Laporan Tugas Besar 3
IF2211 Strategi Algoritma 2025/2026

Disusun oleh:

Kelompok 21 - DaveBeban

Dave Daniell Yanni	(13523003)
Varel Tiara	(13523008)
Shanice Feodora Tjahjono	(13523097)

PROGRAM STUDI TEKNIK INFORMATIKA

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA

INSTITUT TEKNOLOGI BANDUNG

JL. GANESA 10, BANDUNG 40132

2025

KATA PENGANTAR

Puji syukur kami panjatkan ke hadirat Tuhan Yang Maha Esa, yang atas berkat rahmat-Nya, kami berhasil merampungkan laporan Tugas Besar 3 untuk mata kuliah IF2211 Strategi Algoritma. Laporan ini menjadi dokumentasi dari perancangan dan implementasi sistem Applicant Tracking System (ATS) Berbasis CV Digital dengan memanfaatkan teknik Pattern Matching.

Tugas besar ini lahir dari sebuah tantangan nyata di era digital, yaitu kebutuhan akan proses rekrutmen yang cepat dan akurat di tengah tumpukan data yang masif. Kami menjawab tantangan ini dengan membangun sebuah sistem yang mampu membedah dokumen CV digital, khususnya dalam format PDF yang tidak terstruktur untuk menemukan informasi krusial. Inti dari sistem kami adalah implementasi algoritma pencocokan string, yaitu Knuth-Morris-Pratt (KMP), Boyer-Moore (BM), dan Aho-Corasick, untuk melakukan pencocokan presisi (exact match). Lebih dari itu, untuk mengantisipasi ketidaksempurnaan data seperti kesalahan pengetikan (typo), kami juga mengintegrasikan metode fuzzy matching menggunakan algoritma Levenshtein Distance.

Pengerjaan tugas besar ini menjadi sebuah perjalanan yang tidak hanya menguji efisiensi algoritma dalam isolasi, tetapi juga menavigasi kompleksitas integrasinya dengan basis data dan perancangan antarmuka pengguna yang efektif. Pengalaman ini secara fundamental memperkuat pemahaman kami tentang bagaimana strategi algoritma menjadi tulang punggung dari solusi perangkat lunak yang cerdas dan efisien.

Kami mengucapkan terima kasih yang sebesar-besarnya kepada dosen pengampu dan para asisten mata kuliah IF2211 yang telah membimbing kami dengan sabar. Semoga laporan ini tidak hanya memenuhi kewajiban akademis, tetapi juga dapat menjadi inspirasi dan referensi berharga mengenai aplikasi praktis dari strategi algoritma.

Bandung, 14 Juni 2025

Dave Daniell Yanni	(13523003)
Varel Tiara	(13523008)
Shanice Feodora Tjahjono	(13523097)

DAFTAR ISI

KATA PENGANTAR **2**
DAFTAR ISI **3**
DAFTAR GAMBAR **6**
BAB I **7**
DESKRIPSI TUGAS **7**
Penjelasan Implementasi 8
BAB II **12**
LANDASAN TEORI **12**
2.1 Pencocokan String (String Matching) 12
2.2 Algoritma Knuth-Morris-Pratt (KMP) 13
2.2.1 Fungsi Pinggiran (Border Function) 15
2.2.2 Cara Kerja Algoritma dan Kompleksitas 15
2.3 Algoritma Booyer-Moore (BM) 16
2.3.1 Teknik Pencocokan Mundur (Looking-Glass Technique) 16
2.3.2 Teknik Lompatan Karakter (Heuristik Karakter Buruk) 17
2.3.3 Last Occurrence Function (L(x)) 17
2.3.4 Aturan Pergeseran Berdasarkan Karakter Buruk (Bad Character) 17
2.4 Algoritma Aho-Corasick 20
2.4.1 Struktur Data Inti: Automaton Aho-Corasick 20
2.4.2 Mesin Penggerak: Tautan Sukses dan Kegagalan 20
2.4.3 Fase Algoritma dan Kompleksitas 21
2.5 Penjelasan Aplikasi Desktop yang Dibangun 22
2.5.1 Arsitektur dan Alur Kerja Sistem 22
2.5.2 Mekanisme Pencarian dan Penampilan Top CV 23
2.5.3 Antarmuka Pengguna dan Fungsionalitas 23
BAB III **25**
ANALISIS PEMECAHAN MASALAH **25**
3.1 Langkah-Langkah Pemecahan Masalah 25
3.1.1 Pengolahan dan Pra-pemrosesan Data CV 25
3.1.3.1 Ekstraksi Teks dari PDF 25
3.1.2 Implementasi Algoritma String Matching 25
3.1.2.1 Pencocokan Pola Tunggal: KMP dan Boyer-Moore 25
3.1.2.2 Pencocokan Multi-Pola: Aho-Corasick 26
3.1.2.4 Perancangan Layer Persistensi dan Keamanan Data 27
3.1.3 Pengembangan Sistem Peringkat dan Penilaian (Scoring & Ranking) 27

3.1.4.1 Penilaian Berbasis Frekuensi 27
3.1.4.2 Pengukuran Waktu Pencarian 27
3.1.4 Perancangan Antarmuka Pengguna yang Interaktif 28
3.1.4.1 Arsitektur Antarmuka Berbasis Komponen 28
3.1.4.2 Umpan Balik Proses Pencarian secara Real-Time 28
3.1.4.3 Visualisasi Hasil yang Interaktif 28
3.1.5 Pengujian Performa dan Optimalisasi Sistem 28
3.1.4.1 Benchmarking Kinerja Algoritma 28
3.1.4.2 Integrasi Analitik untuk Pemantauan 29
3.2 Proses pemetaan masalah menjadi elemen-elemen algoritma KMP dan BM. 29
3.2.1 Algoritma Knuth-Morris-Pratt (KMP) 29
3.2.1.1 Pra-pemrosesan Pola dengan Fungsi Pinggiran (Border Function) 29
3.2.1.2 Strategi Pencarian Linear Tanpa Mundur (Backtracking) 29
3.2.2 Algoritma Boyer-Moore (BM) 30
3.2.3 Adaptasi Algoritma untuk Domain Perekrutan (CV/Resume) 30
3.2.1.3 Penanganan Case-Insensitivity melalui Normalisasi Teks 30
3.2.1.4 Kontekstualisasi Hasil Pencarian untuk Analisis 31
3.3 Fitur Fungsional dan Arsitektur Aplikasi Desktop yang Dibangun 31
3.3.1 Fitur Fungsional Utama 31
3.3.1.1 Manajemen Basis Data dan Statistik 31
3.3.1.2 Implementasi dan Pemilihan Algoritma Pencarian 31
3.3.1.3 Pencarian CV Berbasis Kata Kunci 32
3.3.1.4 Ekstraksi dan Parsing Konten CV 32
3.3.1.5 Tampilan Ringkasan dan Detail CV 32
3.3.1.6 Enkripsi Data 32
3.3.2 Arsitektur Aplikasi Desktop 33
3.3.2.1 Kerangka 33
3.3.2.2 Struktur Proyek yang Modular 33
3.3.2.3 Arsitektur Berlapis (Layered Architecture) 34
3.3.2.4 Pola Desain Perangkat Lunak (Design Patterns) 34
3.3.2.5 Alur Pengalaman Pengguna (User Experience Flow) 35
3.3.2.6 Fitur Performa dan Skalabilitas 35
3.4 Ilustrasi Kasus 35
BAB IV 40
IMPLEMENTASI DAN PENGUJIAN 40
4.1 Spesifikasi Teknis Program 40
4.1.1 Struktur Data 40
4.1.2 Fungsi 45

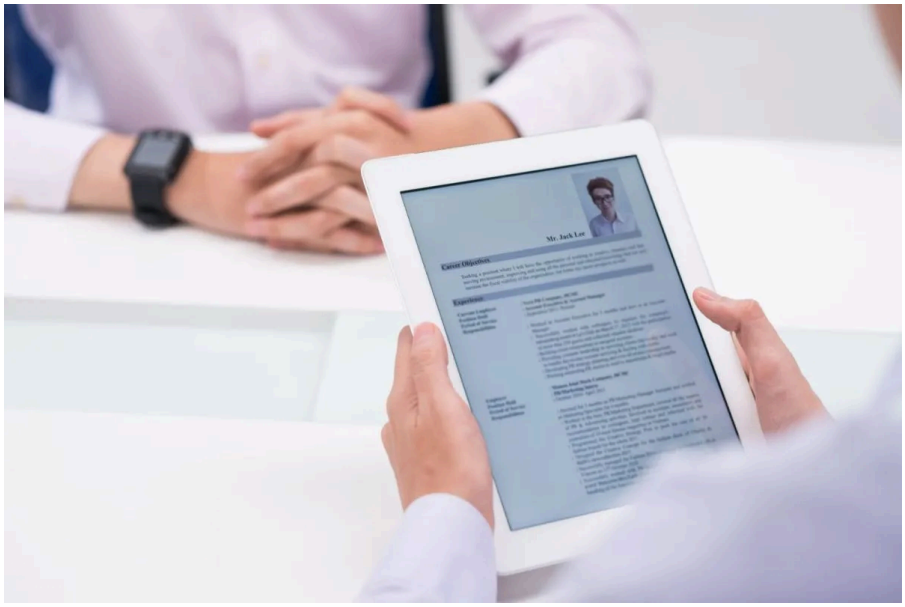
4.1.3 Prosedur 54
4.2 Penjelasan Tata Cara Penggunaan Program (Interface Program, Fitur-fitur yang disediakan program, dan sebagainya) 58
4.2.1 Interface Program, Fitur-fitur yang Disediakan Program 58
4.3 Hasil Pengujian 59
4.4 Analisis Hasil Pengujian 64
4.4.1 KMP 65
4.4.2 Boyer-Moore 65
4.4.3 Aho-Corasick 65
4.4.4 Analisis Perbandingan 66
BAB V **68**
KESIMPULAN DAN SARAN **68**
5.1 Kesimpulan 68
5.2 Saran 68
5.3 Refleksi 69
BAB VI **70**
LAMPIRAN **70**
6.1 Daftar Pustaka 70
6.2 Tautan Repository 70
6.3 Tautan Video 70
6.4 Tabel Checklist 70

DAFTAR GAMBAR

Gambar 1.1. CV ATS dalam Dunia Kerja.....7
Gambar 1.2. Skema Implementasi Applicant Tracking System..... 8
Gambar 1.3. Skema basis data CV ATS.....10
Gambar 2.1. Visualisasi Algoritma KMP..... 14
Gambar 2.2. Visualisasi Last Occurrence Function..... 17
Gambar 2.3. Case 1..... 18
Gambar 2.4. Case 2..... 18
Gambar 2.5. Case 3..... 19
Gambar 3.1. Folder Struct.....34
Gambar 3.2. Input Pencarian “SQL”.....36
Gambar 3.3. Hasil Pencarian untuk Kata Kunci "SQL"..... 37
Gambar 3.4. Input Pencarian "Develap"..... 38
Gambar 3.5. Hasil Pencarian untuk Kata Kunci "Develap".....38
Gambar 3.6. CV Summary.....39

BAB I

DESKRIPSI TUGAS



Gambar 1.1. 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

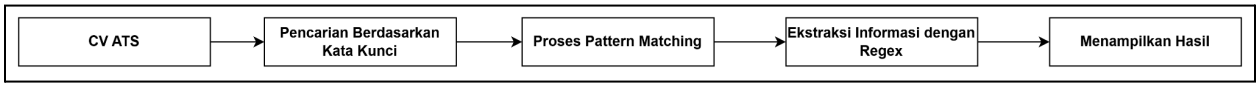
identitas, pengalaman kerja, keahlian, dan riwayat pendidikan secara efisien. Pattern matching menjadi solusi ideal dalam menghadapi tantangan ini.

Pattern matching adalah teknik untuk menemukan dan mencocokkan pola tertentu dalam teks. Dalam konteks ini, algoritma Boyer-Moore, Knuth-Morris-Pratt (KMP), dan Aho-Corasick 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, Kami 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, Knuth-Morris-Pratt, Aho-Corasick sebagai bonus. 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, Kami 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 sebagai bonus) untuk menganalisis dan mencocokkan pola dalam dokumen CV digital, sesuai dengan konsep yang telah dipelajari dalam materi dan slide perkuliahan.



Gambar 1.2. Skema Implementasi Applicant Tracking System
(sumber: Spesifikasi Tugas Besar 3 Stima 2024/2025)

Sistem ini bertujuan untuk mencocokkan kata kunci dari user terhadap isi CV pelamar kerja dengan pendekatan pattern matching menggunakan algoritma KMP (Knuth-Morris-Pratt), BM (Boyer-Moore), atau Aho-Corasick. 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",

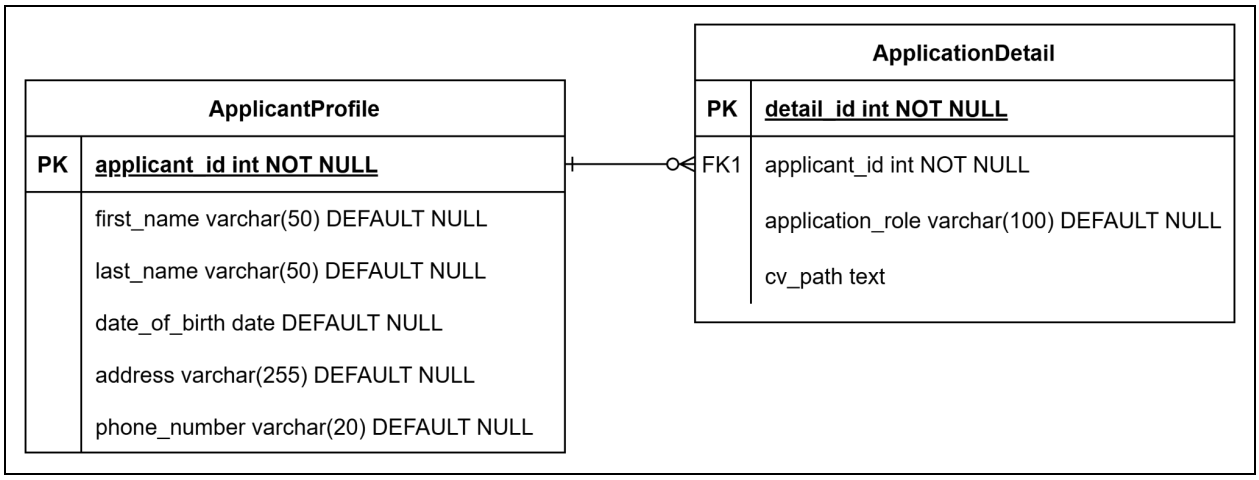
"react", dan "sql)" serta jumlah CV yang ingin ditampilkan (misalnya Top 10 matches). Setiap file CV dalam format PDF akan dikonversi menjadi satu string panjang yang memuat seluruh teks dari dokumen tersebut. Proses konversi ini bertujuan untuk mempermudah pencocokan pola menggunakan algoritma string 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.

