambahkan elemen tambahan seperti gambar, logo, atau komponen visual lainnya. Desain antarmuka untuk aplikasi desktop **tidak wajib mengikuti tata letak persis** seperti contoh yang diberikan, namun harus dibuat semenarik mungkin, serta tetap mencakup seluruh **komponen wajib yang telah ditentukan**:

- Judul Aplikasi
- Kolom input kata kunci memungkinkan pengguna memasukkan satu atau lebih *keyword*, yang dipisahkan dengan koma, seperti contoh: React, Express, HTML.

- Tombol toggle memungkinkan pengguna memilih salah satu dari dua algoritma pencarian, yaitu KMP atau BM, dengan hanya satu algoritma yang bisa dipilih pada satu waktu.
- *Top Matches Selector* digunakan untuk memilih jumlah CV teratas yang ingin ditampilkan berdasarkan hasil pencocokan.
- *Search Button* digunakan untuk memulai proses pencarian. Diletakkan secara mencolok di bawah *input field*.
- *Summary Result Section* berisi informasi waktu eksekusi pencarian untuk kedua tipe matching yang dilakukan (*exact match* dengan KMP/BM dan *fuzzy match* dengan Levenshtein Distance), misalnya: “Exact Match: 100 CVs scanned in 100ms.\n Fuzzy Match: 100 CVs scanned in 101ms.”
- *Container* hasil pencarian atau kartu CV digunakan untuk menampilkan data hasil pencocokan berdasarkan keyword yang sesuai. Setiap kartu memuat informasi seperti nama kandidat, jumlah kecocokan yang dihitung dari jumlah keyword yang ditemukan, serta daftar kata kunci yang cocok beserta frekuensi kemunculannya. Selain itu, tersedia dua tombol aksi: tombol *Summary* untuk menampilkan ekstraksi informasi dari CV, serta tombol *View CV* yang memungkinkan pengguna melihat langsung file CV asli.

Secara umum, berikut adalah cara umum penggunaan program:

1. Pengguna memasukkan kata kunci pencarian.
2. Memilih algoritma pencocokan: KMP atau BM.
3. Menentukan jumlah hasil yang ingin ditampilkan.
4. Menekan tombol Search.
5. Sistem menampilkan daftar CV yang paling relevan, disertai tombol untuk melihat detail (*Summary*) atau CV asli (*View CV*).

## BAB 2 LANDASAN TEORI

### 2.1. Dasar Teori

Pattern matching merupakan salah satu permasalahan fundamental dalam ilmu komputer, khususnya dalam bidang algoritma string, yang bertujuan untuk menemukan kemunculan suatu pola (*pattern*) di dalam sebuah teks yang lebih besar. Pattern matching dapat didefinisikan sebagai proses pencarian substring P (*pattern*) dengan panjang m di dalam string T (*text*) dengan panjang n, di mana  $m \leq n$ . Tujuan utama dari proses ini adalah untuk menentukan apakah P muncul di dalam T, dan jika ya, mengidentifikasi posisi kemunculannya secara tepat. Teknik pattern matching telah banyak diaplikasikan dalam berbagai bidang, seperti pencarian di dalam editor teks, web search engine, analisis citra, hingga bioinformatics.

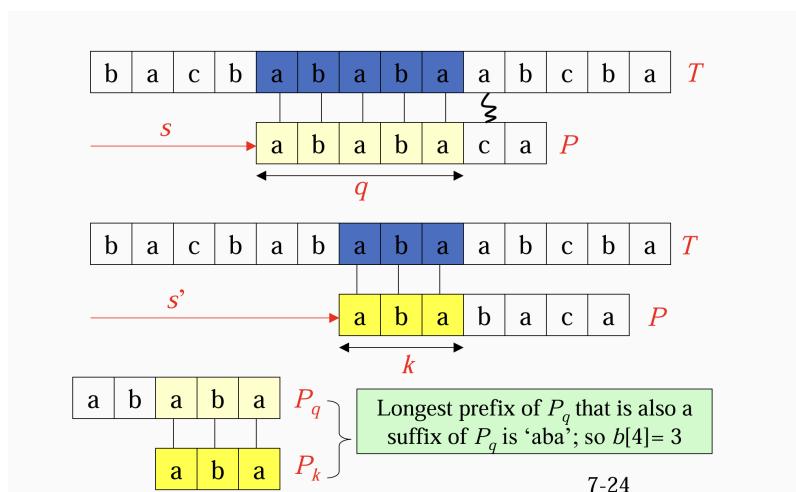
Secara umum, tujuan dari pattern matching adalah untuk menemukan semua posisi di dalam T di mana P muncul, baik secara eksak (*exact match*) maupun secara mirip (*fuzzy match*). Konsep *exact match* mengharuskan seluruh karakter dalam P cocok secara urutan dan posisi dengan substring dalam T. Sementara itu, pencarian *fuzzy match* memperbolehkan sejumlah perbedaan, misalnya seperti kesalahan ketik atau variasi berbeda dalam penulisan, sehingga pencarian tidak seketet yang dilakukan dalam *exact matching*. Terdapat berbagai jenis algoritma *pattern matching* yang telah dikembangkan, beberapa diantaranya adalah Knuth-Morris-Pratt (KMP), Boyer-Moore (BM), dan Aho-Corasick (AC).

#### 2.1.1. Algoritma Knuth-Morris-Pratt

Algoritma Knuth-Morris-Pratt (KMP) merupakan salah satu algoritma *pattern matching* yang efisien untuk menemukan kemunculan *pattern* di dalam sebuah teks. Algoritma ini dikembangkan oleh Donald Knuth, James Morris, dan Vaughan Pratt pada tahun 1977. Tujuan utama dari algoritma ini adalah untuk mencari kemunculan sebuah pola (*pattern*) dalam sebuah teks dengan menghindari perbandingan ulang terhadap karakter-karakter yang telah diperiksa sebelumnya. Tujuan utama dari algoritma ini adalah untuk menemukan kemunculan sebuah pola (*pattern*) dalam sebuah teks (*text*) tanpa perlu melakukan perbandingan ulang terhadap karakter-karakter yang sebelumnya telah diperiksa. Untuk mencapai efisiensi ini, KMP terlebih dahulu mengevaluasi struktur internal dari pola sebelum proses pencocokan dimulai. Keunggulan utama KMP dibandingkan metode pencarian sederhana adalah kemampuannya untuk menghindari perulangan pemeriksaan karakter pada teks, sehingga meningkatkan efisiensi pencarian.

Pada dasarnya, KMP melakukan pencarian pola pada sebuah teks dengan memeriksa karakter secara berurutan dari kiri ke kanan, mirip dengan algoritma brute force. Namun, KMP memiliki keunggulan dibandingkan metode brute force karena algoritma ini mampu menggeser pola dengan lebih cerdas. Jika pada brute force setiap terjadi ketidakcocokan pemeriksaan akan kembali ke posisi awal berikutnya, KMP dapat memanfaatkan informasi dari hasil pencocokan sebelumnya untuk menghindari perbandingan yang tidak perlu.

Ketika terjadi ketidakcocokan antara karakter pada teks dan pola pada posisi tertentu, KMP akan menghitung pergeseran optimal untuk pola tersebut agar proses pencocokan berikutnya tidak mengulang perbandingan yang sudah pernah dilakukan. Pergeseran pola ini ditentukan berdasarkan bagian pola yang sudah cocok sebelumnya. Secara khusus, jika terjadi mismatch pada  $T[i] \neq P[j]$ , pola dapat digeser sedemikian rupa sehingga prefiks terbesar dari  $P[0\dots j-1]$  yang juga merupakan sufiks dari  $P[1\dots j-1]$  akan dijadikan titik awal pencocokan selanjutnya. Dengan strategi ini, KMP mampu menghindari pencocokan ulang pada bagian-bagian teks yang pasti tidak akan menghasilkan kecocokan sehingga mengoptimalkan proses pencarian pola secara signifikan.



Gambar 2.1.1. Ilustrasi Algoritma KMP

(Sumber: [https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/23-Pencocokan-string-\(2025\).pdf](https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/23-Pencocokan-string-(2025).pdf))

Secara teknis, jika terjadi mismatch pada posisi ke- $j$  dalam pattern, maka  $k$  adalah posisi sebelum mismatch ( $k = j - 1$ ). Pada situasi ini, algoritma memanfaatkan apa yang disebut fungsi pinggiran atau border function (sering juga disebut failure function atau fail). Fungsi pinggiran, yang biasanya dilambangkan sebagai  $b(k)$ , didefinisikan sebagai ukuran dari prefiks terbesar dari  $P[0\dots k]$  yang juga merupakan sufiks dari  $P[1\dots k]$ . Nilai  $b(k)$  ini menunjukkan berapa banyak karakter dari awal pola yang sudah pasti cocok dan dapat digunakan kembali pada langkah berikutnya setelah terjadi mismatch.

P: abaaba  
j: 012345

( $k = j-1$ )

<i>j</i>	0	1	2	3	4	5
<i>P[j]</i>	a	b	a	a	b	a

<i>k</i>	0	1	2	3	4
<i>b(k)</i>	0	0	1	1	2

$b(k)$  is the size of the largest border.

Gambar 2.1.2. Fungsi Pinggiran

(Sumber: [https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/23-Pencocokan-string-\(2025\).pdf](https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/23-Pencocokan-string-(2025).pdf))

Sebagai contoh, jika diberikan pattern "abaaba", proses perhitungan fungsi pinggiran dilakukan untuk setiap posisi  $k$  (dimulai dari -1 atau 0 tergantung konvensi). Tabel fungsi pinggiran  $b(k)$  menyimpan untuk setiap posisi  $k$ , nilai ukuran pinggiran terbesar yang ditemukan. Pada contoh ini,  $b(0) = 0, b(1) = 0, b(2) = 1, b(3) = 1, b(4) = 2$ , dan seterusnya, yang berarti pada setiap posisi, fungsi pinggiran memberikan informasi tentang bagian pola mana yang bisa dipakai ulang setelah mismatch.

Dalam implementasi kode, fungsi pinggiran biasanya direpresentasikan sebagai array, di mana setiap elemen array menyimpan ukuran pinggiran terbesar untuk posisi tertentu pada pattern. Dengan adanya fungsi pinggiran ini, proses pencarian pola pada teks menjadi jauh lebih efisien karena meminimalisasi perbandingan ulang karakter yang sudah dicocokkan sebelumnya.

Kompleksitas waktu algoritma KMP terdiri dari dua bagian, yaitu perhitungan fungsi pinggiran dan proses pencarian string. Perhitungan fungsi pinggiran membutuhkan waktu  $O(m)$  dengan  $m$  adalah panjang pattern yang dicari, sedangkan proses pencarian string membutuhkan waktu  $O(n)$  dengan  $n$  adalah panjang teks. Dengan demikian, kompleksitas waktu keseluruhan dari algoritma KMP adalah  $O(m + n)$ . Kompleksitas ini jauh lebih efisien dibandingkan algoritma brute force, sehingga KMP cocok untuk pencarian pada data berskala besar.

Salah satu keunggulan utama algoritma KMP adalah algoritma ini tidak pernah perlu bergerak mundur pada teks masukan (input text, T). Hal ini menjadikan KMP sangat efektif untuk memproses file berukuran sangat besar yang dibaca secara berurutan, baik dari perangkat eksternal maupun dari jaringan (network stream).

Namun, KMP memiliki beberapa keterbatasan. Kinerja KMP menurun seiring dengan meningkatnya jumlah karakter berbeda dalam teks dan pattern. Semakin besar ukuran alfabet, peluang terjadinya mismatch juga semakin besar. Mismatch yang terjadi cenderung muncul lebih awal pada pola dan KMP akan memberikan keuntungan maksimal ketika mismatch terjadi di bagian akhir pola. Meskipun demikian, pada banyak kasus praktis, efisiensi KMP masih lebih baik dibandingkan metode pencarian sederhana, terutama pada teks yang sangat besar dan pola yang relatif panjang.