CV ATS	**Ekstraksi Text untuk Regex**	**Ekstraksi Text untuk *Pattern Matching* (KMP & BM)**
10276858.pdf	Ekstraksi Text Regex.txt	Ekstraksi Text Pattern Matching.txt

Tabel 1. Hasil ekstraksi teks dari CV ATS

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), Boyer-Moore (BM), dan Aho-Corasick 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. Kami 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 1.3. Skema basis data CV ATS

(sumber: Spesifikasi Tugas Besar 3 Stima 2024/2025)

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.

Kami juga 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 II
LANDASAN TEORI

2.1 Pencocokan String (String Matching)

Pencocokan string (string matching) adalah salah-satu masalah fundamental yang bertujuan untuk menemukan keberadaan suatu string pendek (pola) di dalam sebuah string yang lebih panjang (teks). Proses ini menjadi inti dari banyak aplikasi, mulai dari fitur "search" pada editor teks hingga analisis bioinformatika yang kompleks.

Secara formal, masalah ini dapat didefinisikan sebagai berikut: Diberikan sebuah Teks (T), yaitu sebuah string dengan panjang n karakter, dan sebuah Pola (P), yaitu string yang akan dicari dengan panjang m karakter. Dalam sebagian besar aplikasi di kehidupan nyata, dapat diasumsikan bahwa panjang pola jauh lebih kecil daripada panjang teks ($m \ll n$). Lalu, Tujuan utama dari algoritma pencocokan string adalah untuk menemukan lokasi atau indeks awal dari kemunculan pertama Pola P di dalam Teks T.

Sebagai contoh, perhatikan kasus berikut:

T: the rain in spain stays mainly on the plain
P: main

Pada Teks T di atas, algoritma pencocokan string akan berhasil menemukan Pola P ("main") sebagai bagian dari kata "mainly". Sehingga tugas algoritma ini adalah melaporkan posisi awal di mana kecocokan ini terjadi.

Untuk memahami cara kerja dari algoritma pencocokan string yang efisien, penting untuk terlebih dahulu memahami konsep dasar struktur string, yaitu prefix dan suffix. Kedua konsep ini menjadi dasar dalam pra-pemrosesan pola pada algoritma string matching untuk mengoptimalkan proses pencarian.

Misalkan kita memiliki sebuah string S dengan panjang m karakter, yang direpresentasikan sebagai $S = x_0x_1...x_{\square-1}$.

- Prefix: Sebuah prefix dari string S adalah substring yang dimulai dari karakter pertama S. Dengan kata lain, sebuah prefix adalah S[0..k] di mana $0 \leq k < m$.
- Suffix: Sebaliknya, sebuah suffix dari string S adalah substring yang berakhir pada karakter terakhir S. Ini dapat direpresentasikan sebagai S[k..m-1] di mana $0 \leq k < m$.

Untuk memberikan gambaran yang lebih jelas, mari kita ambil string S = "andrew" sebagai contoh:

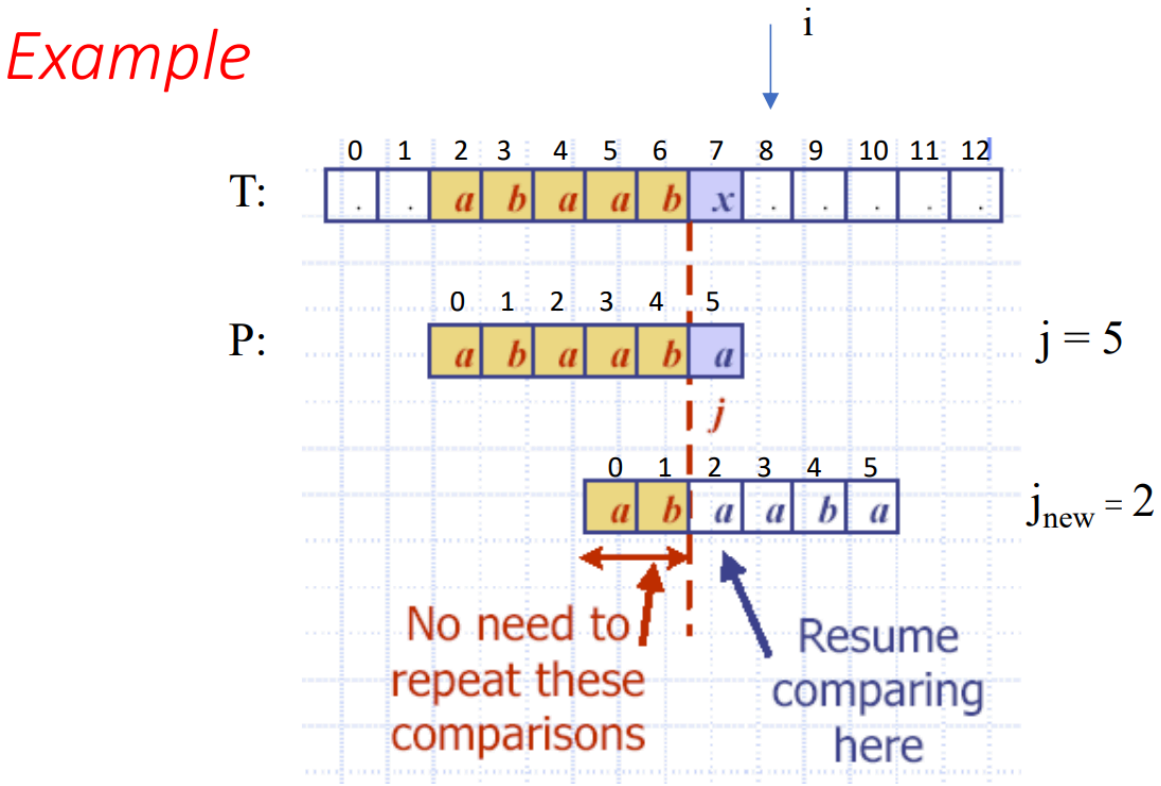
- Seluruh prefix dari "andrew" adalah: "a", "an", "and", "andr", "andre", dan "andrew".
- Seluruh suffix dari "andrew" adalah: "w", "ew", "rew", "drew", "ndrew", dan "andrew".

Kemudian, string itu sendiri selalu dianggap sebagai prefix sekaligus suffix dari dirinya sendiri. Pemahaman mendalam tentang prefix dan suffix ini sangat krusial karena digunakan untuk menghitung pergeseran pola yang “cerdas” selama proses pencarian, sehingga menghindari perbandingan karakter yang berulang dan tidak perlu.

2.2 Algoritma Knuth-Morris-Pratt (KMP)

Algoritma Knuth-Morris-Pratt (KMP) adalah salah satu algoritma pencocokan string yang lebih efisien dibandingkan pendekatan dengan algoritma brute force. Algoritma ini seperti brute-force karena KMP melakukan pemindaian teks T dari kiri ke kanan untuk mencari pola P pada teks T. Namun, KMP lebih unggul dalam menangani ketidakcocokan (mismatch) karena alih-alih menggeser pola hanya satu karakter ke kanan, KMP menggunakan pergeseran yang lebih "cerdas" berdasarkan analisis internal dari pola itu sendiri.

Keunggulan dari KMP adalah kemampuannya untuk menghindari perbandingan karakter yang redundan. Ketika terjadi ketidakcocokan antara karakter teks T[i] dan karakter pola P[j] (itu berarti segmen pola sebelumnya), yaitu P[0...j-1], yang telah berhasil cocok dengan segmen teks yang sesuai, maka inilah yang dimanfaatkan untuk menentukan seberapa jauh pola dapat digeser ke kanan tanpa kehilangan potensi kecocokan tersebut.



Gambar 2.1. Visualisasi Algoritma KMP

(sumber: Pencocokan String (String/Pattern Matching))

Pertanyaan utamanya adalah: Berapa pergeseran maksimal yang aman? Jawabannya ditemukan dengan mencari prefix terpanjang dari segmen yang cocok (P[0...j-1]) yang juga merupakan suffix dari segmen tersebut. Panjang dari prefix yang tumpang tindih ini memberitahu kita posisi baru di pola (j) untuk melanjutkan perbandingan, sehingga secara efektif "melompati" perbandingan yang sudah pasti berhasil.

Sebagai contoh, anggap pola P = "ababa" cocok dengan teks hingga karakter kelima, lalu gagal. Segmen yang cocok adalah P[0..4] = "ababa". Untuk menemukan pergeseran, kita mencari prefix terpanjang dari "ababa" yang juga merupakan suffix-nya, yaitu "aba" dengan panjang 3. Ini berarti kita dapat menggeser pola sejauh panjang_pola - panjang_prefix_suffix = 5 - 3 = 2 posisi, dan melanjutkan perbandingan dari indeks j=3 pada pola.

2.2.1 Fungsi Pinggiran (Border Function)

Untuk mengimplementasikan logika pergeseran “cerdas” ini secara sistematis, KMP melakukan pra-pemrosesan pada pola P untuk membuat sebuah struktur data bantu yang disebut Fungsi Pinggiran (Border Function). Fungsi ini sering juga disebut Failure Function dan biasanya direpresentasikan sebagai sebuah array, kita sebut saja b.

Fungsi b[k] didefinisikan sebagai panjang dari prefix proper terpanjang dari P[0...k] yang sekaligus merupakan suffix dari P[0...k]. "Prefix proper" berarti prefix yang tidak sama dengan string itu sendiri. Nilai b[k] ini secara esensial menyimpan informasi tentang posisi "aman" berikutnya untuk kembali jika terjadi ketidakcocokan pada posisi k+1.

Sebagai contoh, untuk pola P = "abababababca", tabel fungsi pinggirannya adalah sebagai berikut:

j	0	1	2	3	4	5	6	7	8	9
P[j]	a	b	a	b	a	b	a	b	c	a
k	-	0	1	2	3	4	5	6	7	8
b[k]	0	0	1	2	3	4	5	6	0	1

Tabel 2. Fungsi Pinggiran

Pada tabel di atas, misalnya b[7] = 6 karena untuk substring P[0..7] = "abababab", prefix proper terpanjang yang juga merupakan suffix adalah "ababab", yang memiliki panjang 6.

2.2.2 Cara Kerja Algoritma dan Kompleksitas

Algoritma KMP berjalan dalam dua fase:

1. Fase Pra-pemrosesan: Menghitung tabel fungsi pinggiran (b) untuk pola P. Fase ini memiliki kompleksitas waktu O(m), di mana m adalah panjang pola.
2. Fase Pencarian: Memindai teks T dari kiri ke kanan. Jika terjadi ketidakcocokan pada P[j], pointer pada pola j diperbarui menjadi b[j-1], tanpa pernah

menggerakkan pointer pada teks (i) ke belakang. Fase ini memiliki kompleksitas waktu O(n), di mana n adalah panjang teks.

Dengan demikian, kompleksitas waktu total dari Algoritma KMP adalah O(m + n) dengan menghitung fungsi pinggiran sebesar O(m) dan pencarian string sebesar O(n), yang menjadikannya sangat cepat dan efisien.

Kelebihan dari algoritma Knuth-Morris-Pratt adalah pointer pada teks T tidak pernah bergerak mundur. Ini membuat KMP sangat ideal untuk memproses data dari sumber yang bersifat stream, seperti file besar atau aliran data jaringan, di mana pembacaan ulang (mundur) sangat mahal atau tidak mungkin dilakukan.

Keunggulan KMP ini juga kurang terasa signifikan ketika ukuran alfabet (jumlah kemungkinan karakter) sangat besar. Pada kasus seperti itu, ketidakcocokan cenderung terjadi di awal pola, sehingga manfaat dari pergeseran cerdas menjadi minimal.

2.3 Algoritma Booyer-Moore (BM)

Algoritma Boyer-Moore adalah salah satu metode pencocokan string yang lebih efisien. Kecepatan dari algoritma ini bersumber pada dua pendekatan utama, yaitu teknik pencocokan mundur (looking-glass technique) dan teknik lompatan karakter (character-jump technique).

2.3.1 Teknik Pencocokan Mundur (Looking-Glass Technique)

Berbeda dengan algoritma brute-force atau KMP yang membandingkan karakter dari kiri ke kanan, Boyer-Moore melakukan pendekatan sebaliknya. Setelah menyelaraskan pola P dengan segmen teks T, perbandingan dimulai dari karakter terakhir pola bergerak ke kiri (mundur).

Pendekatan ini sangat baik karena jika ketidakcocokan (mismatch) ditemukan di awal perbandingan (misalnya pada karakter terakhir pola), algoritma dapat memperoleh informasi yang cukup untuk menggeser pola dengan lompatan yang sangat jauh ke kanan, melewati banyak karakter teks tanpa perlu membandingkannya sama sekali.

2.3.2 Teknik Lompatan Karakter (Heuristik Karakter Buruk)

Ini adalah mekanisme inti yang membuat Boyer-Moore begitu cepat. Ketika terjadi ketidakcocokan antara karakter teks T[i] = x dengan karakter pola P[j], algoritma memanfaatkan informasi dari karakter di teks yang menyebabkan kegagalan (x), yang disebut sebagai "karakter buruk" (bad character).

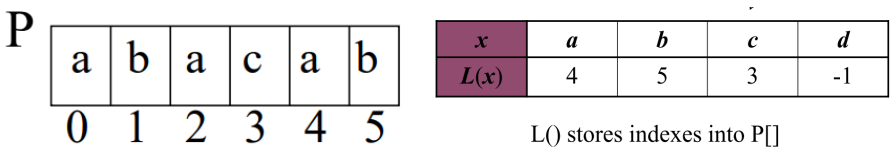
Untuk mengimplementasikan ini, Boyer-Moore melakukan pra-pemrosesan untuk membangun Fungsi Kemunculan Terakhir (Last Occurrence Function), yang kita sebut L(x).

2.3.3 Last Occurrence Function (L(x))

Fungsi L(x) memetakan setiap karakter x dari alfabet yang digunakan ke sebuah bilangan bulat. Nilainya didefinisikan sebagai:

- Indeks terbesar (paling kanan) k di mana P[k] == x.
- -1 jika karakter x tidak ada di dalam pola P.

Fungsi ini dihitung pada fase pra-pemrosesan dan biasanya disimpan dalam sebuah tabel atau array untuk akses cepat. Sebagai contoh, untuk alfabet A = {a, b, c, d} dan pola P = "abacab":



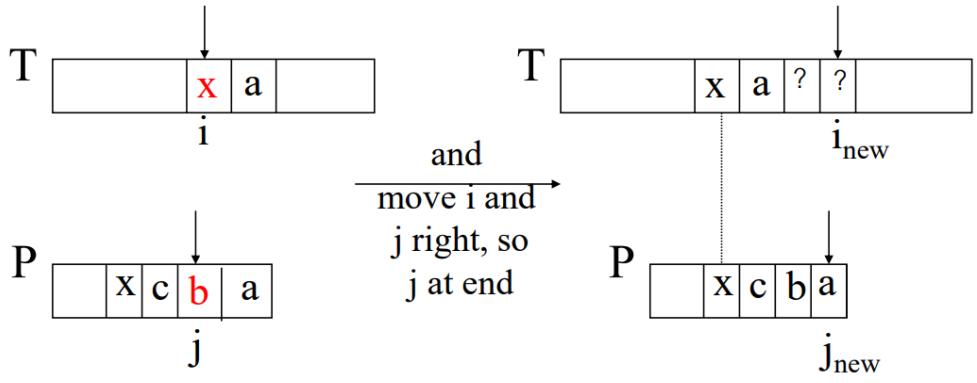
Gambar 2.2. Visualisasi Last Occurrence Function
(sumber: [Pencocokan String (String/Pattern Matching)])

Pada tabel ini, L(a) = 4 karena 'a' terakhir dalam "abacab" berada di indeks 4. L(d) = -1 karena 'd' tidak ada dalam pola.

2.3.4 Aturan Pergeseran Berdasarkan Karakter Buruk (Bad Character)

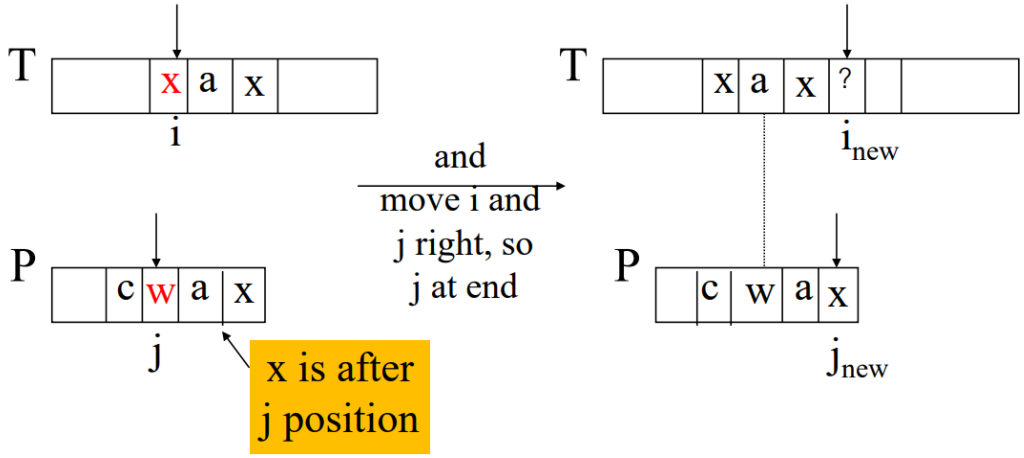
Berdasarkan fungsi L, Boyer-Moore menerapkan aturan pergeseran yang cerdas saat terjadi ketidakcocokan pada P[j] dengan karakter teks x:

1. Jika karakter x ada di dalam pola P: Pola digeser ke kanan sehingga kemunculan terakhir dari x di dalam pola (yang ditemukan melalui L(x)) menjadi sejajar dengan karakter x di teks T[i].



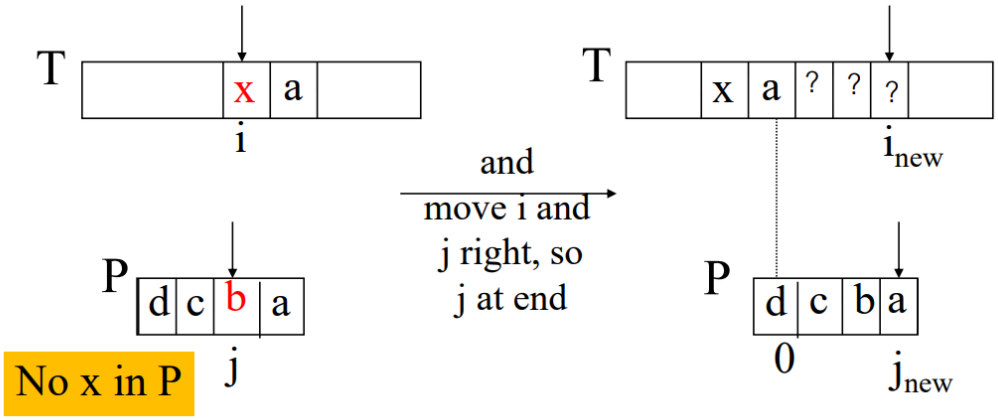
Gambar 2.3. Case 1
(sumber: [Pencocokan String (String/Pattern Matching)])

2. Jika pergeseran menghasilkan gerakan mundur atau diam: Jika kemunculan terakhir x di pola berada di sebelah kanan posisi ketidakcocokan j, aturan di atas akan menyarankan pergeseran negatif. Untuk menghindari ini, pola P cukup digeser satu posisi ke kanan (T[i+1]).



Gambar 2.4. Case 2
(sumber: [Pencocokan String (String/Pattern Matching)])

3. Jika karakter x tidak ada di dalam pola: Kasus 1 dan 2 tidak berhasil, maka ini adalah kasus terbaik. Kita tahu bahwa tidak ada kemungkinan kecocokan sampai kita melewati posisi x sepenuhnya. Oleh karena itu, seluruh pola P digeser ke kanan melewati posisi x sehingga P[0] sejajar dengan T[i+1].



Gambar 2.5. Case 3
(sumber: [Pencocokan String (String/Pattern Matching)])

Aturan-aturan ini memastikan bahwa setiap pergeseran selalu progresif dan seoptimal mungkin berdasarkan informasi yang ada.

Kompleksitas waktu kasus terburuk pada algoritma Boyer-Moore adalah O(nm + |A|), di mana n adalah panjang teks, m adalah panjang pola, dan |A| adalah ukuran alfabet. Kasus terburuk ini jarang terjadi dalam praktik.

Keunggulan sesungguhnya dari Boyer-Moore terlihat pada kinerja rata-ratanya. Algoritma ini sangat cepat ketika digunakan pada alfabet berukuran besar, seperti teks bahasa alami (misalnya, Bahasa Inggris atau Indonesia). Hal ini karena kemungkinan besar karakter buruk tidak ada atau berada jauh di awal pola, sehingga memungkinkan lompatan yang sangat besar. Sebaliknya, kinerjanya cenderung melambat pada alfabet berukuran kecil, seperti data biner ({0, 1}), karena pergeseran yang dihasilkan cenderung lebih kecil.

2.4 Algoritma Aho-Corasick

Algoritma Aho-Corasick adalah algoritma pencocokan string yang secara fundamental berbeda dari KMP atau Boyer-Moore. Jika kedua algoritma tersebut dirancang untuk mencari satu pola tunggal, Aho-Corasick dirancang untuk tugas yang jauh lebih besar, seperti menemukan semua kemunculan dari sekumpulan pola (kamus kata kunci) di dalam sebuah teks hanya dalam satu kali pemindaian. Algoritma ini menggabungkan konsep dari struktur data trie (pohon prefix) dengan teori finite automata (mesin keadaan terbatas) untuk mencapai efisiensi yang sangat baik.

2.4.1 Struktur Data Inti: Automaton Aho-Corasick

Langkah pertama dan paling krusial dalam algoritma ini adalah fase pra-pemrosesan, di mana semua pola dari kamus kata kunci dibangun menjadi sebuah struktur tunggal berbentuk finite automaton. Struktur ini pada dasarnya adalah sebuah trie (pohon prefix) yang dimodifikasi.

Dalam trie ini, setiap simpul (node) merepresentasikan sebuah prefix dari satu atau lebih pola. Setiap jalur dari akar ke sebuah simpul adalah sebuah string. Jika sebuah simpul menandai akhir dari sebuah pola lengkap, simpul tersebut ditandai sebagai simpul keluaran (output node).

Trie ini kemudian diperkaya dengan dua jenis transisi utama:

1. Tautan Sukses (Success Links): Ini adalah tautan standar dalam sebuah trie. Dari sebuah simpul, tautan sukses untuk karakter c akan menunjuk ke simpul anak yang merepresentasikan penambahan karakter c pada prefix saat ini.
2. Tautan Kegagalan (Failure Links): Ini adalah inovasi kunci dari Aho-Corasick dan analog dengan fungsi pinggiran pada KMP. Tautan ini digunakan ketika tidak ada Tautan Sukses untuk karakter berikutnya dari teks.

2.4.2 Mesin Penggerak: Tautan Sukses dan Kegagalan

Saat memindai teks, algoritma bergerak di dalam automaton. Jika karakter teks berikutnya cocok dengan salah satu Tautan Sukses dari simpul saat ini, algoritma akan maju ke simpul berikutnya.

Namun, jika terjadi ketidakcocokan (tidak ada Tautan Sukses untuk karakter tersebut), algoritma tidak perlu kembali ke awal. Di sinilah Tautan Kegagalan berperan. Tautan kegagalan dari sebuah simpul u akan menunjuk ke simpul lain v di dalam trie, di mana string yang direpresentasikan oleh v adalah sufiks sejati (proper suffix) terpanjang dari string yang direpresentasikan oleh u yang juga merupakan prefix dari salah satu pola di dalam kamus.

Secara sederhana, tautan ini memungkinkan mesin untuk "jatuh" ke keadaan lain yang paling relevan tanpa harus memindai ulang teks, mempertahankan informasi dari segmen yang sudah cocok.

2.4.3 Fase Algoritma dan Kompleksitas

Algoritma Aho-Corasick bekerja dalam dua fase:

1. Fase Pembangunan (Construction):
    a. Membangun trie dasar dari semua pola dalam kamus.
    b. Menandai simpul-simpul keluaran.
    c. Menghitung Tautan Kegagalan untuk setiap simpul di dalam trie. Fase ini memiliki kompleksitas O(L), di mana L adalah total panjang semua pola dalam kamus.
2. Fase Pencarian (Searching):
    a. Memindai teks masukan karakter per karakter, dimulai dari akar automaton.
    b. Untuk setiap karakter, ikuti Tautan Sukses. Jika tidak ada, ikuti Tautan Kegagalan hingga menemukan kecocokan atau kembali ke akar.
    c. Setiap kali sebuah simpul dimasuki, periksa apakah itu adalah simpul keluaran. Jika ya, sebuah pola telah ditemukan. Fase pencarian ini memiliki kompleksitas O(n), di mana n adalah panjang teks.

Dengan demikian, kompleksitas waktu total dari Algoritma Aho-Corasick adalah O(n + L), yang sangat efisien.

2.5 Penjelasan Aplikasi Desktop yang Dibangun

Sebagai bagian dari Tugas Besar 3 ini, sebuah sistem Applicant Tracking System (ATS) fungsional dikembangkan dalam bentuk aplikasi desktop. Sistem ini dibangun menggunakan bahasa pemrograman Python dengan Flet sebagai pustaka untuk membangun antarmuka pengguna grafis (GUI). Aplikasi ini juga dirancang untuk membantu para perekrut (HR) mengidentifikasi kandidat yang paling relevan dengan kebutuhan perusahaan berdasarkan serangkaian kata kunci, tanpa harus memeriksa setiap CV secara manual.

2.5.1 Arsitektur dan Alur Kerja Sistem

Sistem ini memiliki arsitektur terintegrasi yang menggabungkan pemrosesan dokumen, manajemen basis data, dan algoritma pencocokan string dalam satu alur kerja yang kohesif.

1. Ekstraksi Data dari PDF: Alur kerja dimulai ketika pengguna menyediakan sekumpulan data CV dalam format PDF. Untuk setiap dokumen, sistem secara otomatis melakukan proses ekstraksi teks, mengubah seluruh konten CV menjadi sebuah string tunggal yang menjadi dasar bagi proses analisis dan pencocokan pola.
2. Ekstraksi Informasi dengan Regular Expression (Regex): Setelah teks mentah didapatkan, sistem menggunakan Regular Expression (Regex) untuk mengidentifikasi dan mengekstrak informasi terstruktur yang akan digunakan untuk CV Summary.
3. Penyimpanan di Basis Data: Informasi terstruktur yang telah diekstrak kemudian disimpan ke dalam basis data MySQL. Skema basis data terdiri dari dua tabel utama: ApplicantProfile untuk menyimpan data pribadi pelamar dan ApplicationDetail untuk menyimpan detail lamaran, termasuk cv_path (lokasi file PDF asli). Pendekatan ini memisahkan data mentah (teks CV) dari data terstruktur (profil), sehingga mengoptimalkan kecepatan akses untuk fitur-fitur aplikasi.

2.5.2 Mekanisme Pencarian dan Penampilan Top CV

Inti dari fungsionalitas aplikasi ini terletak pada mekanisme pencariannya, yang dilakukan sepenuhnya secara in-memory untuk menjamin kecepatan dan responsivitas.

1. Pencocokan Presisi (Exact Matching): Pengguna dapat memilih antara algoritma Knuth-Morris-Pratt (KMP) dan Boyer-Moore (BM). Selain itu, sistem juga mengimplementasikan Aho-Corasick, yang dioptimalkan untuk menangani pencarian multi-pattern dengan mencocokkan semua kata kunci secara simultan dalam satu kali pemindaian teks.
2. Pencocokan Fleksibel (Fuzzy Matching): Jika ada kata kunci yang tidak ditemukan melalui exact matching (misalnya karena kesalahan ketik), sistem secara otomatis melakukan pencarian tahap kedua. Pada tahap ini, Algoritma Levenshtein Distance digunakan untuk mengukur tingkat kemiripan antara kata kunci dan kata-kata di dalam CV. Hasil yang memiliki tingkat kemiripan di atas ambang batas (threshold) yang ditentukan akan dianggap sebagai kecocokan. Threshold yang digunakan dari 0.7 hingga 0.9. Kami menggunakan threshold ini untuk beberapa kasus, jika keyword yang dicari berupa string panjang, maka kemungkinan terjadinya typo lebih besar sehingga kami membuat thresholdnya direndahkan, dan untuk keyword yang dicari berupa string yang relatif pendek, maka kemungkinan terjadinya typo lebih jarang sehingga kami membuat thresholdnya ditinggikan.
3. Penampilan Hasil: Setelah proses pencarian selesai, sistem mengakumulasi jumlah kecocokan untuk setiap kandidat dan mengurutkan hasilnya secara menurun. Kandidat dengan jumlah kecocokan terbanyak ditampilkan di posisi paling atas, sehingga memudahkan perekrut untuk fokus pada profil yang paling relevan.

2.5.3 Antarmuka Pengguna dan Fungsionalitas

Aplikasi ini dilengkapi dengan antarmuka pengguna grafis (GUI) yang dirancang agar intuitif dan mudah digunakan.

1. Halaman Pencarian Utama: Di halaman ini, pengguna dapat memasukkan daftar kata kunci, memilih algoritma pencarian, menentukan jumlah hasil teratas, dan memulai proses dengan menekan tombol "Search".
2. Tampilan Hasil: Setelah pencarian, hasil ditampilkan dalam bentuk kartu-kartu CV yang berisi nama kandidat, jumlah total kecocokan, dan rincian kata kunci yang ditemukan. Waktu eksekusi untuk proses exact dan fuzzy matching juga ditampilkan secara terpisah.
3. Modal Rincian/Summary: Dengan mengklik salah satu kartu hasil, pengguna diarahkan ke modal summary. Modal ini menampilkan detail informasi kandidat yang telah diekstrak (keahlian, pengalaman, pendidikan, dll.) dan menyediakan tombol "View CV" untuk membuka file PDF asli.