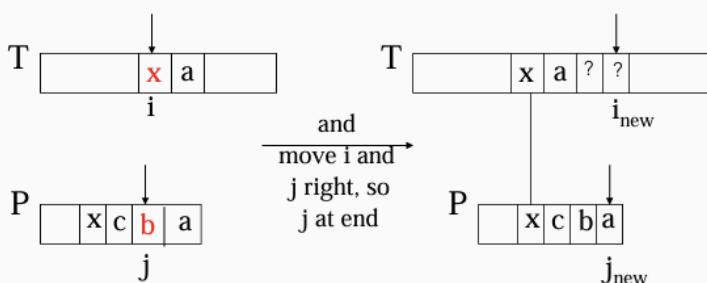
## 2.1.2. Algoritma Boyer-Moore

Algoritma Boyer-Moore merupakan salah satu algoritma *pattern matching* yang sangat efisien, terutama ketika digunakan untuk teks panjang dengan alfabet yang besar. Algoritma ini dikembangkan oleh Robert S. Boyer dan J Strother Moore pada tahun 1977. Algoritma ini menggunakan pendekatan berupa membandingkan karakter dari kanan ke kiri dalam pola. Ketika terjadi mismatch, algoritma ini tidak langsung menggeser pola satu per satu, melainkan memanfaatkan informasi dari pola itu sendiri untuk menentukan seberapa jauh pola dapat digeser. Pendekatan ini memungkinkan Boyer-Moore untuk sering kali melewati sejumlah besar karakter teks tanpa perlu mencocokkan satu per satu, menjadikannya sangat efisien.

Algoritma Boyer-Moore menggabungkan dua teknik utama, yaitu *looking-glass technique* dan *character-jump technique*. *Looking-glass technique* berarti bahwa pencocokan dilakukan mulai dari karakter terakhir dalam pola, bukan dari awal. Jika ditemukan ketidaksesuaian, algoritma kemudian menerapkan *character-jump technique*, yang menggunakan informasi karakter yang menyebabkan mismatch untuk menentukan seberapa jauh pola sebaiknya digeser. Informasi ini didapat dari fungsi *last occurrence*, yang mencatat posisi terakhir kemunculan setiap karakter dalam pola. Jika karakter yang menyebabkan mismatch tidak muncul dalam pola, maka pola dapat langsung digeser penuh sejauh panjangnya. Namun jika karakter tersebut muncul, maka pola digeser sehingga posisi terakhir karakter tersebut dalam pola sejajar dengan posisi karakter tersebut dalam teks.

### Case 1

- If P contains x somewhere, then try to *shift P* right to align the last occurrence of x in P with T[i].



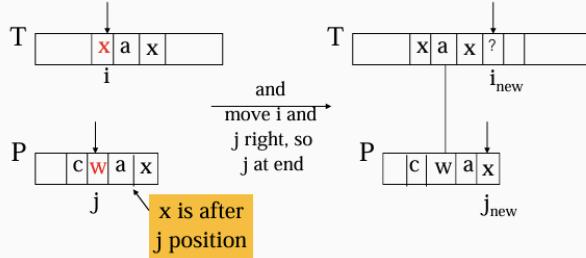
Gambar 2.2.1. Ilustrasi kasus character-jump technique

(Sumber: [https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/23-Pencocokan-string-\(2025\).pdf](https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/23-Pencocokan-string-(2025).pdf))

Kasus diatas terjadi ketika karakter penyebab mismatch (x) ada dalam pola, dan posisinya < j. Jika kasus karakter x yang menyebabkan mismatch muncul dalam pola dan posisinya sebelum j terjadi, maka pola digeser sehingga kemunculan terakhir x dalam pola sejajar dengan T[i]. Hal ini adalah *shifting optimal*.

### Case 2

- If P contains x somewhere, but a shift right to the last occurrence is *not* possible, then *shift P* right by 1 character to T[i+1].



45

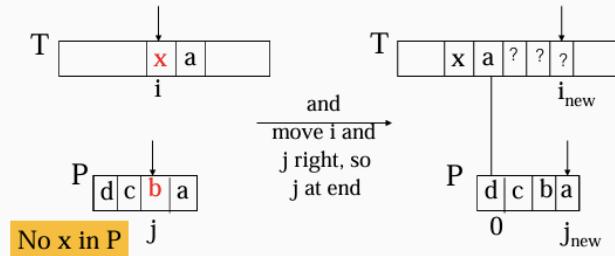
Gambar 2.2.2. Ilustrasi kasus character-jump technique

(Sumber: [https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/23-Pencocokan-string-\(2025\).pdf](https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/23-Pencocokan-string-(2025).pdf))

Kasus kedua terjadi ketika karakter x ada dalam pola, tapi posisinya  $\geq j$ . Jika kasus karakter ada dalam pola, tapi posisinya  $\geq j$  ini terjadi, maka tidak aman untuk menyalaskan posisi itu dengan T[i] karena bisa menyebabkan pencocokan mundur. Sehingga, pola hanya digeser satu posisi ke kanan.

### Case 3

- If cases 1 and 2 do not apply, then *shift P* to align P[0] with T[i+1].



Gambar 2.2.3. Ilustrasi kasus character-jump technique

(Sumber: [https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/23-Pencocokan-string-\(2025\).pdf](https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/23-Pencocokan-string-(2025).pdf))

Kasus ketiga ini terjadi ketika karakter x tidak ada dalam pola. Jika terjadi kasus karakter x tidak ditemukan dalam pola sama sekali, maka pola dapat digeser penuh hingga karakter pertama pola (P[0]) sejajar dengan T[i+1]. Hal ini dikarenakan tidak ada kemungkinan pola cocok dengan karakter tersebut.

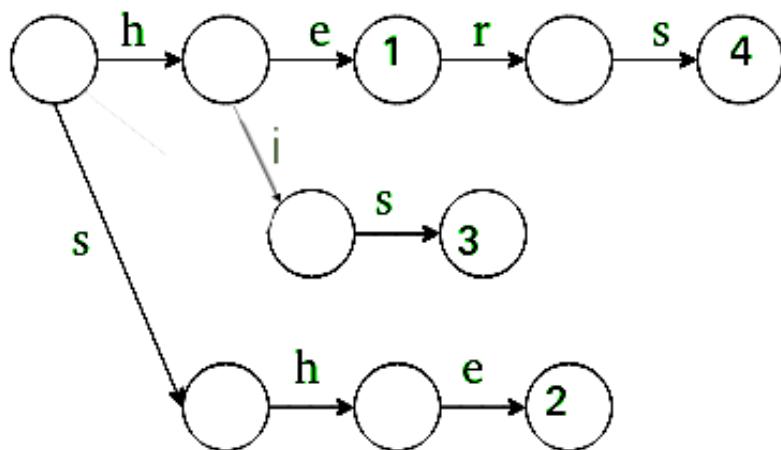
Dalam implementasinya, algoritma Boyer-Moore terdiri atas dua tahap utama. Tahap pertama adalah pra-pemrosesan pola untuk menghitung fungsi last occurrence, yang dilakukan dalam waktu  $O(m + \sigma)$ , dengan  $m$  adalah panjang pola dan  $\sigma$  adalah ukuran alfabet. Tahap kedua adalah proses pencocokan teks, di mana pola dibandingkan terhadap teks dari kanan ke kiri. Ketika ditemukan mismatch, algoritma menghitung pergeseran menggunakan hasil fungsi last occurrence dan kembali mencocokkan dari posisi paling kanan pola.

### 2.1.3 Algoritma Aho-Corasick

Algoritma Aho-Corasick (AC) merupakan algoritma pattern matching yang dirancang khusus untuk mencari sejumlah besar pattern secara bersamaan di dalam satu buah teks. AC dikembangkan oleh Alfred V. Aho dan Margaret J. Corasick pada tahun 1975, dan banyak digunakan dalam aplikasi seperti pencarian kata kunci pada dokumen besar, pemeriksaan kamus, serta pendekripsi malware.

Aho-Corasick melakukan preprocessing terhadap seluruh kumpulan pattern yang ingin dicari. Semua pattern tersebut dimasukkan ke dalam struktur data trie, di mana setiap simpul (node) mewakili karakter dari pattern, dan setiap jalur dari akar ke daun merepresentasikan satu pattern. Selain itu, AC membangun fungsi bantu berupa failure function dan output function. Failure function di sini berfungsi seperti pada KMP, yaitu untuk menangani kondisi mismatch; ketika tidak ada kelanjutan kecocokan di trie, algoritma dapat melakukan transisi ke node lain yang sesuai tanpa perlu memulai pencocokan dari awal. Output function menyimpan semua pattern yang dikenali hingga posisi tertentu, sehingga ketika pencarian mencapai node tersebut, seluruh pattern yang cocok dapat langsung diketahui.

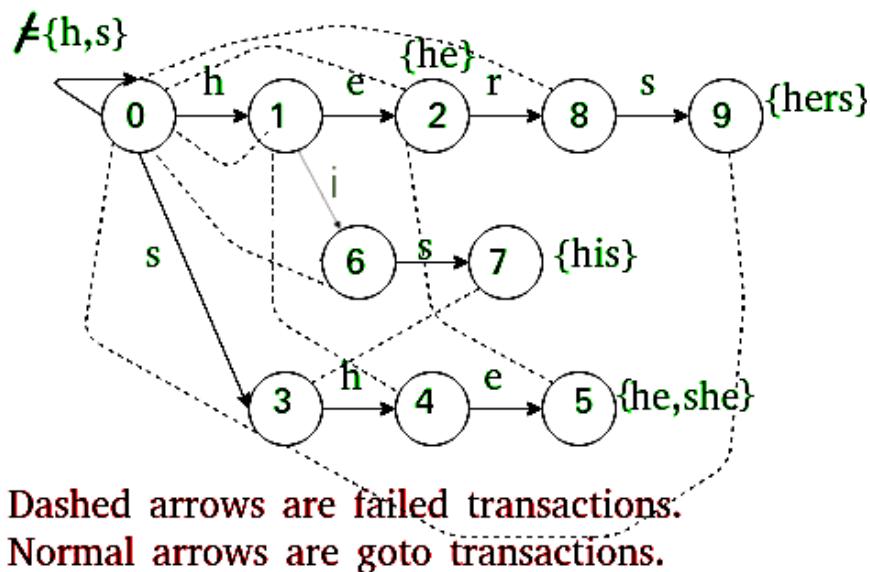
**Trie for  $\text{Arr}[] = \{ \text{he} , \text{she}, \text{his} , \text{hers} \}$**



Gambar 2.3.1. Inisialisasi trie dari semua pattern

(Sumber: <https://www.geeksforgeeks.org/dsa/aho-corasick-algorithm-pattern-searching/>)

Sebagai contoh, misalkan terdapat empat pattern: "he", "she", "his", dan "hers". Algoritma AC akan membangun trie dari ketiga pattern tersebut, dan jika proses pencarian pada teks gagal menemukan karakter selanjutnya dalam jalur trie (mismatch), failure function akan menentukan node alternatif dengan prefiks terpanjang yang masih cocok, serupa prinsip pada KMP namun diterapkan untuk banyak pattern sekaligus. Dengan pendekatan ini, seluruh proses pencarian pada teks dapat dilakukan dalam satu kali traversal dari kiri ke kanan, tanpa perlu mengulang-ulang pencarian setiap pattern satu per satu.



Gambar 2.3.2. Perluasan trie menjadi automaton

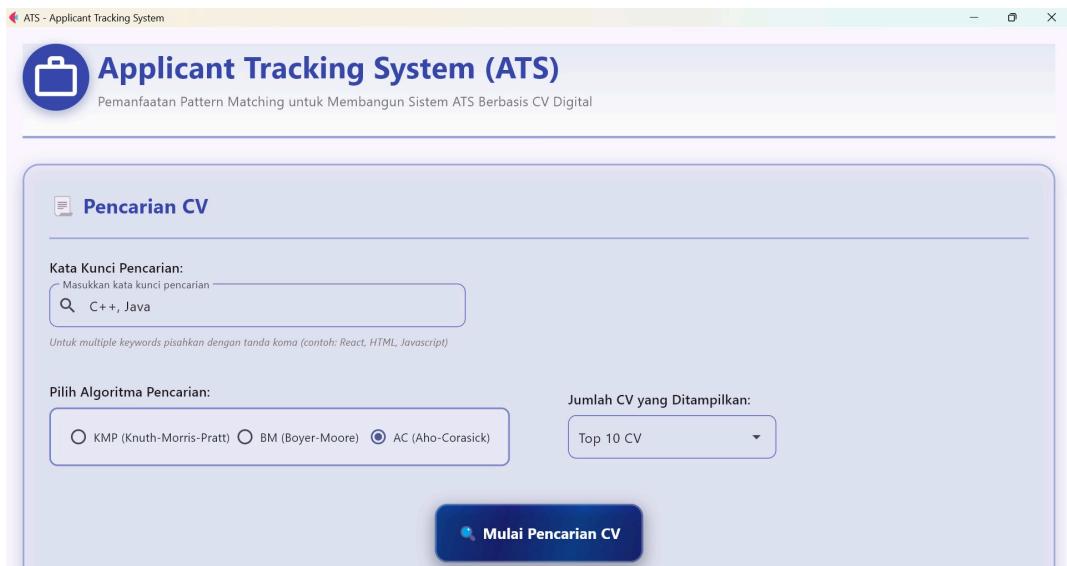
(Sumber: <https://www.geeksforgeeks.org/dsa/aho-corasick-algorithm-pattern-searching/>)

Kompleksitas waktu dari algoritma Aho-Corasick adalah  $O(n + m + z)$ , dengan  $n$  adalah panjang teks,  $m$  adalah total panjang seluruh pattern, dan  $z$  adalah jumlah total kecocokan pattern yang ditemukan. Preprocessing trie dan failure function dilakukan dalam  $O(m)$ , pencarian dalam  $O(n)$ , dan output kecocokan dalam  $O(z)$ . Efisiensi ini menjadikan AC sangat unggul pada kasus multi-pattern matching dalam data berukuran besar.

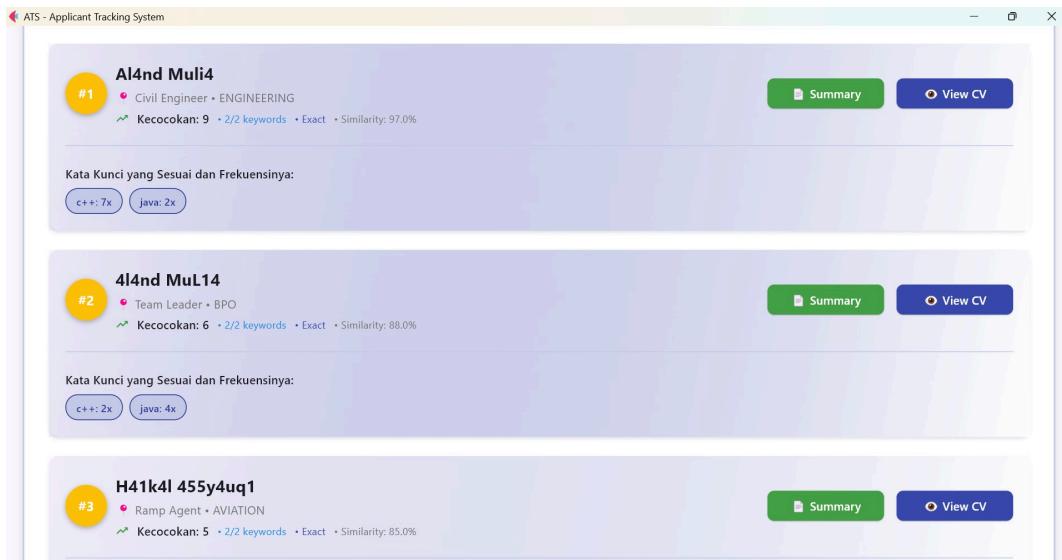
Keunggulan utama Aho-Corasick adalah kemampuannya melakukan pencarian banyak pola sekaligus dalam satu kali pemindai teks, sehingga jauh lebih cepat dibandingkan menjalankan algoritma single-pattern seperti KMP atau Boyer-Moore secara berulang. Selain itu, AC sangat efektif pada aplikasi yang membutuhkan pencarian sejumlah besar keyword secara real time.

Namun, algoritma ini juga memiliki beberapa kekurangan, seperti kebutuhan memori yang besar untuk membangun trie dan failure function, khususnya jika jumlah atau panjang pattern sangat banyak. Selain itu, implementasi AC relatif lebih kompleks dibandingkan KMP dan BM, sehingga kurang sesuai untuk aplikasi dengan resource terbatas atau kebutuhan pencarian pola yang sangat sedikit.

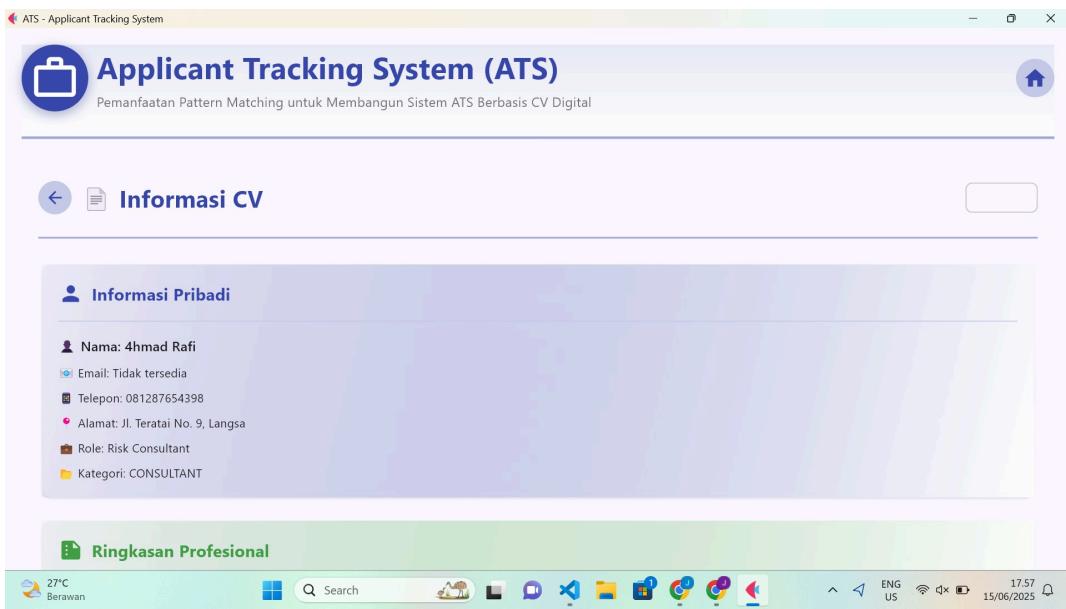
## 2.2. Penjelasan singkat mengenai aplikasi web yang dibangun



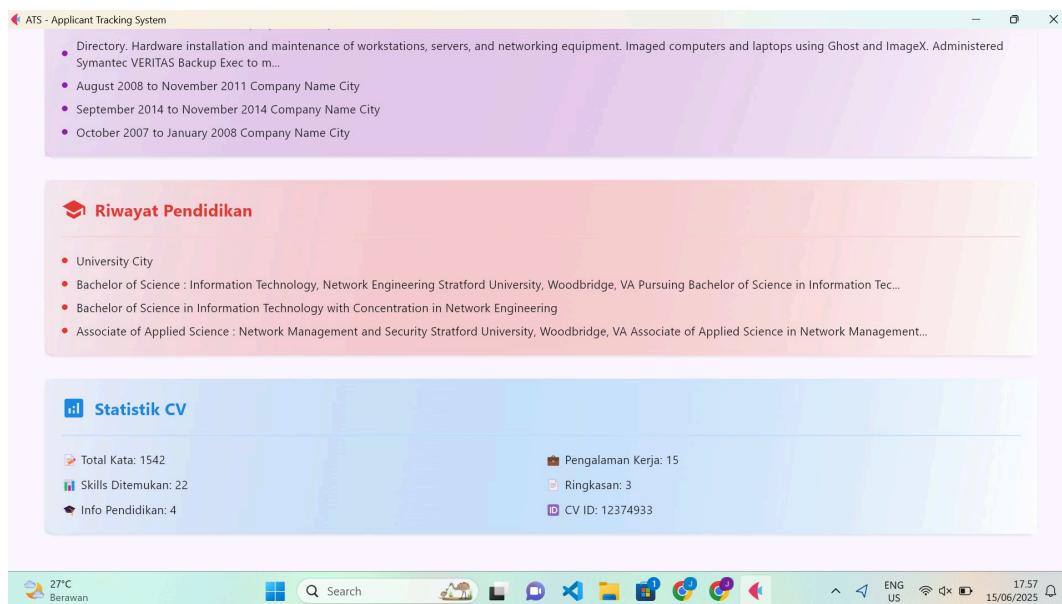
Gambar 2.1. Tampilan awal program



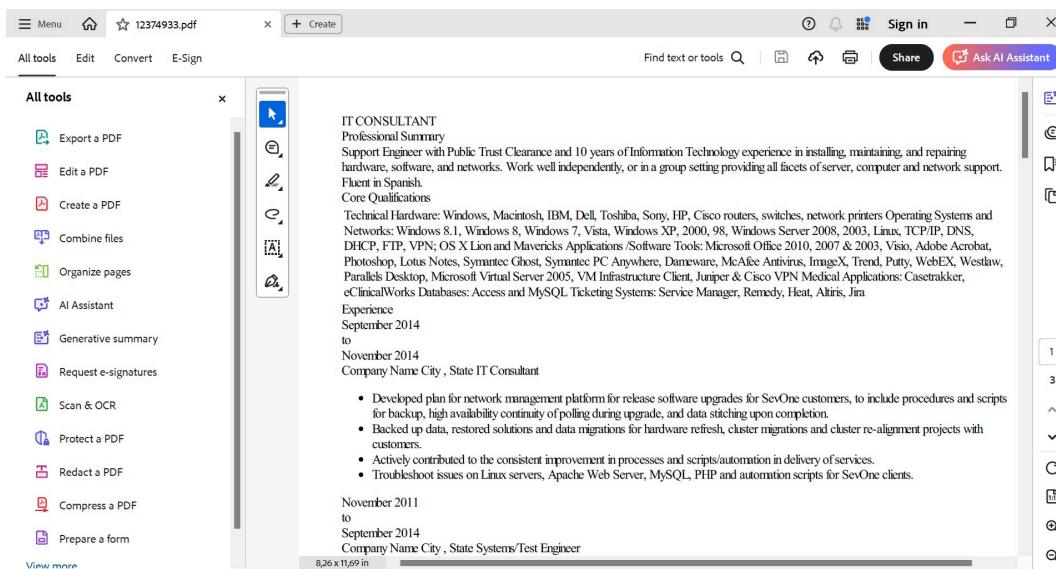
Gambar 2.2 Tampilan hasil pencarian



**Gambar 2.3. Contoh Tampilan Summary CV**



**Gambar 2.4 Contoh Tampilan Summary CV**



**Gambar 2.4 Contoh Tampilan View CV**

Aplikasi web yang dibangun berjudul “Applicant Tracking System (ATS)”. Aplikasi ini dirancang untuk membantu proses pencarian dan pencocokan CV digital dengan memanfaatkan kata kunci (*keywords*) yang dimasukkan oleh pengguna. Tujuan utama dari aplikasi ini adalah untuk menyaring dan menampilkan CV pelamar kerja yang paling relevan berdasarkan kecocokan kata kunci dengan isi dokumen CV, dengan memanfaatkan algoritma *pattern matching*.

Melalui antarmuka yang sederhana dan interaktif, pengguna dapat memilih untuk memasukkan satu atau lebih kata kunci pencarian, misalnya "Python", "React", atau "Java". Jika pengguna ingin memasukkan lebih dari satu kata kunci, maka ia harus memisahkannya dengan menggunakan koma. Selanjutnya, pengguna dapat memilih algoritma pencarian yang ingin digunakan untuk pencocokan kata kunci, yaitu Knuth-Morris-Pratt (KMP), Boyer-Moore (BM), atau Aho-Corasick. Aplikasi juga menyediakan opsi untuk menentukan jumlah hasil CV yang ingin ditampilkan, seperti Top 5, Top 10, Top 15, Top 20, Top 25 atau semua hasil yang cocok. Setelah pencarian dijalankan, sistem akan menampilkan daftar CV yang telah diurutkan berdasarkan jumlah kecocokan kata kunci terbanyak.

Setiap hasil pencarian ditampilkan dalam bentuk kartu CV yang berisi informasi dasar pelamar, persentase kemiripan pencocokan, dan daftar kata kunci yang ditemukan beserta frekuensi kemunculannya. Selain itu, setiap kartu memiliki dua tombol aksi, yaitu tombol *Summary* untuk menampilkan ringkasan hasil ekstraksi informasi dari CV menggunakan regular expression, serta tombol *View CV* untuk melihat isi dokumen CV asli dalam format PDF. Jika tidak ditemukan kecocokan secara exact match, sistem secara otomatis akan melakukan

pencarian fuzzy menggunakan algoritma Levenshtein Distance untuk mendeteksi kemiripan berdasarkan toleransi terhadap kesalahan pengetikan atau variasi kata.