BAB III

ANALISIS PEMECAHAN MASALAH

3.1 Langkah-Langkah Pemecahan Masalah

Untuk membangun sistem CV-ATS (Applicant Tracking System) yang baik, kami melakukan dekomposisi masalah secara sistematis. Pendekatan ini memungkinkan kami untuk fokus pada setiap aspek fungsional secara mendalam, mulai dari pengolahan data mentah hingga penyajian hasil kepada pengguna. Berikut adalah langkah-langkah pemecahan masalah yang kami terapkan dalam pengembangan sistem ini.

3.1.1 Pengolahan dan Pra-pemrosesan Data CV

Fondasi dari sistem ATS yang kami buat adalah kemampuannya untuk mengolah data CV secara presisi. Tahap ini berfokus pada transformasi dokumen PDF menjadi teks yang bersih dan terstruktur.

3.1.3.1 Ekstraksi Teks dari PDF

Ekstraksi Teks dan Informasi CV Terstruktur merupakan langkah awal adalah mengonversi dokumen PDF menjadi string teks menggunakan pustaka PDFPlumber. Untuk memaksimalkan akurasi, kami mengimplementasikan sebuah modul parser yang melakukan deteksi seksi dinamis. Modul ini menerapkan pencocokan pola dengan confidence scoring untuk secara cerdas mengidentifikasi dan memisahkan bagian-bagian penting seperti ringkasan (summary), keahlian (skills), pengalaman kerja (experience), dan pendidikan (education). Selain itu, pattern Regex digunakan untuk mengekstrak informasi kontak (nama, email, telepon), dan sebuah mekanisme validasi konten memastikan akurasi deteksi pada setiap bagian.

3.1.2 Implementasi Algoritma String Matching

3.1.2.1 Pencocokan Pola Tunggal: KMP dan Boyer-Moore

Untuk pencocokan presisi (exact match), kami mengimplementasikan dua algoritma klasik. Pertama, Knuth-Morris-Pratt (KMP), yang menggunakan Fungsi

Pinggiran yang membuat array sebagai failure function untuk melakukan pergeseran yang "cerdas" dan menghindari backtracking. Kedua, Boyer-Moore, yang bekerja dengan memindai pola dari kanan ke kiri dan memanfaatkan heuristik bad character. Heuristik ini menggunakan tabel kemunculan terakhir karakter (last occurrence function) untuk menghitung lompatan pergeseran yang besar ketika terjadi ketidakcocokan.

3.1.2.2 Pencocokan Multi-Pola: Aho-Corasick

Untuk menangani pencarian beberapa kata kunci secara langsung dan efisien, kami mengimplementasikan Aho-Corasick. Algoritma ini membangun sebuah finite automaton dari semua kata kunci dalam bentuk struktur data trie yang diperkaya dengan failure links. Mekanisme ini memungkinkan sistem untuk menemukan semua kemunculan dari seluruh kata kunci hanya dalam satu kali pemindaian teks, dengan kompleksitas waktu linear terhadap panjang teks dan total panjang pola.

3.1.2.3 Pencocokan Fleksibel (*Fuzzy Matching*): Levenshtein Distance

Untuk mengatasi keterbatasan exact matching yang tidak dapat mentolerir kesalahan ketik (typo), kami mengimplementasikan pencocokan fleksibel menggunakan Algoritma Levenshtein Distance. Algoritma ini tidak mencari kesamaan persis, melainkan mengukur edit distance antara dua string. Jarak ini didefinisikan sebagai jumlah minimum operasi penyisipan (insertion), penghapusan (deletion), atau penggantian (substitution) karakter yang diperlukan untuk mengubah satu kata menjadi kata lainnya.

Dalam sistem kami, ketika sebuah kata kunci tidak ditemukan secara eksak, algoritma ini digunakan untuk membandingkannya dengan kata-kata di dalam teks CV. Jika jarak Levenshtein berada di bawah ambang batas (threshold) yang telah ditentukan, kata tersebut dianggap sebagai kecocokan yang relevan. Pendekatan ini secara signifikan meningkatkan daya jangkau pencarian, memastikan kandidat potensial tidak terlewat hanya karena perbedaan penulisan minor.

3.1.2.4 Perancangan Layer Persistensi dan Keamanan Data

- Manajemen Koneksi Basis Data: Sebuah modul koneksi basis data dikembangkan sebagai layer abstraksi untuk semua interaksi dengan MySQL. Modul ini dilengkapi dengan penanganan error yang robust, manajemen koneksi yang efisien, serta mekanisme transaksi dengan automatic rollback untuk menjaga integritas dan konsistensi data.
- Abstraksi Akses Data dengan Repository Pattern: Untuk memisahkan logika bisnis dari detail operasi basis data, kami menerapkan Repository Pattern. Sebuah kelas repository menyediakan antarmuka yang bersih dan terdefinisi dengan baik serta fungsi pencarian yang lebih kompleks. Ini memungkinkan logika aplikasi untuk berinteraksi dengan data tanpa perlu mengetahui detail implementasi SQL.
- Enkripsi Data Sensitif: Keamanan data pelamar tersebut merupakan prioritas sehingga kami mengintegrasikan sebuah modul enkripsi khusus untuk melindungi informasi sensitif sebelum disimpan ke dalam basis data, memastikan kerahasiaan data tetap terjaga.

3.1.3 Pengembangan Sistem Peringkat dan Penilaian (*Scoring & Ranking*)

Untuk menyajikan hasil yang paling relevan, kami mengembangkan sistem penilaian multi-faceted.

3.1.4.1 Penilaian Berbasis Frekuensi

Setiap hasil pencarian dienkapsulasi dalam sebuah struktur data yang menghitung skor berdasarkan jumlah total kecocokan dan frekuensi kemunculan setiap kata kunci.

3.1.4.2 Pengukuran Waktu Pencarian

Sistem secara transparan mencatat dan menampilkan waktu eksekusi yang dibutuhkan untuk fase exact search dan fuzzy search secara terpisah, memberikan wawasan performa kepada pengguna.

3.1.4 Perancangan Antarmuka Pengguna yang Interaktif

Fungsionalitas yang kuat harus didukung oleh antarmuka yang mudah digunakan dan informatif.

3.1.4.1 Arsitektur Antarmuka Berbasis Komponen

Antarmuka pengguna dikembangkan menggunakan framework Flet dengan pendekatan arsitektur berbasis komponen. Setiap elemen UI, seperti panel konfigurasi pencarian, area tampilan hasil, dan visualisasi konten dibangun sebagai komponen mandiri yang dapat digunakan kembali dan mudah dipelihara.

3.1.4.2 Umpan Balik Proses Pencarian secara Real-Time

Untuk meningkatkan pengalaman pengguna, sistem memberikan umpan balik visual secara real-time selama proses pencarian. Indikator kemajuan, pembaruan status, dan hasil yang muncul secara dinamis memberikan transparansi kepada pengguna mengenai apa yang sedang terjadi di latar belakang.

3.1.4.3 Visualisasi Hasil yang Interaktif

Hasil pencarian disajikan dalam bentuk "kartu" interaktif yang menampilkan informasi ringkas pelamar, skor relevansi, dan kata kunci yang cocok. Pengguna dapat mengklik kartu ini untuk melihat ringkasan CV yang lebih detail atau membuka konten PDF asli dengan kata kunci yang relevan telah ditandai (highlighted).

3.1.5 Pengujian Performa dan Optimalisasi Sistem

Tahap akhir yang kami lakukan berfokus pada validasi dan penjaminan kualitas sistem.

3.1.4.1 Benchmarking Kinerja Algoritma

Kami membangun sebuah framework pengujian untuk melakukan benchmarking dan membandingkan kinerja dari setiap algoritma string matching. Metrik yang diukur meliputi waktu eksekusi, jumlah total kecocokan, dan akurasi pada dataset CV nyata, memberikan data kuantitatif untuk analisis perbandingan.

3.1.4.2 Integrasi Analitik untuk Pemantauan

Untuk tujuan pemantauan dan optimalisasi berkelanjutan, kami mengintegrasikan modul pelacakan statistik. Metrik seperti ukuran automaton (untuk Aho-Corasick), waktu eksekusi, dan kepadatan kecocokan (match density) dicatat. Data ini sangat berharga untuk melakukan tuning performa dan memahami perilaku sistem di bawah beban kerja yang berbeda.

3.2 Proses pemetaan masalah menjadi elemen-elemen algoritma KMP dan BM.

Untuk membangun fungsionalitas pencarian pada sistem ATS, setiap algoritma teoretis perlu dipetakan menjadi komponen-komponen perangkat lunak yang konkret. Bagian ini merinci bagaimana konsep-konsep dari algoritma Knuth-Morris-Pratt dan Boyer-Moore yang diadaptasi dan diimplementasikan dalam sistem kami.

3.2.1 Algoritma Knuth-Morris-Pratt (KMP)

Implementasi KMP berfokus pada dua fase utama yang dijelaskan dalam teori: pra-pemrosesan pola dan pencarian linear.

3.2.1.1 Pra-pemrosesan Pola dengan Fungsi Pinggiran (Border Function)

Langkah pertama dalam implementasi kami adalah fase pra-pemrosesan. Kami membangun sebuah fungsi untuk menghitung Fungsi Pinggiran (juga dikenal sebagai failure function). Fungsi ini menghasilkan sebuah array yang disebut LPS (Longest Proper Prefix which is also Suffix). Array ini adalah representasi konkret dari Fungsi Pinggiran dan menjadi struktur data kunci yang menyimpan informasi tentang pergeseran "cerdas" yang bisa dilakukan. Proses ini memiliki kompleksitas O(m) dan menjadi fondasi untuk efisiensi algoritma saat pencarian.

3.2.1.2 Strategi Pencarian Linear Tanpa Mundur (Backtracking)

Fase pencarian diimplementasikan sebagai sebuah pemindaian tunggal (linear) terhadap teks CV. Ketika terjadi ketidakcocokan antara teks dan pola, alih-alih menggeser pola satu langkah, algoritma kami merujuk pada array LPS

yang telah dibuat. Nilai dalam array ini digunakan untuk menentukan indeks selanjutnya pada pola yang harus dibandingkan, sehingga proses pencarian dapat "melompat" ke posisi yang relevan tanpa pernah menggerakkan pointer pada teks ke belakang. Ini secara langsung mengimplementasikan keunggulan utama KMP, yaitu ideal untuk data yang besar dan bersifat stream.

3.2.2 Algoritma Boyer-Moore (BM)

Implementasi Boyer-Moore berpusat pada dua pendekatan, yaitu pencocokan mundur dan pemanfaatan heuristik karakter buruk. Strategi pencarian kami secara faithful mengadopsi teknik pencocokan mundur (looking-glass technique). Proses perbandingan karakter dimulai dari akhir pola, bergerak ke arah kiri. Ketika terjadi ketidakcocokan, kami menerapkan teknik lompatan karakter dengan memanfaatkan heuristik karakter buruk. Untuk ini, sebuah fungsi pra-pemrosesan dibangun untuk menciptakan Fungsi Kemunculan Terakhir (Last Occurrence Function). Fungsi ini menghasilkan sebuah lookup table yang memetakan setiap karakter dalam alfabet ke posisi terakhir kemunculannya di dalam pola. Tabel inilah yang digunakan untuk menghitung pergeseran maksimal yang aman.

3.2.3 Adaptasi Algoritma untuk Domain Perekrutan (CV/Resume)

Mengaplikasikan algoritma teoretis ke dalam domain dunia nyata seperti analisis CV memerlukan beberapa adaptasi praktis agar hasilnya relevan dan akurat.

3.2.1.3 Penanganan Case-Insensitivity melalui Normalisasi Teks

Salah satu tantangan utama pada teks CV adalah variasi penulisan. Seorang kandidat bisa menulis "Python", "python", atau "PYTHON". Untuk memastikan algoritma dapat mengenali semua variasi ini sebagai entitas yang sama, sebuah langkah normalisasi diimplementasikan. Sebelum proses pencocokan dilakukan, baik teks dari CV maupun kata kunci dari pengguna dikonversi ke format seragam. Langkah ini memastikan bahwa pencarian tidak sensitif terhadap besar-kecilnya huruf (case-insensitive), yang merupakan syarat mutlak untuk sistem ATS yang efektif.

3.2.1.4 Kontekstualisasi Hasil Pencarian untuk Analisis

Sebuah algoritma pencocokan string pada dasarnya hanya menemukan apakah sebuah pola ada atau tidak. Namun, untuk kebutuhan ATS, informasi ini tidak cukup. Oleh karena itu, implementasi kami tidak hanya mengembalikan nilai boolean (ditemukan/tidak ditemukan), tetapi sebuah daftar berisi semua indeks posisi di mana kata kunci ditemukan. Pemetaan dari sekadar "menemukan" menjadi "menemukan di mana dan berapa kali" ini sangat krusial. Ini memungkinkan sistem untuk melakukan analisis lebih lanjut, seperti menghitung frekuensi kemunculan sebuah keahlian dan bahkan menampilkan cuplikan teks di sekitar kata kunci yang cocok, memberikan konteks yang lebih kaya kepada perekrut.

3.3 Fitur Fungsional dan Arsitektur Aplikasi Desktop yang Dibangun

3.3.1 Fitur Fungsional Utama

3.3.1.1 Manajemen Basis Data dan Statistik

Untuk menjamin pertukaran data yang dinamis, sistem dilengkapi dengan sebuah modul manajemen koneksi ke basis data MySQL. Modul ini menangani proses koneksi dan diskoneksi secara otomatis, yang lengkap juga dengan penanganan *error* untuk menjaga stabilitas. Sebagai fitur tambahan, halaman utama aplikasi menampilkan statistik ringkas dari basis data, seperti jumlah total CV, jumlah peran pekerjaan yang tersedia, dan rincian jumlah CV untuk setiap peran.

3.3.1.2 Implementasi dan Pemilihan Algoritma Pencarian

Inti dari sistem ini adalah kemampuannya untuk melakukan pencocokan string menggunakan algoritma-algoritma, seperti Knuth-Morris-Pratt (KMP), Boyer-Moore, dan Aho-Corasick. Antarmuka pengguna menyediakan komponen radio button yang memungkinkan pengguna untuk secara dinamis memilih algoritma mana yang akan digunakan untuk proses pencarian, memberikan fleksibilitas dan sarana untuk perbandingan kinerja secara langsung.

3.3.1.3 Pencarian CV Berbasis Kata Kunci

Pengguna dapat memasukkan beberapa kata kunci sekaligus, yang dipisahkan dengan koma, melalui kolom input yang disediakan. Sistem juga memungkinkan konfigurasi untuk menampilkan sejumlah hasil teratas (misalnya, Top 10) sesuai kebutuhan. Untuk meningkatkan relevansi, pencarian dilengkapi dengan sistem penilaian kemiripan (similarity scoring) menggunakan Algoritma Levenshtein Distance, yang menangani kasus kesalahan pengetikan. Proses pencarian dieksekusi secara real-time dengan indikator kemajuan visual untuk memberikan umpan balik kepada pengguna.