Aplikasi ini dikembangkan berbasis Python dan Flet sebagai GUI yang mengimplementasikan algoritma pattern matching dan pemrosesan teks. Antarmuka frontend dirancang agar mudah digunakan oleh semua orang. Informasi CV disimpan dalam basis data MySQL, sementara file dokumen CV berada dalam direktori lokal sistem. Dengan pendekatan ini, aplikasi *Applicant Tracking System (ATS)* diharapkan akan mampu menyediakan pencarian CV yang cepat, akurat, dan dapat disesuaikan dengan kebutuhan pengguna.

## **BAB 3 ANALISIS PEMECAHAN MASALAH**

### **3.1. Langkah-Langkah Pemecahan Masalah**

Permasalahan utama yang harus diselesaikan dalam Tugas Besar 3 Mata Kuliah IF2211 Strategi Algoritma ini adalah merancang sebuah sistem *Applicant Tracking System* (ATS) berbasis CV digital yang mampu melakukan pencarian dan pencocokan data pelamar kerja secara efisien berdasarkan input kata kunci dari pengguna. Tantangan utamanya adalah melakukan pencocokan informasi dalam dokumen CV yang tidak terstruktur secara tepat dan cepat, menggunakan algoritma pencocokan string seperti Knuth-Morris-Pratt (KMP), Boyer-Moore (BM), dan Aho-Corasick (AC). Untuk menyelesaikan permasalahan ini, langkah-langkah yang dilakukan adalah sebagai berikut:

#### **1. Input Kata Kunci dan Pemilihan Algoritma oleh Pengguna**

Langkah pertama dimulai dengan menerima masukan dari pengguna berupa daftar kata kunci pencarian, seperti keahlian, riwayat pendidikan atau kata kunci lain yang relevan dengan tipe CV yang ingin dicari. Pengguna juga diminta memilih algoritma pencocokan yang akan digunakan, yaitu Knuth-Morris-Pratt (KMP), Boyer-Moore (BM), atau Aho-Corasick (AC). Setelah itu, pengguna juga harus menentukan berapa jumlah hasil CV yang ingin ditampilkan berdasarkan relevansi tertinggi. Input yang diberikan oleh

pengguna ini akan menjadi dasar proses pencarian.

## 2. Ekstraksi Teks dari Dokumen CV Digital

Setelah menerima input, sistem kemudian akan mengakses dokumen CV yang tersimpan dalam basis data, lalu melakukan proses ekstraksi teks dari file PDF menggunakan parser PDF. Setiap dokumen diubah menjadi string panjang yang mewakili isi lengkap CV dalam bentuk linier, agar dapat diolah oleh algoritma pattern matching dengan efisien. Proses ini dilakukan secara otomatis dan menyeluruh untuk seluruh dokumen.

## 3. Pencocokan Pola dengan Algoritma String Matching

Selanjutnya, sistem akan mencocokkan setiap kata kunci dari pengguna dengan isi teks CV menggunakan algoritma pencocokan yang telah dipilih di awal. Jika menggunakan KMP, maka terlebih dahulu dibentuk *border function* dari pola, lalu dilakukan pencocokan linier. Jika menggunakan Boyer-Moore, sistem membentuk *last occurrence table* dan melakukan pencocokan dari kanan ke kiri. Jika menggunakan Aho-Corasick, semua kata kunci dibangun dalam bentuk trie dan dilakukan pencarian sekaligus untuk semua pola. Seluruh proses dilakukan in-memory.

## 4. Perhitungan Similarity dengan Levenshtein Distance (Jika Diperlukan)

Apabila tidak dapat ditemukan kecocokan secara eksak (*exact match*), sistem akan melanjutkan dengan perhitungan *fuzzy matching* menggunakan algoritma Levenshtein Distance. Untuk setiap kata kunci yang tidak cocok, dihitung jarak edit antara kata kunci dan kata-kata dalam CV. Jika kemiripan melebihi ambang batas tertentu, maka kata tersebut dianggap relevan. Proses ini memastikan agar hasil yang diberikan tetap akurat walaupun terdapat kesalahan ketik atau variasi penulisan.

## 5. Menyusun dan Menampilkan Hasil Pencarian

Kemudian, hasil pencarian yang diperoleh akan disusun berdasarkan jumlah kecocokan, baik exact maupun fuzzy. Lalu, ditampilkan dalam bentuk daftar CV dengan visualisasi kartu. Setiap kartu menampilkan nama pelamar, persentase kecocokan, kata kunci yang ditemukan beserta jumlah kemunculannya, serta tombol aksi untuk melihat ringkasan hasil ekstraksi atau membuka file CV asli. Selain itu, sistem juga menampilkan waktu eksekusi pencarian dan jumlah dokumen yang dipindai.

## 3.2. Proses Pemetaan Masalah Menjadi Elemen-Elemen Algoritma KMP, BM, AC

Pada aplikasi *Applicant Tracking System (ATS)* yang dibuat, setiap kata kunci yang dimasukkan oleh pengguna diperlakukan sebagai sebuah *pattern*,

sedangkan teks hasil ekstraksi dari file CV pelamar direpresentasikan sebagai *text*. Proses pencocokan antara *pattern* dan *text* ini, kemudian dilakukan dengan menggunakan algoritma string matching, yaitu Knuth-Morris-Pratt (KMP), Boyer-Moore (BM), atau Aho-Corasick (AC). Pemilihan algoritma ini dilakukan oleh pengguna sebelum pencarian dimulai. Hasil dari proses pencocokan ini akan berupa daftar kartu CV yang paling relevan dengan kata kunci yang dimasukkan oleh pengguna, serta jumlah dan lokasi kemunculannya dalam dokumen CV masing-masing orang. Meskipun tiga algoritma yang ada memiliki pemetaan proses pencocokan yang berbeda, namun semuanya memiliki tujuan akhir yang sama, yaitu menemukan pola dalam teks seefisien mungkin.

### 3.2.1. Knuth-Morris-Pratt (KMP)

Algoritma KMP dalam program ini digunakan untuk mencocokkan satu kata kunci terhadap isi teks dari satu dokumen CV dengan pendekatan *linear string matching*. Algoritma KMP menghindari perbandingan ulang saat mismatch dengan memanfaatkan border function. Berikut adalah langkah-langkahnya:

1. Langkah pertama. setiap kata kunci yang dimasukkan oleh pengguna akan diperlakukan sebagai pola (*pattern*).
2. Untuk setiap *pattern*, sistem akan membentuk *border function*, yaitu array yang menyimpan informasi prefix dan suffix dalam pattern tersebut.
3. Sistem akan mencocokkan pattern terhadap teks isi CV dari kiri ke kanan.
4. Jika ditemukan *mismatch*, sistem akan menggeser pola berdasarkan nilai border function tanpa mengulangi perbandingan karakter sebelumnya.
5. Setiap kali ditemukan kecocokan penuh antara pattern dan bagian dari teks CV, informasi posisi kemunculan dan jumlah kecocokan disimpan.
6. Proses diulang untuk seluruh kata kunci terhadap seluruh dokumen CV yang tersedia.
7. Terakhir, hasil berupa jumlah keyword yang cocok dan lokasi kemunculannya akan dikembalikan dan digunakan untuk menentukan ranking relevansi data-data CV.

### **3.2.2. Boyer-Moore (BM)**

Algoritma BM dalam program ini digunakan untuk mencocokkan pattern terhadap teks dengan pendekatan pencocokan dari kanan ke kiri dan melakukan lompatan besar saat mismatch terjadi, sesuai dengan strategi *character-jump*. Berikut adalah langkah-langkahnya:

1. Langkah pertama. setiap kata kunci dari pengguna akan diperlakukan sebagai sebuah *pattern*. Untuk setiap *pattern*, sistem akan kemudian membentuk *last occurrence table* dari semua karakter dalam *pattern*.
2. Proses pencocokan kata kunci dilakukan dari kanan ke kiri dalam *pattern* terhadap isi CV yang telah diekstraksi.
3. Jika terjadi *mismatch*, sistem akan menggeser *pattern* berdasarkan posisi terakhir karakter *mismatch* dalam *pattern* sesuai nilai pada tabel *last occurrence*.
4. Jika karakter tidak terdapat dalam *pattern*, maka *pattern* akan digeser sejauh panjang *pattern*.
5. Jika ditemukan kecocokan, informasi posisi dan frekuensi kata kunci dicatat oleh sistem.
6. Proses akan terus dilakukan untuk setiap *pattern* terhadap semua dokumen CV dalam database. Terakhir, hasil akhir akan dikumpulkan untuk penilaian relevansi data masing-masing CV.

### **3.2.3. Aho-Corasick (AC)**

Algoritma Aho-Corasick digunakan dalam program ini untuk melakukan pencocokan banyak kata kunci secara paralel dan efisien dalam satu lintasan teks. Berikut adalah langkah-langkahnya:

1. Langkah pertama. semua kata kunci yang dimasukkan oleh pengguna akan dikonstruksi menjadi sebuah trie yang merepresentasikan semua *pattern*.
2. Trie yang dibuat dilengkapi dengan *failure link* untuk menangani *mismatch* dan *output link* untuk mencatat hasil kecocokan.
3. Sistem melakukan satu kali traversal terhadap teks hasil ekstraksi dari CV, karakter demi karakter.
4. Saat traversal mencapai state dengan output, artinya ada kata kunci yang cocok dan hal ini akan langsung dicatat.

5. Semua kecocokan terhadap semua *pattern* dilakukan secara sekaligus dalam satu kali lintasan. Proses ini sangat efisien untuk *pattern* dan teks panjang, karena tidak perlu pengulangan pencocokan untuk setiap kata kunci.
6. Terakhir, output berupa daftar kata kunci yang cocok, frekuensi kemunculan, dan posisi match akan dikembalikan oleh algoritma sebagai dasar penilaian relevansi CV.

### 3.3 Fitur Fungsional dan Arsitektur Aplikasi Web yang Dibangun

#### 3.3.1. Fitur Fungsional

Aplikasi web yang dikembangkan merupakan sebuah Applicant Tracking System (ATS) yang bertujuan untuk membantu proses seleksi pelamar kerja berbasis CV digital, dengan menggunakan teknik *pattern matching*. Fitur-fitur fungsional dari aplikasi ini adalah:

##### 1. Pencarian Kata Kunci dalam CV Digital

Pengguna dapat memasukkan satu atau lebih kata kunci pencarian, misalnya (C++, react) yang akan dicocokkan dengan isi CV setiap pelamar. Sistem akan mencari kecocokan menggunakan algoritma pencocokan string. Lalu, mengembalikan ranking urutan CV yang paling relevan dari hasil tersebut.

##### 2. Pilihan Algoritma *Pattern Matching*

Pengguna dapat memilih satu dari tiga algoritma *pattern matching*, yaitu **KMP (Knuth-Morris-Pratt)**, **BM (Boyer-Moore)**, atau **AC (Aho-Corasick)**. Pilihan algoritma yang dipilih oleh pengguna ini akan menentukan pendekatan pencarian *pattern* yang digunakan terhadap teks dalam CV.

##### 3. Opsi Jumlah Hasil Pencarian (Top CV Results)

Pengguna dapat menentukan jumlah hasil CV terbaik yang ingin ditampilkan, berdasarkan tingkat kecocokan keyword. Misalnya, seperti Top 5 CV, Top 10 CV, Top 15 CV, Top 20 CV, Top 25 CV atau semua hasil. CV akan diurutkan berdasarkan hasil yang paling cocok dengan kata kunci yang di input oleh pengguna.

##### 4. Tampilan Interaktif Hasil Pencarian

Setelah pencarian, sistem akan menampilkan daftar CV dengan urutan dari kecocokan tertinggi. Setiap entri yang ditampilkan akan memuat:

- Nama Pelamar
- Daftar keyword yang cocok beserta frekuensinya
- Skor similarity,
- Tombol Summary (untuk melihat ringkasan informasi CV)

- Tombol View CV (untuk membuka file CV asli).

## 5. Perhitungan Similarity Tambahan (Fuzzy Matching)

Jika keyword tidak ditemukan melalui proses perhitungan *exact matching*, maka sistem akan langsung menggunakan Levenshtein Distance untuk mengukur kemiripan string secara *fuzzy*. Hal ini berguna untuk memastikan sistem tetap menemukan kandidat yang paling relevan, meskipun kata kunci yang diberi pengguna tidak sama persis.