3.3.1.4 Ekstraksi dan Parsing Konten CV

Aplikasi ini mampu memproses dokumen PDF secara otomatis. Sebuah modul parser PDF bertugas mengekstrak seluruh teks dari file. Selanjutnya, modul extractor CV akan mem-parsing teks mentah tersebut untuk mengidentifikasi dan memisahkan informasi penting seperti data pribadi, daftar keahlian (skills), pengalaman kerja, dan riwayat pendidikan ke dalam format data yang terstruktur.

3.3.1.5 Tampilan Ringkasan dan Detail CV

Hasil pencarian disajikan dalam bentuk kartu-kartu interaktif. Setiap kartu dapat diklik untuk membuka jendela modal yang menampilkan ringkasan detail dari CV kandidat. Informasi yang ditampilkan mencakup data personal, ringkasan profesional, daftar keahlian dalam format tag, riwayat kerja dan pendidikan secara kronologis, serta daftar kata kunci yang cocok yang telah ditandai (highlighted) untuk memudahkan identifikasi.

3.3.1.6 Enkripsi Data

Untuk meningkatkan keamanan, sistem dilengkapi dengan utilitas enkripsi. Data sensitif pelamar akan dienkripsi sebelum disimpan ke basis data, yang diakses melalui sebuah *repository* terenkripsi khusus untuk melindungi privasi data.

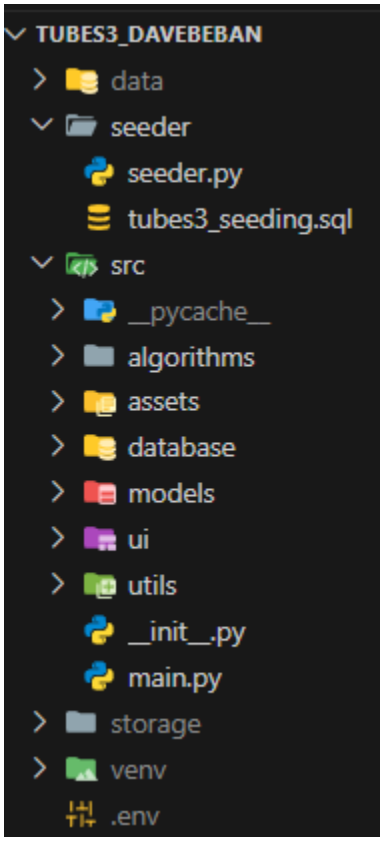
3.3.2 Arsitektur Aplikasi Desktop

3.3.2.1 Kerangka

1. UI Framework: Aplikasi ini dibangun menggunakan Flet, sebuah kerangka kerja berbasis Python yang memungkinkan pengembangan aplikasi desktop cross-platform (Windows, macOS, Linux).
2. Database: MySQL dipilih sebagai sistem manajemen basis data, dengan interaksi yang ditangani oleh pustaka mysql-connector-python.
3. Pola Arsitektur: Sistem mengadopsi pola Model-View-Handler (MVH), sebuah variasi dari pola MVC/MVP yang memisahkan data (Model), tampilan (View), dan logika event (Handler).

3.3.2.2 Struktur Proyek yang Modular

Struktur direktori proyek dirancang untuk menegakkan pemisahan tanggung jawab (separation of concerns):



Gambar 3.1. Folder Struct

3.3.2.3 Arsitektur Berlapis (Layered Architecture)

Aplikasi ini dibangun di atas empat lapisan logis yang berbeda:

1. Presentation Layer (UI): Bertanggung jawab atas semua hal yang dilihat dan diinteraksikan oleh pengguna. Dikelola oleh direktori ui/, yang berisi komponen-komponen UI statis dan event handler untuk logika bisnis.
2. Business Logic Layer: Merupakan otak dari aplikasi, tempat logika inti berada. Ini mencakup implementasi algoritma pencocokan string (algorithms/) dan logika pemrosesan CV (utils/).
3. Data Access Layer (DAL): Menjembatani lapisan logika bisnis dengan basis data. Diimplementasikan melalui Repository Pattern (database/repository.py), lapisan ini mengabstraksi semua operasi data.
4. Data Layer: Merupakan lapisan paling bawah, yang terdiri dari basis data MySQL itu sendiri, dengan skema yang telah dinormalisasi untuk menyimpan data aplikasi.

3.3.2.4 Pola Desain Perangkat Lunak (Design Patterns)

Beberapa pola desain kunci diterapkan untuk meningkatkan kualitas kode:

1. Repository Pattern: Mengisolasi logika akses data, sehingga lapisan bisnis tidak perlu tahu tentang detail implementasi SQL. Ini membuat sistem lebih mudah diuji dan dimodifikasi.
2. Factory Pattern: Diterapkan untuk pembuatan komponen UI, memastikan elemen-elemen antarmuka (seperti kartu hasil) dibuat secara konsisten di seluruh aplikasi.
3. Strategy Pattern: Digunakan dalam pemilihan algoritma pencarian. Pengguna dapat memilih "strategi" (KMP, Boyer-Moore, atau Aho-Corasick) saat runtime, dan sistem akan menggunakan implementasi yang sesuai tanpa mengubah logika pemanggil.

3.3.2.5 Alur Pengalaman Pengguna (User Experience Flow)

Dari sudut pandang pengguna, alur aplikasi berjalan sebagai berikut:

1. Inisialisasi: Aplikasi memuat dan memverifikasi koneksi ke basis data.
2. Konfigurasi Pencarian: Pengguna memilih algoritma, memasukkan kata kunci, dan mengatur jumlah hasil.
3. Eksekusi Pencarian: Proses pencarian dijalankan dengan pelacakan kemajuan (progress tracking).
4. Tampilan Hasil: Hasil ditampilkan dalam kartu-kartu interaktif.
5. Tampilan Detail: Pengguna mengklik kartu untuk melihat ringkasan CV yang komprehensif.

3.3.2.6 Fitur Performa dan Skalabilitas

Untuk memastikan aplikasi tetap responsif, beberapa fitur performa diimplementasikan:

1. Lazy Loading: Konten penuh dari sebuah CV hanya dimuat dari basis data saat pengguna memintanya, bukan saat pencarian awal.
2. Manajemen Koneksi: Koneksi ke basis data digunakan kembali untuk mengurangi latensi.
3. Penanganan Error: Sistem dilengkapi manajemen error yang komprehensif untuk menangani kegagalan (misalnya, koneksi database terputus) dan memberikan umpan balik yang jelas kepada pengguna.

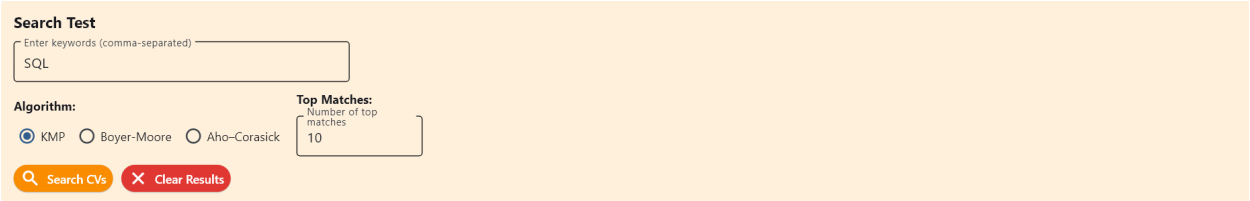
3.4 Ilustrasi Kasus

Untuk dapat menggunakan sistem pencarian kandidat, pengguna (perekrut) pertama-tama membuka aplikasi desktop. Halaman utama akan langsung menampilkan antarmuka untuk melakukan pencarian. Kasus penggunaan sistem ATS ini dapat dianalisis dengan mencoba beberapa skenario pencarian yang berbeda.

1. Skenario 1: Pencarian Presisi (Exact Match) Satu Keahlian

Dalam skenario ini, seorang perekrut ingin mencari kandidat yang memiliki keahlian spesifik dan krusial, yaitu menguasai "SQL". Pengguna akan memasukkan parameter berikut pada antarmuka utama:

- Keywords: SQL
- Search Algorithm: KMP (dipilih melalui radio button)
- Top N Matches: 10

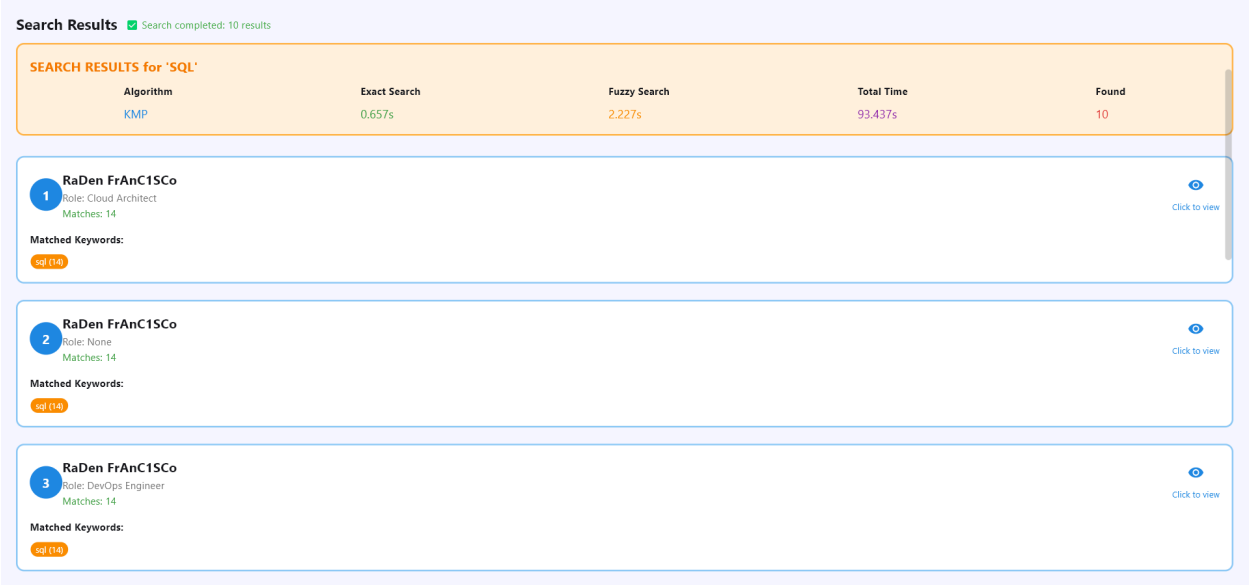


Gambar 3.2. Input Pencarian "SQL"

Saat tombol "Search" ditekan, backend akan mencatat waktu mulai untuk pengukuran performa. Proses pencarian kemudian dimulai:

1. Sistem memuat teks mentah (pdf) dari seluruh CV yang tersimpan di dalam basis data ke dalam memori.
2. Algoritma KMP dijalankan pada setiap teks CV untuk mencari kemunculan kata kunci "SQL" secara persis.
3. Setiap kali kata kunci ditemukan dalam sebuah CV, penghitung kecocokan (match count) untuk CV tersebut akan bertambah.
4. Setelah semua CV selesai dipindai, sistem akan mengurutkan kandidat berdasarkan jumlah kecocokan tertinggi dan menampilkan 10 kandidat teratas.

Hasilnya akan ditampilkan pada area hasil di bawah panel pencarian. Setiap kandidat akan direpresentasikan oleh sebuah "kartu" yang berisi informasi ringkas.



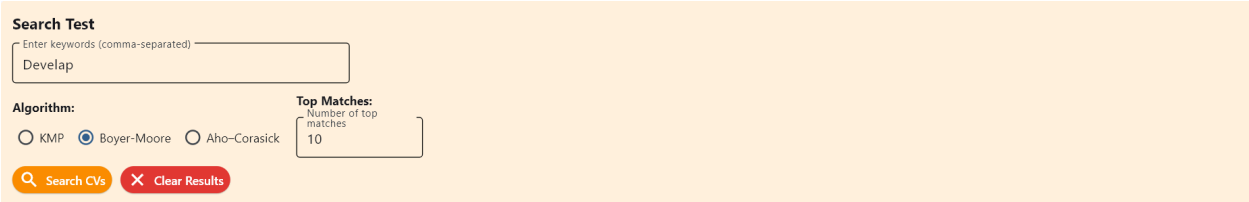
Gambar 3.3. Hasil Pencarian untuk Kata Kunci "SQL"

Pada kartu hasil, terlihat nama kandidat, total kecocokan, dan rincian kata kunci yang ditemukan (misalnya, SQL(10)). Di bagian atas, ditampilkan statistik pencarian, yaitu Waktu Eksekusi (Exact Match), yang menunjukkan berapa lama proses pencarian menggunakan KMP berlangsung, jumlah kandidat yang diminta dan yang ditampilkan.

2. Skenario 2: Pencarian Kompleks dengan Fuzzy Match

Selanjutnya, perekrut ingin mencari kandidat untuk posisi Data Scientist dengan beberapa kriteria keahlian. Namun, saat memasukkan kata kunci, terjadi sedikit kesalahan pengetikan.

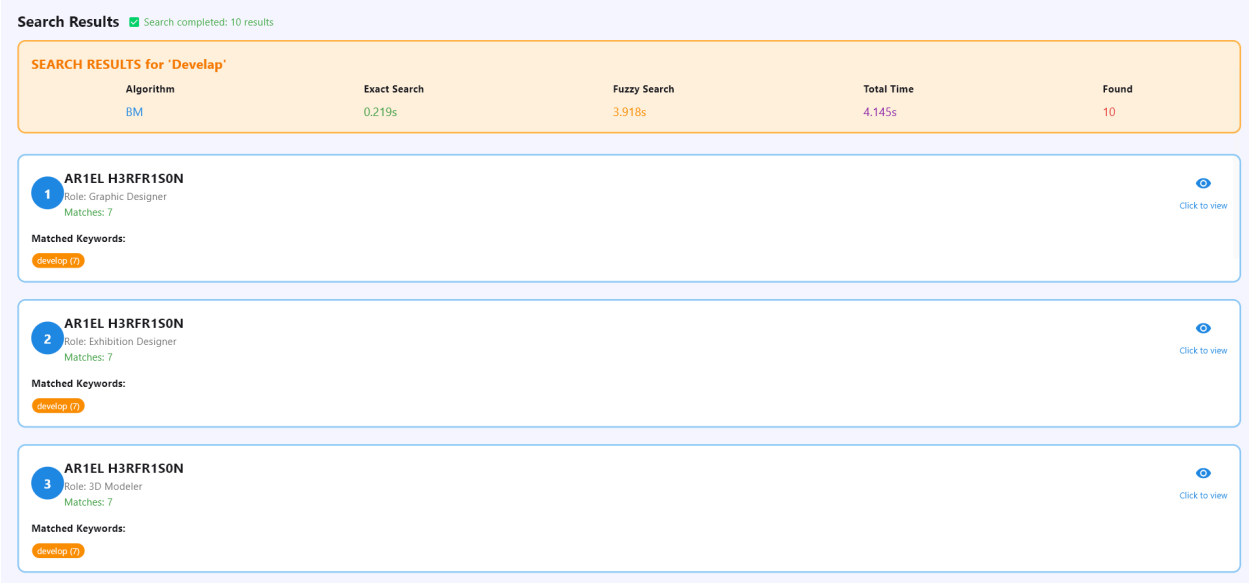
- Keywords: Develap(terdapat typo "Develap" seharusnya "Develop")
- Search Algorithm: Boyer-Moore
- Top N Matches: 10



Gambar 3.4. Input Pencarian "Develap"

Proses yang terjadi di backend lebih kompleks dan melibatkan dua tahap:

1. Tahap Exact Match: Algoritma Boyer-Moore dijalankan untuk mencari kata kunci "Develap" di keseluruhan CV. Namun, kata kunci "Develap" tidak ditemukan di CV manapun. Waktu untuk tahap ini dicatat.
2. Tahap Fuzzy Match: Karena "Develap" gagal ditemukan, sistem secara otomatis memicu pencarian fuzzy hanya untuk kata kunci tersebut. Algoritma Levenshtein Distance akan membandingkan "Develap" dengan kata-kata lain di dalam CV. Sistem akan menemukan bahwa "Develap" memiliki jarak editorial yang sangat kecil dari "Develop" dan menganggapnya sebagai kecocokan yang valid. Waktu untuk tahap kedua ini juga dicatat.



Gambar 3.5. Hasil Pencarian untuk Kata Kunci "Develap"

Hasilnya kini menampilkan kandidat yang relevan, di mana keahlian "Develap" tetap terdeteksi berkat fuzzy matching. Pada bagian statistik, ditampilkan waktu eksekusi: Waktu Eksekusi (Exact Match) dan Waktu Eksekusi (Fuzzy Match), memberikan gambaran performa untuk kedua proses tersebut.

3. Skenario 3: Melihat Rincian Profil Kandidat

Setelah menemukan kandidat yang menjanjikan dari skenario 2, perekrut ingin melihat profilnya lebih detail. Pengguna cukup mengklik kartu kandidat yang diinginkan.

Aksi ini akan memicu aplikasi untuk menampilkan jendela modal atau halaman baru yang berisi ringkasan CV kandidat tersebut. Informasi ini diambil dari data terstruktur yang telah diekstrak menggunakan Regex dan disimpan di basis data.

Gambar 3.6. CV Summary

Selain itu, terdapat tombol "View Original PDF" dan “View Full CV Text” yang jika diklik akan membuka file PDF asli dari kandidat tersebut dan menampilkan text dari CV tersebut, memungkinkan perekrut untuk melakukan verifikasi dan peninjauan mendalam. Dengan alur ini, pengguna dapat beralih dari pencarian skala besar ke analisis kandidat individual dengan cepat dan efisien.

BAB IV

IMPLEMENTASI DAN PENGUJIAN

4.1 Spesifikasi Teknis Program

4.1.1 Struktur Data

Struktur Data	Tampilan Kode

Applicant Profile Model	`@dataclass`
	`class ApplicantProfile:`
	`  applicant_id: Optional[int] = None`
	`  first_name: Optional[str] = None`
	`  last_name: Optional[str] = None`
	`  date_of_birth: Optional[date] = None`
	`  address: Optional[str] = None`
	`  phone_number: Optional[str] = None`
	
	`  @property`
	`  def full_name(self) -> str:`
	`  """Get full name"""`
	`  parts = []`
	`  if self.first_name:`
	`  parts.append(self.first_name)`
	`  if self.last_name:`
	`  parts.append(self.last_name)`
	`  return " ".join(parts) if parts else "Unknown"`

Application Detail Model	`@dataclass`
	`class ApplicationDetail:`
	`  detail_id: Optional[int] = None`
	`  applicant_id: Optional[int] = None`
	`  application_role: Optional[str] = None`
	`  cv_path: Optional[str] = None`
	`  applicant_profile: Optional[ApplicantProfile] = None`

CV Search Result Model	
Section Boundary Structure	
CV Summary Structure	

```python
@dataclass
class CVSearchResult:
    applicant_profile: ApplicantProfile
    application_detail: ApplicationDetail
    matched_keywords: List[Tuple[str, int]] = field(default_factory=list)
    cv_text: str = ""
    search_timing: Dict[str, float] = field(default_factory=dict)

    @property
    def total_matches(self) -> int:
        """Get total number of matches across all keywords"""
        return sum(count for _, count in self.matched_keywords)

    @property
    def number_of_matches(self) -> int:
        """Alias for total_matches for backward compatibility"""
        return self.total_matches
```

```python
class SectionBoundary(NamedTuple):
    """Information about detected section boundary"""
    section_type: str
    start_line: int
    end_line: int
    content: str
    detection_confidence: float
```

```python
@dataclass
class CVSummary:
    """Data class to store extracted CV information."""
    first_name: str = ""
    last_name: str = ""
    phone: str = ""
    address: str = ""
    date_of_birth: str = ""
    summary: str = ""
    skills: List[str] = field(default_factory=list)
    experience: List[Dict[str, str]] = field(default_factory=list)
    education: List[Dict[str, str]] = field(default_factory=list)
```

Trie Node untuk Aho-Corasick	

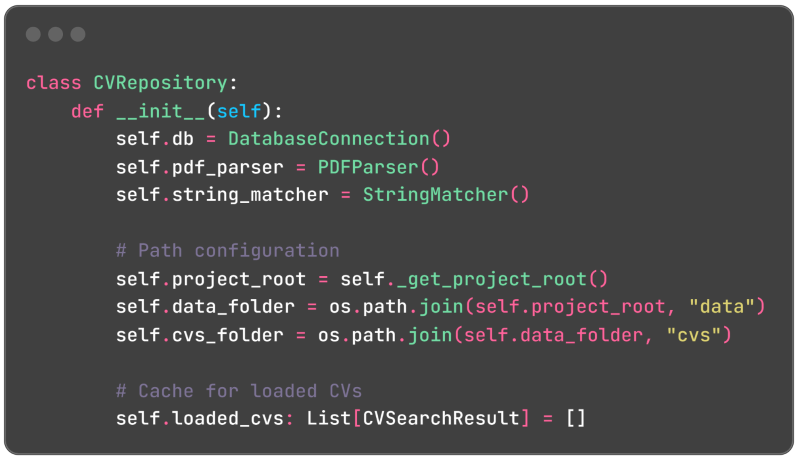
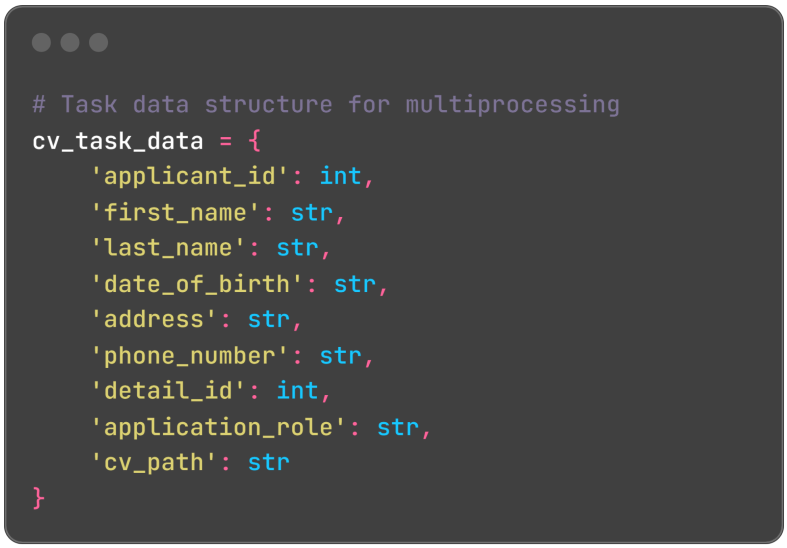
```python
class TrieNode:
    """Node for Aho-Corasick trie structure"""

    def __init__(self):
        self.children = {}
        self.failure = None
        self.output = []
        self.is_end = False
```

String Matcher Unified Interface	

```python
class StringMatcher:
    def __init__(self):
        self.kmp = KMPSearch()
        self.boyer_moore = BoyerMooreSearch()
        self.aho_corasick = AhoCorasickSearch()
        self.levenshtein = LevenshteinDistance()
```

Database Connection Configuration	

```python
class DatabaseConnection:
    def __init__(self):
        self.connection = None
        self.cursor = None
        self.config = {
            'host': os.getenv('DB_HOST', 'localhost'),
            'user': os.getenv('DB_USER', 'root'),
            'password': os.getenv('DB_PASSWORD', ''),
            'database': os.getenv('DB_NAME', 'cv_ats'),
            'port': int(os.getenv('DB_PORT', 3306)),
            'charset': 'utf8mb4',
            'collation': 'utf8mb4_unicode_ci'
        }
```

	
Multiprocessing Task Structure	

UI Components Configuration	
Search Configuration Structure	
Database Statistics Structure	
Search Timing Structure	

```python
class UIHandlers:
    def __init__(self, page: ft.Page):
        self.page = page
        self.repository = CVRepository()

        # UI Components
        self.keywords_input: ft.TextField = None
        self.algorithm_radio: ft.RadioGroup = None
        self.top_matches_input: ft.TextField = None
        self.results_container: ft.Column = None
        self.status_text: ft.Text = None
        self.progress_ring: ft.ProgressRing = None
```

```python
# Search configuration extracted from UI
search_config = {
    'keywords': str,  # dari keywords_input.value
    'algorithm': str,  # dari algorithm_radio.value ("kmp", "bm", "aho")
    'top_matches': int,  # dari top_matches_input.value
}
```

```python
# Return structure dari get_statistics() dan get_cv_summary_statistics()
statistics_structure = {
    'total_cvs': int,
    'total_roles': int,
    'role_breakdown': Dict[str, int]  # role_name -> count
}
```

```python
# Search timing tracking dalam CVSearchResult
search_timing = {
    'exact': float,  # waktu exact search dalam detik
    'fuzzy': float,  # waktu fuzzy search dalam detik
    'total': float   # total waktu search
}
```

Algorithm Performance Metrics	

```python
# Structure untuk algorithm performance tracking
algorithm_stats = {
    'algorithm_name': str,
    'pattern_length': int,
    'text_length': int,
    'matches_found': int,
    'execution_time': float,
    'comparisons_made': int
}
```

4.1.2 Fungsi

Fungsi	Tampilan Kode

Fungsi Manajemen Koneksi Database	

```python
class DatabaseConnection:
    def __init__(self):
        self.connection = None
        self.cursor = None

    def connect(self) -> bool:
        """ CONNECT: Membuka koneksi ke database"""
        try:
            self.connection = mysql.connector.connect(
                host=os.getenv('DB_HOST', 'localhost'),
                user=os.getenv('DB_USER', 'root'),
                password=os.getenv('DB_PASSWORD', ''),
                database=os.getenv('DB_NAME', 'cv_ats'),
                port=int(os.getenv('DB_PORT', 3306)),
                charset='utf8mb4',
                collation='utf8mb4_unicode_ci'
            )

            if self.connection.is_connected():
                self.cursor = self.connection.cursor(dictionary=True)
                print("Successfully connected to MySQL database")
                return True

        except Error as e:
            print(f"Error connecting to MySQL: {e}")
            return False

    def disconnect(self):
        """ DISCONNECT: Menutup koneksi database"""
        try:
            if self.cursor:
                self.cursor.close()
                self.cursor = None

            if self.connection and self.connection.is_connected():
                self.connection.close()
                self.connection = None
                print("MySQL connection closed")

        except Error as e:
            print(f"Error closing connection: {e}")

    def execute_query(self, query: str, params: tuple = None, fetch: bool = True) ->
Optional[List[Dict]]:
        """ SELECT: Mengambil data dari database"""
        try:
            if params:
                self.cursor.execute(query, params)
            else:
                self.cursor.execute(query)

            if fetch:
                return self.cursor.fetchall()  # Return data
            else:
                self.connection.commit()  # For non-SELECT queries
                return True

        except Error as e:
            print(f"Error executing query: {e}")
            if not fetch:
                self.connection.rollback()
            return None

    def execute_insert(self, query: str, params: tuple = None) -> Optional[int]:
        """INSERT: Menambah data baru, return ID yang baru dibuat"""
        try:
            if params:
                self.cursor.execute(query, params)
            else:
                self.cursor.execute(query)

            self.connection.commit()
            return self.cursor.lastrowid  # Return new ID

        except Error as e:
            print(f"Error executing insert: {e}")
            self.connection.rollback()
            return None

    def execute_update(self, query: str, params: tuple = None) -> bool:
        """UPDATE/DELETE: Mengubah atau menghapus data"""
        try:
            if params:
                self.cursor.execute(query, params)
            else:
                self.cursor.execute(query)

            self.connection.commit()
            return True

        except Error as e:
            print(f"Error executing update: {e}")
            self.connection.rollback()
            return False

    def is_connected(self) -> bool:
        """CHECK: Cek apakah masih terhubung"""
        try:
            return self.connection and self.connection.is_connected()
        except:
            return False

    def get_last_insert_id(self) -> Optional[int]:
        """GET ID: Ambil ID terakhir yang di-insert"""
        return self.cursor.lastrowid if self.cursor else None
```

Fungsi Algoritma Knuth-Morris-Pratt	

```python
class KMPSearch:
    """
    Knuth-Morris-Pratt (KMP) String Matching Algorithm

    Efficient single-pattern string matching with O(n+m) time complexity
    Uses failure function to avoid redundant comparisons
    """

    @staticmethod
    def search(text: str, pattern: str) -> List[int]:
        """
        Search for pattern in text using KMP algorithm

        Args:
            text: Text to search in
            pattern: Pattern to search for

        Returns:
            List of starting positions where pattern is found
        """
        def compute_lps(pattern):
            """Compute Longest Proper Prefix which is also Suffix (LPS) array"""
            lps = [0] * len(pattern)
            length = 0
            i = 1

            while i < len(pattern):
                if pattern[i] == pattern[length]:
                    length += 1
                    lps[i] = length
                    i += 1
                else:
                    if length != 0:
                        length = lps[length - 1]
                    else:
                        lps[i] = 0
                        i += 1
            return lps

        if not pattern:
            return []

        text = text.lower()
        pattern = pattern.lower()

        lps = compute_lps(pattern)
        matches = []
        i = j = 0

        while i < len(text):
            if pattern[j] == text[i]:
                i += 1
                j += 1

            if j == len(pattern):
                matches.append(i - j)
                j = lps[j - 1]
            elif i < len(text) and pattern[j] != text[i]:
                if j != 0:
                    j = lps[j - 1]
                else:
                    i += 1

        return matches

    @staticmethod
    def search_with_stats(text: str, pattern: str) -> dict:
        """
        KMP search with performance statistics

        Returns:
            Dictionary with matches and performance metrics
        """
        import time

        start_time = time.time()
        matches = KMPSearch.search(text, pattern)
        end_time = time.time()

        return {
            'matches': matches,
            'total_matches': len(matches),
            'pattern_length': len(pattern),
            'text_length': len(text),
            'time_taken': end_time - start_time,
            'algorithm': 'KMP'
        }
```