## 6. Ekstraksi Informasi Otomatis via Regex

Sistem secara otomatis mengekstrak informasi dari file CV, seperti:

- Nama lengkap, email, nomor telepon
- Riwayat pendidikan dan pekerjaan
- Keahlian (*skills*)
- Ringkasan (*summary*)

Ekstraksi ini dilakukan dengan *regular expression* dan hasilnya ditampilkan pada halaman summary.

### 3.3.2. Arsitektur Aplikasi Web yang Dibangun

Aplikasi Applicant Tracking System (ATS) ini dirancang menggunakan arsitektur client-server berbasis Python dan SQLite, dengan antarmuka pengguna berbasis Flet. Arsitektur ini memisahkan logika antarmuka pengguna dengan proses pencocokan string dan pengolahan data.

#### 1. Backend

Bagian backend pada program dikembangkan menggunakan bahasa pemrograman Python dengan framework SQLite untuk database. Fungsi-fungsi utama backend meliputi:

- Menyediakan **Database Management System** yang menangani penyimpanan dan pengambilan data CV.
- Menjalankan **algoritma pattern matching** untuk pencarian CV (KMP, Boyer-Moore, Aho-Corasick).
- Mengelola proses pencarian berdasarkan tipe, yaitu:
  - a. Single Keyword Search

- b. Multiple Keywords Search
  - c. Fuzzy Search (menggunakan Levenshtein Distance)
- Melakukan ekstraksi teks dari file PDF menggunakan PDF Extractor.
  - Mengimplementasikan Regex Extractor untuk mengidentifikasi informasi terstruktur (email, phone, skills).
  - Mengelola database dengan tiga tabel utama, yaitu Applicant Profile (data profil pelamar), ApplicationDetail (detail aplikasi dan path CV), serta CVContent (konten teks yang diekstrak dari CV)
  - Melakukan validasi input dan mengembalikan hasil pencarian dengan ranking berdasarkan relevansi dan frekuensi keyword.

## 2. Frontend

Bagian antarmuka pengguna pada program ini dibangun dengan menggunakan Flet sebagai GUI dan bahasa pemrograman Python. Fungsi-fungsi utama pada frontend meliputi:

- Membuat tampilan yang interaktif bagi pengguna untuk memilih algoritma pencarian, jumlah CV yang ingin ditampilkan, serta untuk memasukkan kata kunci.
- Mengirim permintaan pencarian ke backend database.
- Menampilkan hasil pencarian dalam bentuk CV cards yang dapat dipaginasi dengan informasi, yaitu:
  - a. Ranking berdasarkan relevansi
  - b. Jumlah kecocokan keyword
  - c. Tipe matching (exact/fuzzy)
  - d. Similarity score
  - e. Frekuensi kemunculan setiap keyword
- Menyediakan fitur detail view untuk melihat informasi lengkap CV.
- Menampilkan summary dari proses pencarian.

### **3.4. Contoh Ilustrasi Kasus**

Terdapat beberapa contoh kasus yang mungkin terjadi dalam penggunaan program Applicant Tracking System (ATS) ini, berikut adalah beberapa skenario kasus yang mungkin terjadi:

#### **1. Kasus 1: Pencarian CV dengan Exact Match Keywords**

Pada kasus ini, kata kunci yang dicari terdapat secara persis dalam dokumen CV. Karena tidak ada perbedaan penulisan, algoritma KMP, Boyer-Moore, maupun Aho-Corasick akan memberikan hasil yang identik dengan akurasi 100%. Contoh: pencarian kata "Python", "React", atau "MySQL" yang tertulis utuh dan konsisten dalam teks CV.

#### **2. Kasus 2: Pencarian CV dengan Multiple Keywords**

Pada kasus ini, pengguna memasukkan beberapa kata kunci sekaligus, seperti "Java, SQL, Docker". CV yang ditemukan bisa memuat sebagian atau seluruh keyword tersebut. Tingkat similarity akan bergantung pada jumlah kata kunci yang cocok (misalnya 33%, 66%, atau 100%). Pada kasus ini, algoritma Aho-Corasick akan unggul dalam performa karena mampu melakukan pencocokan multi-pattern dalam satu lintasan teks, sedangkan KMP dan Boyer-Moore membutuhkan iterasi per pattern.

#### **3. Kasus 3: Pencarian CV dengan Fuzzy Matching**

Kasus ini akan terjadi jika isi CV mengandung typo atau variasi penulisan berbeda dalam kata kunci, sehingga tidak cocok secara *exact*. Sistem akan memanfaatkan Levenshtein Distance untuk menghitung kemiripan antara keyword dan teks. Contohnya, pencarian keyword "JavaScript" dapat mencocokkan "JavaScript" dengan similarity 90%, atau "JS" dengan similarity lebih rendah.

#### **4. Kasus 4: Pencarian CV dengan Pattern Variations**

CV pelamar pada database kadang dapat menyebut keahlian dengan nama yang berbeda dari keyword standar. Misalnya, a keyword "Node.js" mungkin ditulis dalam CV sebagai "NodeJS", "Node", atau "Express.js". Meskipun istilah tersebut berkaitan, sistem perlu menggunakan pendekatan fuzzy matching untuk mendeteksi kesamaan kontekstual. Similarity yang diperoleh akan bervariasi tergantung pada representasi dan Levenshtein Distance.

#### **5. Kasus 5: Pencarian CV dengan Limited Context**

Pada kasus ini, CV yang digunakan dapat memiliki informasi yang terbatas atau tidak terstruktur dengan baik. Hal ini membuat extraction process menghasilkan data yang incomplete atau parsing yang tidak optimal. Pada kasus ini dapat terjadi similarity yang

rendah meskipun kandidat sebenarnya qualified, karena sistem bergantung pada quality dari extracted text dan structured information. PDF dengan format yang complex, scan quality yang buruk, atau layout yang non-standard dapat mempengaruhi akurasi pattern matching.

## **BAB 4 IMPLEMENTASI DAN PENGUJIAN**

## 4.1. Spesifikasi Teknis Program

### 4.1.1 Struktur Data Direktori

Program yang kami buat memiliki struktur direktori sebagai berikut:

```
TUBES3_PejuangCV-7 (Root Directory)
```

```
  └── .gitignore
  └── LICENSE
  └── README.md
  └── main.py
  └── requirements.txt
  |
  └── data
      └── .gitignore
      └── cv_lookup.csv
      └── extracted_cvs.csv
      └── seed_data.sql
      └── tubes3_seeding.sql
      └── cv
          └── ACCOUNTANT
          └── ADVOCATE
          └── AGRICULTURE
          └── APPAREL
          └── ARTS
          └── AUTOMOBILE
          └── AVIATION
```

```
|   └── BANKING
|   └── BPO
|   └── BUSINESS-DEVELOPMENT
|   └── CHEF
|   └── CONSTRUCTION
|   └── CONSULTANT
|   └── DESIGNER
|   └── DIGITAL-MEDIA
|   └── ENGINEERING
|   └── FINANCE
|   └── FITNESS
|   └── HEALTHCARE
|   └── HR
|   └── INFORMATION-TECHNOLOGY
|   └── PUBLIC-RELATIONS
|   └── SALES
|   └── TEACHER
|
└── src
    |
    |   └── algorithms
    |       |
    |       └── __init__.py
    |       └── aho_corasick.py
    |       └── bm.py
```

```
|   |   └── encryption.py
|   |   └── kmp.py
|   |   └── levenshtein.py
|   |   └── __pycache__
|   |
|   └── database
|       |   └── __init__.py
|       |   └── db_manager.py
|       |   └── dump_db.py
|       |   └── seed_db.py
|       |   └── __pycache__
|   |
|   └── frontend
|       |   └── __init__.py
|       |   └── app.py
|       |   └── components.py
|       |   └── event_handlers.py
|       |   └── utils.py
|   |
|   └── utils
|       |   └── __pycache__
|       |   └── __init__.py
|       |   └── cv2csv.py
|       |   └── pdf_extractor.py
```

```

|   └── regex_extractor.py

```

#### 4.1.2. Fungsi dan Prosedur yang Dibangun

##### 4.1.2.1. File: main\_old.py

Nama Fungsi	Kegunaan
__init__(self) : None	Inisialisasi aplikasi, koneksi DB, load seed & data ekstraksi, inisialisasi algoritma dan komponen.
load_seed_data(self) : None	Memuat data awal pelamar dan lamaran dari file SQL ke database jika kosong.
load_extracted_cv_data(self) : None	Memuat hasil ekstraksi teks CV dari CSV ke database.
load_cvs_from_db(self) : list[dict]	Mengambil seluruh data aplikasi dari database (list of dictionary).
main(self, Page) : None	Entry point aplikasi: inisialisasi UI Flet dan komponen utama.
Init_components(self) : None	Membuat dan menginisialisasi semua komponen UI.
on_button_hover(self, Event) : None	Mengatur efek animasi saat button di-hover.
create_header(self) : Row/Column	Membuat header aplikasi: logo, judul, dsb.
create_home_view(self) : Column	Membuat tampilan halaman utama (form input & hasil pencarian).
search_cv(self, Event) : None	Proses utama pencarian CV: matching, fuzzy, scoring, menampilkan hasil.
update_summary_result_section(self) : None	Update ringkasan hasil pencarian (jumlah hasil, waktu, algoritma).
get_paginated_results(self) : list[dict]	Mengambil hasil pencarian yang dipaginasi sesuai halaman.
update_pagination(self) : None	Update tampilan pagination di UI.

<b>Nama Fungsi</b>	<b>Kegunaan</b>
prev_page(self, Event) : None	Navigasi ke halaman sebelumnya pada pagination.
next_page(self, Event) : None	Navigasi ke halaman selanjutnya pada pagination.
go_to_page(self, int) : None	Navigasi langsung ke halaman tertentu.
update_results_display(self) : None	Menampilkan kartu hasil pencarian pada halaman.
create_cv_card(self, dict) : Card	Membuat tampilan kartu CV pada hasil pencarian.
show_summary(self, dict) : None	Menampilkan ringkasan (summary) detail data CV terpilih ke UI.
view_cv(self, dict) : None	Membuka file CV asli (PDF) menggunakan path yang disimpan.
go_to_home(self, Event) : None	Navigasi kembali ke halaman utama aplikasi.
show_snackbar(self, str, str) : None	Menampilkan notifikasi popup pada aplikasi.
main(Page) : None	Entry-point aplikasi, menjalankan instance ATSFrontend dengan halaman utama.

#### 4.1.2.2. File: cv2csv.py

<b>Nama Fungsi</b>	<b>Kegunaan</b>
extract_id_from_filename(str) : str	Mengekstrak 8 digit ID dari nama file PDF CV.
clean_text_for_csv(str) : str	Membersihkan teks hasil ekstraksi dari spasi berlebih dan newline, agar siap disimpan di format CSV.

<b>Nama Fungsi</b>	<b>Kegunaan</b>
generate_html_from_text(str, str) : str	Mengubah teks CV hasil ekstraksi menjadi HTML sederhana yang dipisah berdasarkan bagian summary, experience, education, skills.
process_cv_directory() : list[dict]	Mengekstrak seluruh CV (PDF) dalam direktori data/cv/, memproses isinya, dan mengembalikan list data CV terstruktur.
save_to_csv(cv_data: list[dict], output_file: str) : None	Menyimpan data CV hasil ekstraksi ke file CSV lengkap (extracted_cvs.csv).
create_lookup_csv(cv_data: list[dict], output_file: str) : None	Membuat file lookup CSV ringkas untuk pencarian cepat (cv_lookup.csv) dengan informasi keyword penting dan preview teks.
main() : None	Fungsi utama, menjalankan seluruh proses ekstraksi, penyimpanan ke CSV, dan menampilkan ringkasan hasil ekstraksi.

#### 4.1.2.3. File: aho\_corasick.py

<b>Nama Fungsi</b>	<b>Kegunaan</b>
add_pattern(str) : None	Menambahkan satu pattern ke trie untuk proses pencarian.