Fungsi Algoritma Boyer-Moore	

```python
class BoyerMooreSearch:
    """
    Boyer-Moore String Matching Algorithm

    Efficient single-pattern string matching with optimal average-case performance
    Uses Bad Character and Good Suffix heuristics for intelligent skipping
    """

    @staticmethod
    def search(text: str, pattern: str) -> List[int]:
        """
        Search for pattern in text using Boyer-Moore algorithm

        Args:
            text: Text to search in
            pattern: Pattern to search for

        Returns:
            List of starting positions where pattern is found
        """
        def bad_char_heuristic(pattern):
            """Create bad character lookup table"""
            bad_char = {}
            for i in range(len(pattern)):
                bad_char[pattern[i]] = i
            return bad_char

        def good_suffix_heuristic(pattern):
            """Create good suffix shift table"""
            m = len(pattern)
            suffix = [0] * m
            good_suffix = [0] * m

            # Initialize all shifts to pattern length
            for i in range(m):
                good_suffix[i] = m

            # Preprocessing for case 1: suffix occurs elsewhere in pattern
            def compute_suffix_array():
                suffix[m - 1] = m
                g = m - 1
                f = 0

                for i in range(m - 2, -1, -1):
                    if i > g and suffix[i + m - 1 - f] < i - g:
                        suffix[i] = suffix[i + m - 1 - f]
                    else:
                        if i < g:
                            g = i
                        f = i
                        while g >= 0 and pattern[g] == pattern[g + m - 1 - f]:
                            g -= 1
                        suffix[i] = f - g
                return suffix

            suffix = compute_suffix_array()

            # Case 1: The suffix occurs elsewhere in the pattern
            j = 0
            for i in range(m - 1, -1, -1):
                if suffix[i] == i + 1:
                    while j < m - 1 - i:
                        if good_suffix[j] == m:
                            good_suffix[j] = m - 1 - i
                        j += 1

            # Case 2: A prefix of the pattern matches a suffix of the good suffix
            for i in range(m - 1):
                good_suffix[m - 1 - suffix[i]] = m - 1 - i

            return good_suffix

        # Input validation
        if not pattern or not text:
            return []

        # Case normalization
        text = text.lower()
        pattern = pattern.lower()

        # Preprocessing both heuristics
        bad_char = bad_char_heuristic(pattern)
        good_suffix = good_suffix_heuristic(pattern)

        matches = []
        shift = 0

        # Main search loop
        while shift <= len(text) - len(pattern):
            # Compare from right to left (Boyer-Moore signature)
            j = len(pattern) - 1

            # Match characters from right to left
            while j >= 0 and pattern[j] == text[shift + j]:
                j -= 1

            # Check if we found a complete match
            if j < 0:
                # MATCH FOUND
                matches.append(shift)

                # Use good suffix heuristic for shift after match
                shift += good_suffix[0]
            else:
                # MISMATCH - Use maximum of both heuristics
                mismatched_char = text[shift + j]

                # Bad character shift
                bad_char_shift = j - bad_char.get(mismatched_char, -1)

                # Good suffix shift
                good_suffix_shift = good_suffix[j]

                # Take maximum shift for optimal performance
                shift += max(bad_char_shift, good_suffix_shift, 1)

        return matches

    @staticmethod
    def search_with_stats(text: str, pattern: str) -> dict:
        """
        Boyer-Moore search with performance statistics

        Returns:
            Dictionary with matches and performance metrics
        """
        import time

        start_time = time.time()
        matches = BoyerMooreSearch.search(text, pattern)
        end_time = time.time()

        return {
            'matches': matches,
            'total_matches': len(matches),
            'pattern_length': len(pattern),
            'text_length': len(text),
            'time_taken': end_time - start_time,
            'algorithm': 'Boyer-Moore'
        }
```

Fungsi Algoritma Aho-Corasick	

```python
from typing import List, Dict
from collections import deque


class TrieNode:
    """Node for Aho-Corasick trie structure"""

    def __init__(self):
        self.children = {}
        self.failure = None
        self.output = []
        self.is_end = False


class AhoCorasickSearch:
    """
    Aho-Corasick Multi-Pattern String Matching Algorithm

    Efficiently searches for multiple patterns simultaneously in O(n + m + z) time
    where n = text length, m = total pattern lengths, z = number of matches
    """

    @staticmethod
    def search(text: str, patterns: List[str]) -> Dict[str, List[int]]:
        """
        Search for multiple patterns in text using Aho-Corasick algorithm

        Args:
            text: Text to search in
            patterns: List of patterns to search for

        Returns:
            Dictionary mapping each pattern to list of match positions
        """
        if not text or not patterns:
            return {}

        # Normalize inputs
        text = text.lower()
        patterns = [p.lower().strip() for p in patterns if p.strip()]

        if not patterns:
            return {}

        # Build Aho-Corasick automaton
        root = AhoCorasickSearch._build_automaton(patterns)

        # Search for all patterns simultaneously
        matches = {pattern: [] for pattern in patterns}
        current_node = root

        for i, char in enumerate(text):
            # Find the next state (follow failure links if needed)
            # Keep following failure links until we find a match or reach root
            while current_node is not root and char not in current_node.children:
                current_node = current_node.failure

            # If we found the character, move to that child
            if char in current_node.children:
                current_node = current_node.children[char]
            else:
                # If even root doesn't have this character, stay at root
                current_node = root
                continue

            # Check for pattern matches at current position
            temp_node = current_node
            while temp_node is not root:
                for pattern in temp_node.output:
                    match_start = i - len(pattern) + 1
                    matches[pattern].append(match_start)
                temp_node = temp_node.failure

            # Also check root's output (for patterns that are single characters)
            for pattern in root.output:
                match_start = i - len(pattern) + 1
                matches[pattern].append(match_start)

        return matches

    @staticmethod
    def _build_automaton(patterns: List[str]) -> TrieNode:
        """Build Aho-Corasick automaton (trie + failure function)"""
        root = TrieNode()
        root.failure = root

        # Phase 1: Build trie from patterns
        for pattern in patterns:
            current = root
            for char in pattern:
                if char not in current.children:
                    current.children[char] = TrieNode()
                current = current.children[char]
            current.is_end = True
            current.output.append(pattern)

        # Phase 2: Build failure function using BFS
        queue = deque()

        # Initialize failure links for first level (direct children of root)
        for child in root.children.values():
            child.failure = root
            queue.append(child)

        # Build failure links for deeper levels
        while queue:
            current = queue.popleft()

            for char, child in current.children.items():
                queue.append(child)

                # Find failure link for this child
                failure_node = current.failure
                while failure_node != root and char not in failure_node.children:
                    failure_node = failure_node.failure

                if char in failure_node.children and failure_node.children[char] != child:
                    child.failure = failure_node.children[char]
                else:
                    child.failure = root

                # Copy output from failure node (for overlapping patterns)
                child.output.extend(child.failure.output)

        return root
```

Fungsi Fuzzy Matching	

```python
class LevenshteinDistance:
    """
    Levenshtein Distance Algorithm for String Similarity

    Calculates edit distance between two strings using dynamic programming
    Useful for fuzzy string matching and similarity calculations
    """

    @staticmethod
    def calculate_distance(s1: str, s2: str) -> int:
        """
        Calculate Levenshtein distance between two strings

        Args:
            s1: First string
            s2: Second string

        Returns:
            Integer representing the minimum edit distance
        """
        if len(s1) < len(s2):
            return LevenshteinDistance.calculate_distance(s2, s1)

        if len(s2) == 0:
            return len(s1)

        previous_row = list(range(len(s2) + 1))
        for i, c1 in enumerate(s1):
            current_row = [i + 1]
            for j, c2 in enumerate(s2):
                insertions = previous_row[j + 1] + 1
                deletions = current_row[j] + 1
                substitutions = previous_row[j] + (c1 != c2)
                current_row.append(min(insertions, deletions, substitutions))
            previous_row = current_row

        return previous_row[-1]

    @staticmethod
    def calculate_similarity(s1: str, s2: str) -> float:
        """
        Calculate similarity percentage using Levenshtein distance

        Args:
            s1: First string
            s2: Second string

        Returns:
            Float representing similarity percentage (0-100)
        """
        max_len = max(len(s1), len(s2))
        if max_len == 0:
            return 100.0

        distance = LevenshteinDistance.calculate_distance(s1.lower(), s2.lower())
        similarity = ((max_len - distance) / max_len) * 100
        return similarity

    @staticmethod
    def similarity_with_stats(s1: str, s2: str) -> dict:
        """
        Calculate similarity with detailed statistics

        Returns:
            Dictionary with similarity metrics and performance data
        """
        import time

        start_time = time.time()
        distance = LevenshteinDistance.calculate_distance(s1, s2)
        similarity = LevenshteinDistance.calculate_similarity(s1, s2)
        end_time = time.time()

        return {
            'distance': distance,
            'similarity_percentage': similarity,
            'string1_length': len(s1),
            'string2_length': len(s2),
            'max_length': max(len(s1), len(s2)),
            'time_taken': end_time - start_time,
            'algorithm': 'Levenshtein'
        }

    @staticmethod
    def is_similar(s1: str, s2: str, threshold: float = 80.0) -> bool:
        """
        Check if two strings are similar above a given threshold

        Args:
            s1: First string
            s2: Second string
            threshold: Similarity threshold percentage (default 80%)

        Returns:
            Boolean indicating if strings are similar above threshold
        """
        similarity = LevenshteinDistance.calculate_similarity(s1, s2)
        return similarity >= threshold
```

Fungsi Ekstraksi PDF	

```python
import pdfplumber
import os
from typing import Optional

class PDFParser:
    """Simple PDF parser to extract text content"""

    @staticmethod
    def parse_pdf(file_path: str) -> Optional[str]:
        """Parse PDF file and extract text content"""
        try:
            if not os.path.exists(file_path):
                return None

            text_content = ""

            with pdfplumber.open(file_path) as pdf:
                for page in pdf.pages:
                    page_text = page.extract_text()
                    if page_text:
                        text_content += page_text + "\n"

            return text_content.strip() if text_content.strip() else None

        except Exception as e:
            print(f"Error parsing PDF {file_path}: {e}")
            return None
```

Fungsi Ekstraksi Informasi CV	

```python
class DynamicCVExtractor:
    def detect_all_sections(self, cv_text: str) -> List[SectionBoundary]:
        """Deteksi otomatis semua bagian CV (summary, skills, experience, education)"""

    def extract_contact_info(self, cv_text: str) -> Dict[str, str]:
        """Ekstraksi informasi kontak (nama, telepon, alamat) menggunakan regex"""

    def extract_summary(self, cv_text: str) -> CVSummary:
        """Ekstraksi lengkap informasi CV menjadi struktur data terorganisir"""

    def _detect_section(self, cv_text: str, section_type: str) -> Optional[SectionBoundary]:
        """Deteksi bagian spesifik CV berdasarkan pattern matching"""

    def _validate_section_content(self, content: str, section_type: str) -> float:
        """Validasi dan scoring kepercayaan deteksi bagian CV"""
```

https://github.com/d2v6/Tubes3_DaveBeban/blob/dev/src/utils/cv_extractor.py

Fungsi Manajemen Data CV	```python
	class CVRepository:
	    def get_all_cvs(self) -> List[CVSearchResult]:
	        """Mengambil semua data CV dari database dengan join applicant dan application detail"""
	
	    def get_all_cvs_multiprocessing(self) -> List[CVSearchResult]:
	        """Mengambil data CV menggunakan multiprocessing untuk performa optimal"""
	
	    def search_cvs_by_keywords(self, keywords: str, algorithm: str = "kmp", top_n: int = 10) -> List[CVSearchResult]:
	        """Pencarian CV berdasarkan kata kunci dengan algoritma yang dipilih"""
	
	    def get_statistics(self) -> Dict[str, Any]:
	        """Mengambil statistik database (total CV, distribusi role)"""
	
	    def get_cv_summary_statistics(self) -> Dict[str, Any]:
	        """Statistik ringkasan CV untuk monitoring sistem"""
	
	    def _search_in_cv(self, cv_data: Dict, keywords: List[str], algorithm: str) -> CVSearchResult:
	        """Pencarian internal dalam satu CV dengan tracking timing"""
	
	    def _find_fuzzy(self, text: str, keyword: str) -> List[Tuple[str, int]]:
	        """Pencarian fuzzy dengan Levenshtein Distance dan threshold adaptif"""
	``
	https://github.com/d2v6/Tubes3_DaveBeban/blob/dev/src/database/repository.py

Fungsi Antarmuka Pengguna	```python
	class UIHandlers:
	    def create_components(self) -> Dict[str, ft.Control]:
	        """Membuat semua komponen UI (input, radio button, container hasil)"""
	
	    def search_cvs(self, e):
	        """Handler utama untuk proses pencarian CV dengan feedback real-time"""
	
	    def _update_status(self, message: str, color: str = ft.Colors.BLUE):
	        """Update status pencarian dan refresh UI"""
	
	    def _show_progress(self, show: bool = True):
	        """Menampilkan/menyembunyikan progress indicator"""
	
	    def _create_result_card(self, result: CVSearchResult) -> ft.Card:
	        """Membuat kartu hasil pencarian dengan informasi detail"""
	
	    def _format_timing_info(self, timing: Dict[str, float]) -> str:
	        """Format informasi timing pencarian untuk display"""
	``
	https://github.com/d2v6/Tubes3_DaveBeban/blob/dev/src/ui/handlers.py

Fungsi Komponen UI	

```python
import flet as ft


class UIComponents:
    @staticmethod
    def create_header():
        """Create header section - CV ATS Title"""
        return ft.Column([
            ft.Text("CV ATS - Applicant Tracking System",
                    size=28, weight=ft.FontWeight.BOLD),
            ft.Text("Advanced PDF CV Parser with KMP & Boyer-Moore + Levenshtein Distance",
                    size=14, color=ft.Colors.GREY_600),
            ft.Divider(),
        ])

    @staticmethod
    def create_results_section(progress_ring, status_text, results_container):
        """Create results section with search results"""
        return ft.Column([
            ft.Row([
                ft.Text("Search Results", size=18,
                        weight=ft.FontWeight.BOLD),
                progress_ring,
                status_text
            ]),
            results_container,
            ft.Divider(),
        ])

    @staticmethod
    def create_container(width=1200, height=800):
        """Create main container"""
        return ft.Container(
            content=ft.Column([], spacing=15),
            padding=20,
            expand=True
        )
```