<b>Nama Fungsi</b>	<b>Kegunaan</b>
build_failure_links() : None	Membuat failure link untuk setiap node pada trie (untuk penanganan mismatch saat pencarian).
search_multiple(str, list) : dict	Mencari seluruh pattern dalam teks sekaligus, mengembalikan posisi kemunculan masing-masing.
search_single(str, str) : list	Mencari satu pattern dalam teks, mengembalikan list posisi kemunculannya.
count_occurrences(str, str) : int	Menghitung jumlah kemunculan satu pattern dalam teks.
search_case_insensitive(str, list) : dict	Pencarian pattern tidak case-sensitive (semua huruf dianggap kecil).
find_all_matches(str, list) : list	Mengembalikan semua detail kecocokan (pattern, posisi awal, posisi akhir, matched_text).

#### 4.1.2.4. File: bm.py

<b>Nama Fungsi</b>	<b>Kegunaan</b>
bad_char_heuristic(str) : dict	Membuat tabel posisi terakhir setiap karakter dalam pattern (heuristik bad character).

<b>Nama Fungsi</b>	<b>Kegunaan</b>
good_suffix_heuristic(str) : list	Membuat tabel pergeseran untuk pola berdasarkan suffix yang cocok (heuristik good suffix).
search(str, str) : int	Mencari posisi kemunculan pertama pattern dalam teks, mengembalikan indeks pertama (atau -1).
search_all(str, str) : list	Mencari semua posisi kemunculan pattern dalam teks, mengembalikan list indeks kemunculan.
count_occurrences(str, str) : int	Menghitung jumlah total kemunculan pattern dalam teks.
search_case_insensitive(str, str) : list	Mencari semua posisi kemunculan pattern secara case-insensitive.

#### 4.1.2.5. File: encryption.py

<b>Nama Fungsi</b>	<b>Kegunaan</b>
__init__(str) : None	Konstruktor kelas, menginisialisasi key dan parameter enkripsi.
_generate_caesar_shift(str) : int	Menghasilkan nilai shift untuk Caesar Cipher dari key yang diberikan.

<b>Nama Fungsi</b>	<b>Kegunaan</b>
_generate_xor_key(str) : bytes	Menghasilkan kunci XOR berupa bytes dari key yang diberikan.
_caesar_encrypt(str, int) : str	Melakukan enkripsi Caesar Cipher pada string dengan nilai shift tertentu.
_caesar_decrypt(str, int) : str	Melakukan dekripsi Caesar Cipher pada string dengan nilai shift tertentu.
_xor_encrypt_decrypt(bytes, bytes) : bytes	Melakukan operasi XOR pada data bytes dengan key bytes.
_custom_substitution(str, bool) : str	Melakukan substitusi karakter custom pada string (cipher ↔ plain).
encrypt(str) : str	Mengenkripsi plaintext menjadi ciphertext menggunakan beberapa lapisan enkripsi.
decrypt(str) : str	Mendekripsi ciphertext menjadi plaintext.
encrypt_dict(dict, list) : dict	Mengenkripsi field tertentu pada dictionary.
decrypt_dict(dict, list) : dict	Mendekripsi field tertentu pada dictionary.
verify_encryption(str, str) : bool	Memverifikasi apakah plaintext cocok dengan hasil dekripsi ciphertext.
change_key(str) : None	Mengganti key enkripsi, update parameter

<b>Nama Fungsi</b>	<b>Kegunaan</b>
	internal.
get_encryption_info() : dict	Mengembalikan info parameter enkripsi saat ini (shift, key, lapisan enkripsi, dsb).
generate_secure_key(int) : str	(Fungsi global) Menghasilkan key acak yang aman sepanjang n karakter.
encrypt_sensitive_cv_data(dict) : dict	(Fungsi global) Mengenkripsi field sensitif pada data CV berbentuk dictionary.
decrypt_sensitive_cv_data(dict) : dict	(Fungsi global) Mendekripsi field sensitif pada data CV berbentuk dictionary.

#### 4.1.2.6. File: kmp.py

<b>Nama Fungsi</b>	<b>Kegunaan</b>
compute_lps_array(str) : list[int]	Menghitung array LPS (Longest Prefix Suffix) untuk pattern yang diberikan.
search(str, str) : int	Mencari posisi kemunculan pertama pattern dalam teks, mengembalikan indeks pertama (atau -1 jika tidak ada)
search_all(str, str) : list[int]	Mencari semua posisi kemunculan pattern dalam teks, mengembalikan list indeks semua kemunculan.

<b>Nama Fungsi</b>	<b>Kegunaan</b>
count_occurrences(str, str) : int	Menghitung jumlah total kemunculan pattern dalam teks.
search_case_insensitive(str, str) : list[int]	Mencari semua posisi kemunculan pattern secara case-insensitive.

#### 4.1.2.7. File: levenshtein.py

<b>Nama Fungsi</b>	<b>Kegunaan</b>
__init__(None)	Konstruktor kelas, inisialisasi instance LevenshteinDistance.
distance(str, str) : int	Menghitung jarak edit (Levenshtein distance) antara dua string.
similarity(str, str) : float	Menghitung skor kemiripan (0.0–1.0) dua string berbasis Levenshtein.
similarity_percentage(str, str) : float	Mengembalikan kemiripan dua string dalam bentuk persentase (0–100).
is_similar(str, str, float) : bool	Menentukan apakah dua string mirip (di atas threshold tertentu).
find_closest_match(str, list) : tuple	Mencari string kandidat yang paling mirip dengan target, serta nilai similarity-nya.

<b>Nama Fungsi</b>	<b>Kegunaan</b>
find_all_similar(str, list, float) : list	Mengambil semua string kandidat yang similarity-nya di atas threshold.
distance_with_operations(str, str) : tuple	Menghitung jarak edit dan memberikan daftar operasi edit yang diperlukan.
normalized_distance(str, str) : float	Mengembalikan nilai distance yang dinormalisasi (0.0–1.0).

#### 4.1.2.4. File: db\_manager.py

<b>Nama Fungsi</b>	<b>Kegunaan</b>
__init__(str, int, str, str, str) : None	Konstruktor kelas, membuka koneksi ke database MySQL/MariaDB dan membuat tabel utama.
disconnect() : None	Menutup koneksi database.
create_tables() : None	Membuat tabel utama (ApplicantProfile, ApplicationDetail) di database jika belum ada.
insert_applicant(str, str, Optional[str], str, str) : int	Menambah data pelamar ke tabel ApplicantProfile, mengembalikan applicant_id baru.

<b>Nama Fungsi</b>	<b>Kegunaan</b>
insert_application(int, str, str) : int	Menambah data aplikasi lamaran ke tabel ApplicationDetail, mengembalikan detail_id baru.
get_all_applications() : List[Dict]	Mengambil seluruh data aplikasi (join profil pelamar & detail aplikasi).
get_all_applicants() : List[Dict]	Mengambil seluruh data profil pelamar.
import_csv_to_db(str) : None	Mengimpor data pelamar dan lamaran dari file CSV ke database.
import_sql_file(str) : None	Mengimpor data dari file SQL eksternal ke database.
convert_sqlite_to_mysql(str) : str	Mengonversi sintaks SQL SQLite ke sintaks SQL MySQL.
get_cv_id_from_path(Optional[str]) : Optional[str]	Mengekstrak 8 digit ID CV dari path file PDF.

#### 4.1.2.8. File: dump\_db.py

<b>Nama Fungsi</b>	<b>Kegunaan</b>
(tidak ada fungsi/def, script berjalan langsung)	Script berjalan langsung saat dijalankan (top-level code). Meng-dump seluruh database SQLite ke file SQL.

#### 4.1.2.9. File: seed\_db.py

Nama Fungsi	Kegunaan
seed_dummy_data() : None	Men-seed (mengisi) database dengan data dummy otomatis: mengambil file CV dari data/cv/, membuat profil pelamar dan aplikasi di database dengan data acak, serta memetakan file PDF ke profil secara round-robin.
__main__ (script block)	Memastikan fungsi seed_dummy_data() dieksekusi saat file dijalankan langsung (bukan sebagai modul).

#### 4.1.2.10. File: app.py

Nama Fungsi	Kegunaan
__init__(self) : None	Konstruktor kelas, inisialisasi database, data profil/CV, algoritma, dan komponen UI.
load_seed_data(self) : None	Memuat data awal pelamar dan aplikasi dari file SQL ke database melalui utilitas.
load_extracted_cv_data(self) : None	Memuat data hasil ekstraksi teks CV dari file CSV ke database melalui utilitas.

<b>Nama Fungsi</b>	<b>Kegunaan</b>
load_cvs_from_db(self) : list[dict]	Mengambil seluruh data aplikasi dari database (profil dan detail lamaran) lewat utilitas.
init_components(self) : None	Menginisialisasi seluruh komponen UI aplikasi melalui utilitas.
on_button_hover(self, e) : None	Handler efek hover pada tombol di UI (visual feedback), dipanggil dari event.
create_header(self) : ft.Row	Membuat dan mengembalikan header aplikasi (logo, judul, dsb) dengan Flet Row.
create_home_view(self) : ft.Column	Membuat tampilan halaman utama aplikasi (form pencarian, dsb) dengan Flet Column.
search_cv(self, e) : None	Proses pencarian CV utama: eksekusi pencocokan keyword, scoring, dan update hasil ke UI, dipanggil dari event.
update_summary_result_section(self, int, float, float, str) : None	Meng-update ringkasan hasil pencarian pada UI (jumlah hasil, waktu eksekusi, algoritma yang digunakan).
get_paginated_results(self) : list[dict]	Mengambil hasil pencarian CV yang sudah dipaginasi sesuai halaman aktif.
update_pagination(self, int) : None	Memperbarui tampilan pagination pada UI berdasarkan total hasil pencarian.

<b>Nama Fungsi</b>	<b>Kegunaan</b>
prev_page(self, e) : None	Navigasi ke halaman sebelumnya pada hasil pencarian (pagination), dipanggil dari event.
next_page(self, e) : None	Navigasi ke halaman berikutnya pada hasil pencarian (pagination), dipanggil dari event.
go_to_page(self, page_num) : None	Navigasi langsung ke halaman tertentu dalam pagination hasil pencarian.
update_results_display(self) : None	Memperbarui tampilan kartu hasil pencarian CV di UI, termasuk pagination.
create_cv_card(self, dict, int) : ft.Container	Membuat dan mengembalikan kartu CV (hasil pencarian) untuk ditampilkan di UI.
show_summary(self, dict) : None	Menampilkan ringkasan detail CV yang dipilih dalam tampilan summary di UI.
view_cv(self, dict) : None	Membuka file PDF CV asli dari path yang disimpan, biasanya dengan aplikasi eksternal.
go_to_home(self, e=None) : None	Navigasi kembali ke halaman utama dari tampilan lain, dipanggil dari event UI.
show_snackbar(self, str, str=ft.Colors.INDIGO_600) : None	Menampilkan notifikasi popup (snackbar) pada UI, parameter warna opsional.
close_dialog(self, e) : None	Menutup dialog pop-up summary CV di UI, dipanggil dari event.

<b>Nama Fungsi</b>	<b>Kegunaan</b>
main(self, ft.Page) : None	Fungsi utama yang mengatur layout, style, inisialisasi konten, dan menjalankan aplikasi Flet.
main_app(page: ft.Page) : None	Fungsi global, memulai aplikasi dengan membuat instance ATSFrontend dan menjalankan main().

#### 4.1.2.11. File: components.py

<b>Nama Fungsi</b>	<b>Kegunaan</b>
create_header_component(app) : ft.Container	Membuat komponen header aplikasi (logo, judul, subtitle, tombol home/back).
create_home_view_component(app) : ft.Container	Membuat komponen halaman utama pencarian CV (form input, algoritma, hasil, pagination).
create_summary_view_component(app) : ft.Container	Membuat kontainer tampilan summary, awalnya kosong dan diisi oleh hasil pencarian.
init_ui_components(app) : None	Menginisialisasi seluruh komponen UI ke objek app (input keyword, tombol, kontainer dsb).

<b>Nama Fungsi</b>	<b>Kegunaan</b>
create_cv_card_component(app, dict, int) : ft.Card	Membuat kartu hasil pencarian CV lengkap (nama, match, keyword, tombol summary/view CV).
create_summary_dialog_component(app, dict, dict) : ft.AlertDialog	Membuat dialog popup berisi ringkasan CV detail (nama, email, skill, pendidikan, dsb).
create_pdf_view_component(app, dict) : Optional[ft.Container]	Membuka file PDF CV, menampilkan pesan jika gagal (atau membuka dengan viewer eksternal).
create_summary_page_component(app, dict, dict) : ft.Container	Membuat tampilan summary penuh CV dalam satu halaman terstruktur (personal, skill, stats).

#### 4.1.2.12. File:event\_handlers.py

<b>Nama Fungsi</b>	<b>Kegunaan</b>
handle_search_cv(app, e) : None	Handler utama proses pencarian CV berdasarkan kata kunci, algoritma, dan jumlah top match, lalu update hasil ke UI.
perform_exact_search(app, str, list, str) : tuple	Melakukan pencarian exact matching pada teks dengan algoritma (KMP, BM, AC); mengembalikan hasil pencarian, total match, dll.

<b>Nama Fungsi</b>	<b>Kegunaan</b>
perform_fuzzy_search(app, str, list) : tuple	Melakukan pencarian fuzzy matching pada teks menggunakan Levenshtein; mengembalikan hasil pencarian fuzzy.
handle_button_hover(app, e) : None	Handler efek visual saat tombol di-hover pada UI aplikasi.
handle_go_to_home(app, e=None) : None	Handler untuk navigasi kembali ke halaman utama aplikasi.
handle_show_snackbar(app, str, str=ft.Colors.INDIGO_600) : None	Handler untuk menampilkan pesan snackbar (notifikasi) pada UI.
handle_prev_page(app, e) : None	Handler navigasi ke halaman sebelumnya pada hasil pencarian (pagination).
handle_next_page(app, e) : None	Handler navigasi ke halaman berikutnya pada hasil pencarian (pagination).
handle_go_to_page(app, int) : None	Handler navigasi langsung ke halaman tertentu pada hasil pencarian.
handle_show_summary(app, dict) : None	Handler untuk menampilkan tampilan ringkasan detail dari hasil CV yang dipilih.
handle_view_cv(app, dict) : None	Handler untuk membuka file PDF CV asli pada sistem pengguna.

<b>Nama Fungsi</b>	<b>Kegunaan</b>
handle_close_dialog(app, e) : None	Handler untuk menutup dialog pop-up pada halaman aplikasi.

#### 4.1.2.13. File: utils.py

<b>Nama Fungsi</b>	<b>Kegunaan</b>
load_seed_data_util(app) : None	Memuat data awal pelamar dan aplikasi dari file SQL ke database jika database masih kosong.
load_extracted_cv_data_util(app) : None	Memuat data hasil ekstraksi teks CV dari file CSV ke memori aplikasi.
load_cvs_from_db_util(app) : list[dict]	Mengambil seluruh data aplikasi dari database melalui method app.db.
update_summary_result_section_util(app, int, float, float, str) : None	Mengupdate tampilan ringkasan hasil pencarian di UI aplikasi (jumlah hasil, waktu proses, algoritma, dll).
get_paginated_results_util(app) : tuple	Mengembalikan hasil pencarian yang sudah dipaginasi untuk halaman aktif (list hasil & jumlah total hasil).
update_pagination_util(app, int) : None	Mengupdate tampilan pagination pada UI aplikasi sesuai total hasil pencarian.

<b>Nama Fungsi</b>	<b>Kegunaan</b>
update_results_display_util(app) : None	Mengupdate/mengatur tampilan kartu hasil pencarian CV di UI, menampilkan pesan jika hasil kosong.

#### 4.1.2.14. File: pdf\_extractor.py

<b>Nama Fungsi</b>	<b>Kegunaan</b>
extract_text_pypdf2(str) : str	Mengekstrak teks dari PDF menggunakan library PyPDF2.
extract_text_pymupdf(str) : str	Mengekstrak teks dari PDF menggunakan library PyMuPDF (fitz).
extract_text(str, str="auto") : str	Mengekstrak teks dari PDF dengan metode yang dipilih (auto, pypdf2, pymupdf), lalu dibersihkan.
clean_text(str) : str	Membersihkan teks hasil ekstraksi dari spasi, newline, dan karakter tidak perlu.
extract_text_by_pages(str) : list	Mengekstrak teks tiap halaman PDF dan mengembalikan list teks bersih per halaman.
get_pdf_info(str) : dict	Mengambil metadata file PDF (jumlah halaman, metadata, ukuran file, nama file).

<b>Nama Fungsi</b>	<b>Kegunaan</b>
is_valid_pdf(str) : bool	Mengecek apakah file PDF valid dan dapat dibaca.
extract_structured_data(str) : dict	Mengekstrak teks dan metadata PDF, lalu mengolahnya menjadi data terstruktur menggunakan RegexExtractor.

#### 4.1.2.15. File: regex\_extractor.py

<b>Nama Fungsi</b>	<b>Kegunaan</b>
__init__(self) : None	Konstruktor kelas, menginisialisasi seluruh pola regex untuk ekstraksi informasi dari CV.
extract_emails(str) : list	Mengekstrak semua alamat email valid dari teks.
extract_phone_numbers(str) : list	Mengekstrak semua nomor telepon/HP valid dari teks.
extract_skills(str) : list	Mengekstrak seluruh skills (keahlian) yang sesuai dengan daftar pattern.
extract_education(str) : list	Mengekstrak informasi pendidikan (institusi, gelar, tahun, dsb) dari teks CV.
extract_experience(str) : list	Mengekstrak pengalaman kerja, magang, jabatan, dan durasi dari teks.

Nama Fungsi	Kegunaan
extract_summary(str) : list	Mengekstrak ringkasan/summary dari bagian awal CV atau section summary/objective.
extract_dates(str) : list	Mengekstrak semua tanggal yang terdeteksi (tahun, bulan-tahun, range tahun, dsb) dari teks.
extract_names(str) : list	Mengekstrak nama kandidat (dari baris awal CV yang lolos validasi pola nama dan filter keyword).
extract_cv_info(str) : dict	Mengekstrak semua info penting dari CV sekaligus (nama, email, phone, skill, education, experience, summary, dates) ke dalam dictionary.

## 4.2. Penjelasan Tata Cara Penggunaan Program

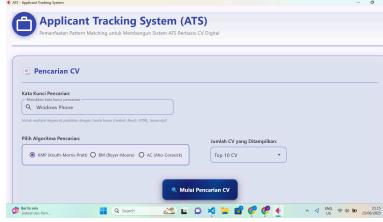
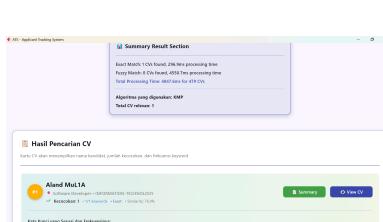
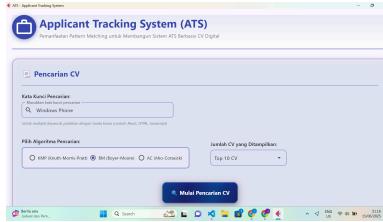
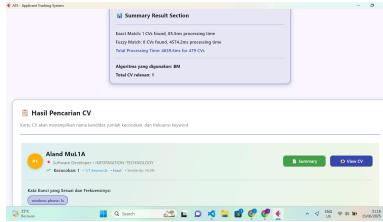
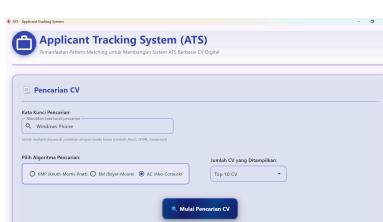
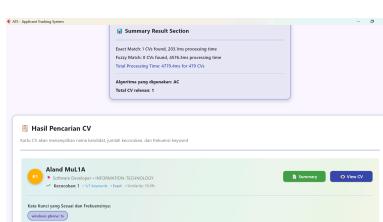
1. Jalankan perintah berikut di direktori utama proyek:

```
'''bash
python main.py
'''
```

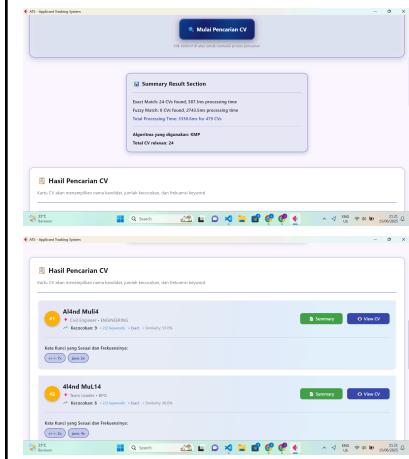
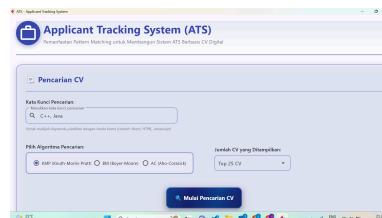
2. Aplikasi akan mengekstraksi teks dari PDF ke CSV, memasukkan database SQL dari data/tubes3\_seeding.sql ke \_host\_ database melalui MySQL, kemudian membuka antarmuka Flet.
3. Di UI, masukkan kata kunci, pilih algoritma (KMP, BM, Aho-Corasick), dan pilih maks hasil.
4. Klik \*\*Cari\*\* untuk mencari profil pelamar beserta CV yang mengandung kata kunci tersebut.

5. Hasil pencarian akan ditampilkan dalam tabel beserta waktu eksekusi dan jumlah kemunculan kata kunci pada setiap profil tersebut.

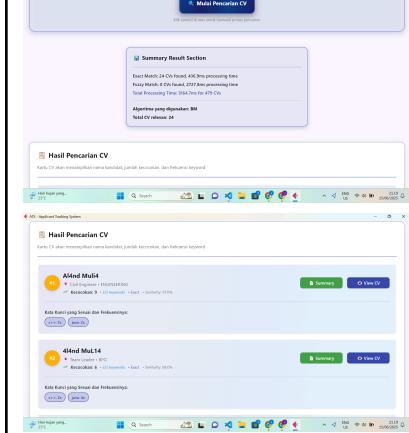
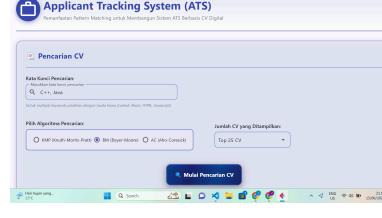
### 4.3. Hasil Pengujian Minimal 4 Pencarian dengan Variasi Keyword, Algoritma, dan Panjang Keyword yang Berbeda

Variasi	Input	Hasil
<b>Windows Phone; KMP; 1 Keyword</b>		
<b>Windows Phone; BM; 1 Keyword</b>		
<b>Windows Phone; Aho-Corasick; 1 Keyword</b>		

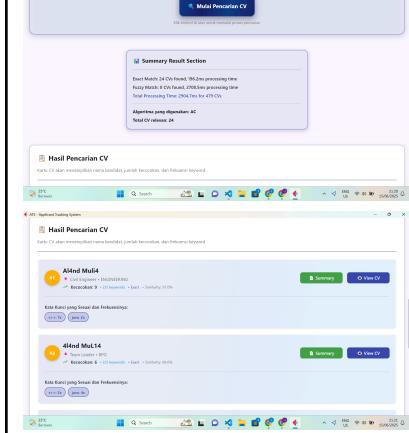
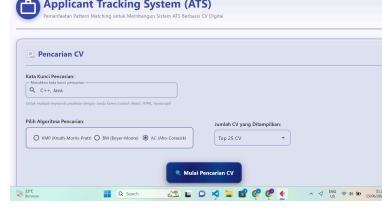
## C++, Java; KMP; 2 Keyword



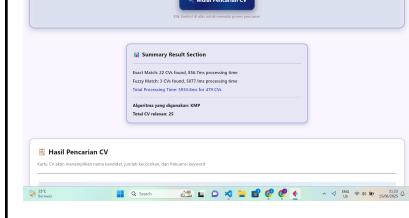
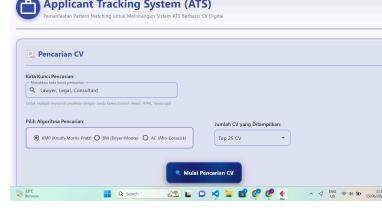
## C++, Java; BM; 2 Keyword

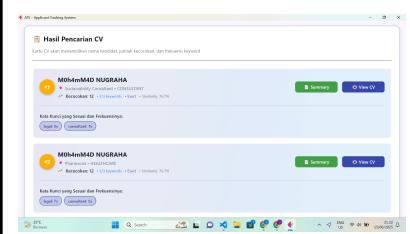
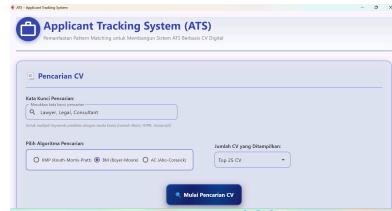
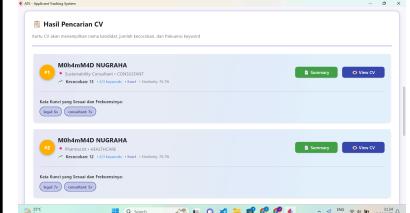
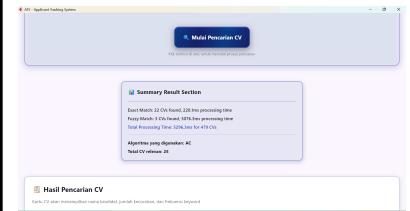
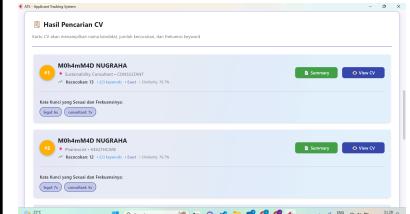
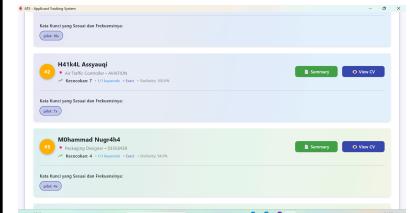


## C++, Java; Aho-Corasick; 2 Keyword



## Lawyer, Legal, Consultant; KMP; 3 Keyword



		
<b>Lawyer, Legal, Consultant; BM; 3 Keyword</b>		 
<b>Lawyer, Legal, Consultant; Aho-Corasick; 3 Keyword</b>		 
<b>Pilot; KMP; 1 Keyword</b>		 

## Business, Chef; BM; 2 Keyword

The screenshot shows the ATS application window. On the left, the search interface has 'Business, Chef' entered into the search bar. On the right, the results section displays two search results:

- Summary Result Section**:
  - Count: 10 CV found, 100ms processing time
  - Pattern Matched: 10 CV found, 100ms processing time
  - Total Processing Time: 100ms for 10 CV
  - Algo used: BM
  - Total CV release: 10
- Hasil Pencarian CV**:
  - Kata kunci yang dicari tidak ditemukan dalam hasil pencarian.
- Hasil Pencarian CV**:
  - Kata kunci yang dicari tidak ditemukan dalam hasil pencarian.
  - REHMAN ALHAKIM**
    - Berangkat: 10/08/2020
    - Keterkaitan: 24%
    - Konten: 100%
  - ICHWAN Alhakim**
    - Berangkat: 10/08/2020
    - Keterkaitan: 25%
    - Konten: 100%

## Business Development; Aho-Corasick; 1 Keyword

The screenshot shows the ATS application window. On the left, the search interface has 'Business Development' entered into the search bar. On the right, the results section displays two search results:

- Summary Result Section**:
  - Count: 25 CV found, 100ms processing time
  - Pattern Matched: 25 CV found, 100ms processing time
  - Total Processing Time: 100ms for 25 CV
  - Algo used: AC
  - Total CV release: 25
- Hasil Pencarian CV**:
  - Kata kunci yang dicari tidak ditemukan dalam hasil pencarian.
- Hasil Pencarian CV**:
  - Kata kunci yang dicari tidak ditemukan dalam hasil pencarian.
  - Astro Muli**
    - Berangkat: 10/08/2020 - BUSINESS DEVELOPMENT
    - Keterkaitan: 8%
    - Konten: 100%
  - H414KL\_Ayusqij**
    - Berangkat: 10/08/2020 - BUSINESS DEVELOPMENT
    - Keterkaitan: 8%
    - Konten: 100%

## 4.4. Analisis Hasil Pengujian

Berdasarkan hasil pengujian, aplikasi ini mampu menjalankan fungsi pencarian string dengan baik menggunakan tiga pendekatan berbeda, yakni Knuth-Morris-Pratt (KMP), Boyer-Moore, dan Aho-Corasick pada berbagai kondisi. Kami menguji ketiganya dalam beberapa skenario: pertama, pencarian pola tunggal pada teks pendek; kedua, pola tunggal pada teks yang sangat panjang; dan ketiga, pencarian berskala besar dengan banyak pola sekaligus. Setiap algoritma menunjukkan performa masing-masing, namun secara umum Aho-Corasick keluar sebagai yang tercepat.

Dalam skenario pencarian *multi-pattern*, Aho-Corasick lebih unggul berkat pemanfaatan struktur automaton yang memproses teks dalam satu kali lintasan, sehingga kompleksitasnya linear terhadap jumlah karakter dan pola. Ketika dihadapkan pada lusinan atau bahkan ratusan pola sekaligus, waktu eksekusinya tetap stabil dan jauh lebih singkat dibandingkan dua algoritma

lainnya. Sementara itu, Boyer-Moore, meski didesain terutama untuk satu pola, masih menunjukkan kecepatan yang cukup tinggi pada teks panjang sekali pun, berkat heuristik “*bad character*” dan “*good suffix*” yang mengurangi jumlah pergeseran *pattern* dan perbandingan karakter. Namun, begitu dihadapkan pada banyak pola sekaligus, Boyer-Moore mulai tertinggal dari Aho-Corasick dalam hal efisiensi karena algoritma tersebut harus mengulangi proses heuristik untuk setiap pola secara terpisah.

Di sisi lain, KMP, walaupun implementasinya paling sederhana dengan memanfaatkan tabel *prefix-suffix* (*LSP table*) untuk menghindari pencocokan ulang, cenderung paling lambat di antara ketiganya dalam sebagian besar kasus. Algoritma ini tetap efisien untuk pencarian satu pola pada teks berukuran sedang, tetapi lebih lambat jika teksnya sangat panjang atau pola yang dicari berjumlah banyak.

## **BAB 5 KESIMPULAN, SARAN, DAN REFLEKSI TENTANG TUGAS BESAR 3**

## 5.1. Kesimpulan

Pada pelajaran Tugas Besar 3 IF2211 Strategi Algoritma, kelompok kami berhasil mengembangkan sistem Applicant Tracking System (ATS) yang dapat secara efisien dan efektif mencocokkan CV pelamar dengan kriteria-kriteria yang dibutuhkan pekerjaan dengan menggunakan algoritma pattern matching. Pada pembuatan program, kami berhasil mengimplementasikan tiga algoritma, Knuth-Morris-Pratt, Boyer-Moore, serta Aho-Corasick. Algoritma Knuth-Morris-Pratt menawarkan pencocokan eksak dengan kompleksitas linier melalui pemanfaatan *border function*, sementara Boyer-Moore memungkinkan pencarian yang sangat cepat pada teks panjang dengan memanfaatkan heuristik *last occurrence* dan *good suffix*. Di sisi lain, Aho-Corasick memungkinkan pencocokan banyak kata kunci sekaligus secara optimal melalui struktur automata berbasis *trie*. Untuk menangani perbedaan penulisan dan kesalahan ketik dalam kata kunci, kami juga mengintegrasikan sistem *fuzzy matching* menggunakan algoritma Levenshtein Distance. Hal ini bertujuan untuk menangani variasi input yang mungkin diberikan oleh pengguna.

Secara keseluruhan, pelajaran tugas besar ini telah memberikan pengalaman dan pemahaman yang lebih mendalam mengenai penerapan algoritma *string matching* dalam konteks nyata, khususnya dalam penggunaan algoritma Knuth-Morris-Pratt,, Boyer-Moore, dan Aho-Corasick. Kami juga mendapat wawasan lebih dalam mengenai pentingnya pemilihan algoritma yang tepat sesuai dengan karakteristik data dan kebutuhan sistem, serta bagaimana cara untuk dapat mengintegrasikan multiple *pattern matching algorithms* untuk menciptakan sistem yang efektif dan efisien.

## 5.2. Saran

Berdasarkan pengalaman yang kami dapat selama pelajaran tugas besar ini, kami memiliki beberapa saran yang dapat menjadi bahan perbaikan dan pengembangan di masa mendatang, khususnya dalam hal peningkatan *user experience* dan efektivitas sistem.

Dari sisi desain antarmuka, pengalaman pengguna dapat lebih ditingkatkan dengan menambahkan berbagai fitur tambahan, seperti *advanced filtering*, *customizable dashboard*, serta grafik visualisasi frekuensi kemunculan kata kunci. Implementasi fitur-fitur tersebut diharapkan akan membuat sistem menjadi lebih interaktif, serta mempermudah pengguna dalam memahami hasil pencocokan dan mengelola hasil pencarian dengan lebih baik.

Selain itu, sistem juga dapat dikembangkan kembali dengan menambahkan fitur pengelompokan otomatis CV berdasarkan kategori pekerjaan atau bidang keahlian. Hal ini akan membantu pengguna dalam menavigasi dan membandingkan pelamar yang memiliki latar belakang serupa, serta meningkatkan efisiensi proses seleksi secara keseluruhan.

### **5.3. Refleksi**

Pengerjaan tugas besar ini telah memberikan kami banyak pengetahuan baru dan pembelajaran berharga, terutama dalam memahami dan menerapkan konsep integrasi berbagai algoritma *pattern matching* dalam konteks dunia nyata, khususnya untuk membangun sistem *Applicant Tracking System* (ATS) berbasis pencocokan isi CV. Melalui pengalaman mengimplementasikan berbagai algoritma, seperti Knuth-Morris-Pratt, Boyer-Moore, dan Aho-Corasick, kami memperoleh pemahaman yang lebih dalam mengenai karakteristik, efisiensi, serta kelebihan dan keterbatasan masing-masing algoritma dalam menangani proses pencarian string. Penggunaan algoritma Levenshtein Distance untuk *fuzzy matching* juga memperluas wawasan kami mengenai pentingnya penanganan variasi input secara adaptif. Selain itu, kami mendapatkan pengalaman praktis dalam membangun sistem secara menyeluruh, mulai dari ekstraksi data menggunakan *regular expression*, pengolahan file PDF, hingga perancangan antarmuka pengguna yang interaktif dan fungsional. Secara keseluruhan tugas besar ini mengajarkan kami tentang pentingnya kolaborasi antara perancangan algoritma, pemrosesan data, dan aspek desain sistem dalam menciptakan solusi yang efisien dan sesuai kebutuhan pengguna.

## LAMPIRAN

**Tautan Repository GitHub:**

[https://github.com/jovan196/Tubes3\\_PejuangCV.git](https://github.com/jovan196/Tubes3_PejuangCV.git)

**Tautan Video YouTube:**

<https://youtube.com/@nataliadesiany9127?si=5-Q7xcQhWEcse4Id>

**Tabel Pengecekan Fitur:**

No	Poin	Ya	Tidak
1	Aplikasi dapat dijalankan.	✓	
2	Aplikasi menggunakan basis data berbasis SQL dan berjalan dengan lancar.	✓	
3	Aplikasi dapat mengekstrak informasi penting menggunakan Regular Expression (Regex).	✓	
4	Algoritma <i>Knuth-Morris-Pratt (KMP)</i> dan <i>Boyer-Moore (BM)</i> dapat menemukan kata kunci dengan benar.	✓	
5	Algoritma Levenshtein Distance dapat mengukur kemiripan kata kunci dengan benar.	✓	
6	Aplikasi dapat menampilkan <i>summary CV applicant</i> .	✓	

7	Aplikasi dapat menampilkan CV <i>applicant</i> secara keseluruhan.	✓	
8	Membuat laporan sesuai dengan spesifikasi.	✓	
9	Membuat bonus enkripsi data profil <i>applicant</i> .		✓
10	Membuat bonus algoritma Aho-Corasick.	✓	
11	Membuat bonus video dan diunggah pada Youtube.	✓	



## DAFTAR PUSTAKA

- Munir, R. (2025). *Pencocokan string (string matching) dengan algoritma brute force, KMP, Boyer-Moore.* Diakses pada 13 Juni 2025, dari [https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/23-Pencocokan-string-\(2025\).pdf](https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/23-Pencocokan-string-(2025).pdf)
- Munir, R. (2025). *Pencocokan string dengan regular expression (regex).* Diakses pada 14 Juni 2025, dari [https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/24-String-Matching-dengan-Regex-\(2025\).pdf](https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/24-String-Matching-dengan-Regex-(2025).pdf)
- GeeksforGeeks. (2024). *Aho-Corasick Algorithm – Pattern Searching.* Diakses pada 15 Juni 2025, dari <https://www.geeksforgeeks.org/dsa/aho-corasick-algorithm-pattern-searching/>