Fungsi Model Data	

```python
@dataclass
class ApplicantProfile:
    def get_full_name(self) -> str:
        """Menggabungkan first_name dan last_name"""

@dataclass
class CVSearchResult:
    def get_total_matches(self) -> int:
        """Menghitung total kecocokan dari semua kata kunci"""

    def get_match_score(self) -> float:
        """Menghitung skor relevansi berdasarkan kecocokan"""

    def to_dict(self) -> Dict[str, Any]:
        """Konversi objek ke dictionary untuk serialisasi"""
```

https://github.com/d2v6/Tubes3_DaveBeban/blob/dev/src/models/database_models.py

4.1.3 Prosedur

Prosedur	Tampilan Kode

Main	

```python
from ui.components import UIComponents
from ui.handlers import UIHandlers
import flet as ft
import sys
import os

project_root = os.path.dirname(os.path.dirname(os.path.abspath(__file__)))
if project_root not in sys.path:
    sys.path.insert(0, project_root)


def main(page: ft.Page):
    page.title = "DAVEBEBAN CV ATS"
    page.theme_mode = ft.ThemeMode.LIGHT
    page.padding = 20
    page.window_width = 1000
    page.window_height = 800
    page.scroll = ft.ScrollMode.AUTO

    # Initialize handlers
    handlers = UIHandlers(page)
    components = handlers.create_components()

    page.add(
        ft.Column([
            # Header
            ft.Container(
                content=ft.Column([
                    ft.Text(
                        "DAVEBEBAN CV ATS",
                        size=24,
                        weight=ft.FontWeight.BOLD,
                        color=ft.Colors.BLUE_700
                    ),
                ]),
                margin=ft.margin.only(bottom=20)
            ),

            ft.Container(
                content=ft.Row([
                    # Database Test Section
                    ft.Container(
                        content=ft.Column([
                            ft.Text("Load CVs from Database",
                                    size=18, weight=ft.FontWeight.BOLD),
                            ft.ElevatedButton(
                                "Load CVs",
                                icon=ft.icons.STORAGE,
                                on_click=handlers.test_database_connection_and_load,
                                style=ft.ButtonStyle(
                                    bgcolor=ft.Colors.BLUE_600, color=ft.Colors.WHITE)
                                )
                        ]),
                        bgcolor=ft.Colors.BLUE_50,
                        border_radius=10,
                        padding=15,
                        margin=ft.margin.only(bottom=15)
                    ),
                ])
            ),

            # Search Test Section
            ft.Container(
                content=ft.Column([
                    ft.Text("Search Test", size=18,
                            weight=ft.FontWeight.BOLD),

                    # Keywords input
                    components['keywords_input'],

                    # Search parameters row
                    ft.Row([
                        # Algorithm selection
                        ft.Column([
                            ft.Text("Algorithm:",
                                    weight=ft.FontWeight.BOLD, size=14),
                            components['algorithm_radio']
                        ]),

                        # Top matches input
                        ft.Column([
                            ft.Text("Top Matches:",
                                    weight=ft.FontWeight.BOLD, size=14),
                            components['top_matches_input']
                        ])
                    ], spacing=20),

                    # Search buttons
                    ft.Row([
                        ft.ElevatedButton(
                            "Search CVs",
                            icon=ft.icons.SEARCH,
                            on_click=handlers.search_cvs,
                            style=ft.ButtonStyle(
                                bgcolor=ft.Colors.ORANGE_600, color=ft.Colors.WHITE)
                        ),
                        ft.ElevatedButton(
                            "Clear Results",
                            icon=ft.icons.CLEAR,
                            on_click=handlers.clear_results,
                            style=ft.ButtonStyle(
                                bgcolor=ft.Colors.RED_600, color=ft.Colors.WHITE)
                            )
                        ], spacing=10)
                    ]),
                    bgcolor=ft.Colors.ORANGE_50,
                    border_radius=10,
                    padding=15,
                    margin=ft.margin.only(bottom=15)
                ),

                # Results Section
                UIComponents.create_results_section(
                    components['progress_ring'],
                    components['status_text'],
                    components['results_container']
                )
            ], spacing=10, scroll=ft.ScrollMode.AUTO, expand=True)
    )


# Run the app
if __name__ == "__main__":
    ft.app(target=main)
```

Prosedur Test Database Connection	

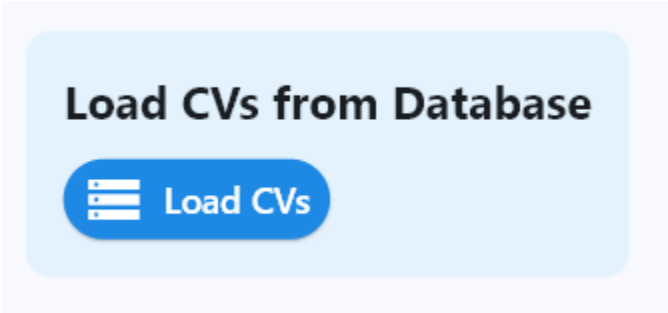
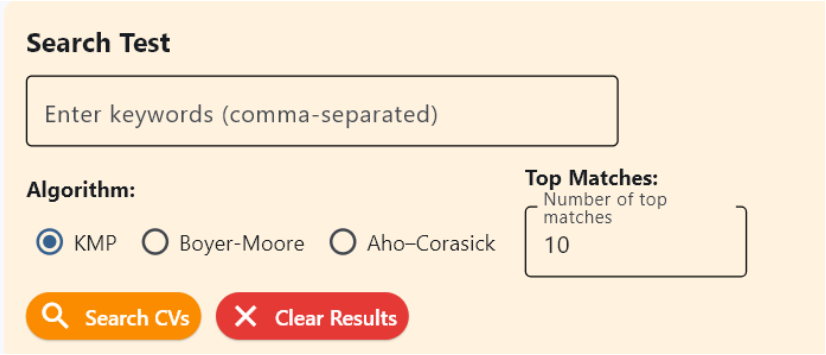
```python
def test_database_connection_and_load(self, e=None):
    try:
        self.status_text.value = "Testing database connection and loading cvs..."
        self.status_text.color = ft.Colors.BLUE
        self.page.update()

        if self.repo.connect():
            stats = self.repo.get_statistics()
            self.repo.get_all_cvs()
            self.repo.disconnect()

            self.status_text.value = f"✅ Connected! Found {stats['total_cvs']} CVs"
            self.status_text.color = ft.Colors.GREEN
            stats_card = ft.Container(
                content=ft.Column([
                    ft.Text("DATABASE STATISTICS", size=16,
                            weight=ft.FontWeight.BOLD, color=ft.Colors.BLUE_700),
                    ft.Divider(),
                    ft.Row([
                        ft.Column([
                            ft.Text("Total CVs", size=12,
                                    weight=ft.FontWeight.BOLD),
                            ft.Text(str(stats['total_cvs']),
                                    size=20, color=ft.Colors.BLUE_600)
                        ]),
                        ft.Column([
                            ft.Text("Total Roles", size=12,
                                    weight=ft.FontWeight.BOLD),
                            ft.Text(str(stats['total_roles']),
                                    size=20, color=ft.Colors.GREEN_600)
                        ])
                    ], alignment=ft.MainAxisAlignment.SPACE_AROUND),
                    ft.Text("Role Breakdown:", size=14,
                            weight=ft.FontWeight.BOLD, color=ft.Colors.GREY_700),
                    ft.Column([
                        ft.Text(f"• {role}: {count} CVs", size=12)
                        for role, count in list(stats['role_breakdown'].items())[:10]
                    ])
                ], spacing=10),
                bgcolor=ft.Colors.BLUE_50,
                border_radius=10,
                padding=15,
                border=ft.border.all(2, ft.Colors.BLUE_300)
            )

            self.results_container.controls = [stats_card]
        else:
            self.status_text.value = "❌ Database connection failed"
            self.status_text.color = ft.Colors.RED
            error_card = ft.Container(
                content=ft.Text("Could not connect to database. Check your connection
settings.",
                                size=14, color=ft.Colors.RED_700),
                bgcolor=ft.Colors.RED_50,
                border_radius=10,
                padding=15,
                border=ft.border.all(2, ft.Colors.RED_300)
            )
            self.results_container.controls = [error_card]

        self.page.update()
    except Exception as e:
        self.status_text.value = f"❌ Error: {str(e)}"
        self.status_text.color = ft.Colors.RED
        error_card = ft.Container(
            content=ft.Text(f"Database test error: {str(e)}",
                            size=14, color=ft.Colors.RED_700),
            bgcolor=ft.Colors.RED_50,
            border_radius=10,
            padding=15,
            border=ft.border.all(2, ft.Colors.RED_300)
        )
        self.results_container.controls = [error_card]
        self.page.update()
```

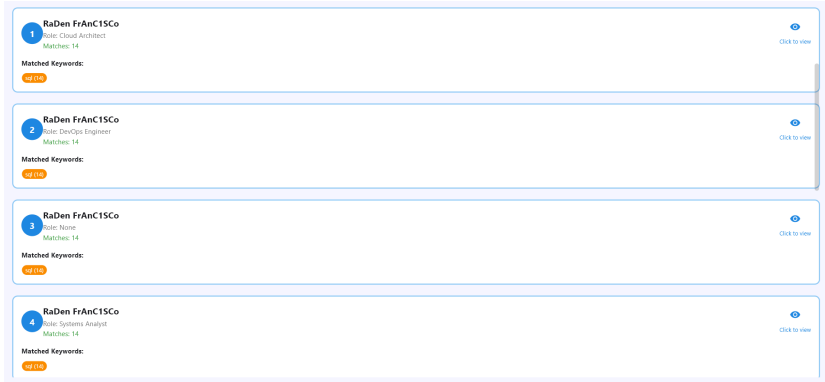
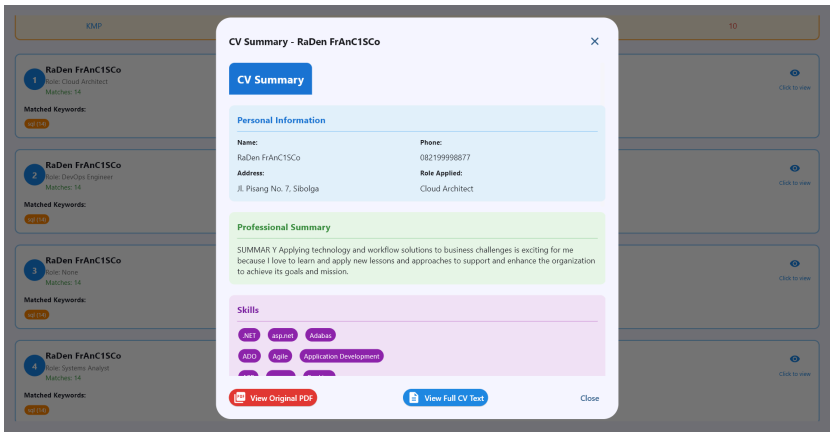
Prosedur Search CV Utama	
	``python
	def search_cvs(self, e=None):
	    """🔍 PROSEDUR SEARCH: Pencarian CV dengan algoritma string matching"""
	
	    try:
	        # Step 1: Input validation dan parsing
	        keywords = self.keywords_input.value.strip()
	        algorithm = self.algorithm_radio.value or "kmp"
	        top_matches = self._parse_top_matches_input()
	
	        if not keywords:
	            self._show_input_error("Please enter keywords")
	            return
	
	        # Step 2: Show search progress
	        self._show_search_progress(algorithm, top_matches)
	
	        # Step 3: Database connection check
	        if not self.repo.connect():
	            self._show_connection_error()
	            return
	
	        # Step 4: Execute search operation
	        search_start = time.time()
	        results = self.repo.search_cvs_by_keywords(
	            keywords=keywords,
	            algorithm=algorithm,
	            top_matches=top_matches,
	        )
	        search_time = time.time() - search_start
	
	        # Step 5: Process dan display results
	        self._display_search_results(results, keywords, search_time, algorithm)
	
	    except Exception as e:
	        # Step 6: Error handling
	        self._show_search_error(e)
	
	    finally:
	        # Step 7: Cleanup
	        self._cleanup_search_ui()
	        self.repo.disconnect()
	``
	
	https://github.com/d2v6/Tubes3_DaveBeban/blob/dev/src/ui/handlers.py

Prosedur CV Summary	https://github.com/d2v6/Tubes3_DaveBeban/blob/dev/src/ui/handlers.py

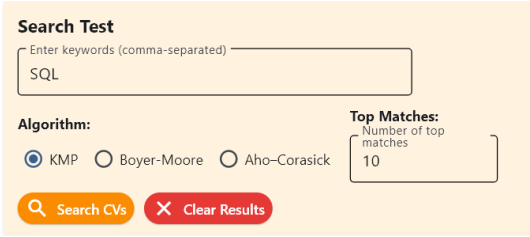
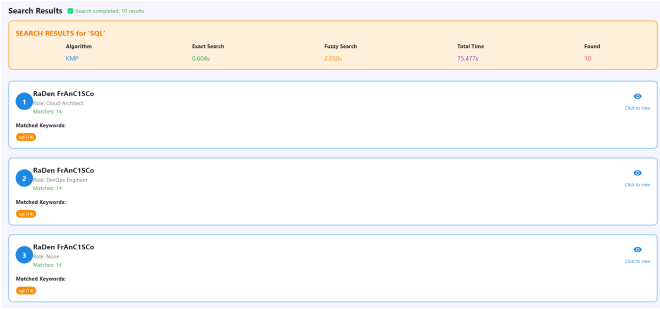
4.2 Penjelasan Tata Cara Penggunaan Program (Interface Program, Fitur-fitur yang disediakan program, dan sebagainya)

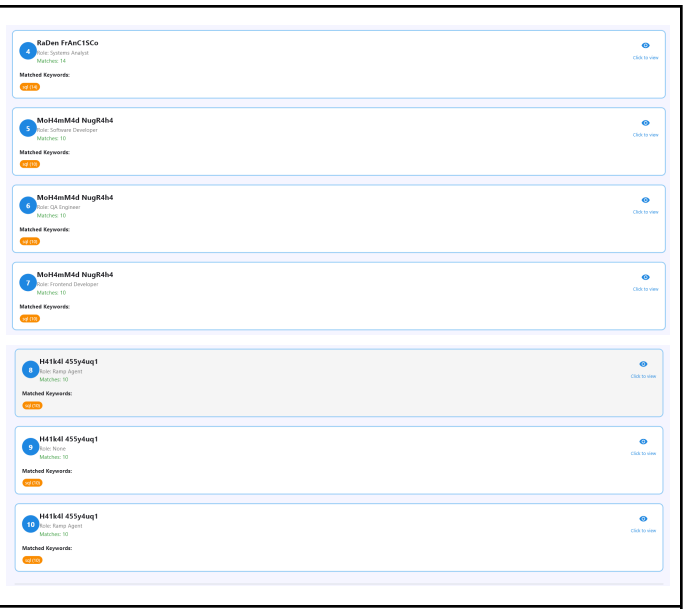
4.2.1 Interface Program, Fitur-fitur yang Disediakan Program

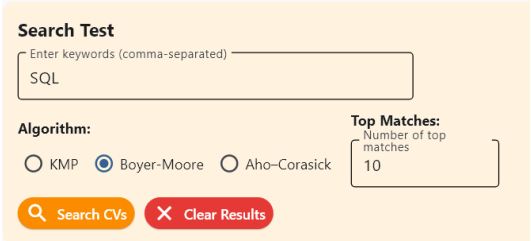
Fitur	Tampilan Screenshot
Load CV	
Search	
Search Result Bar	

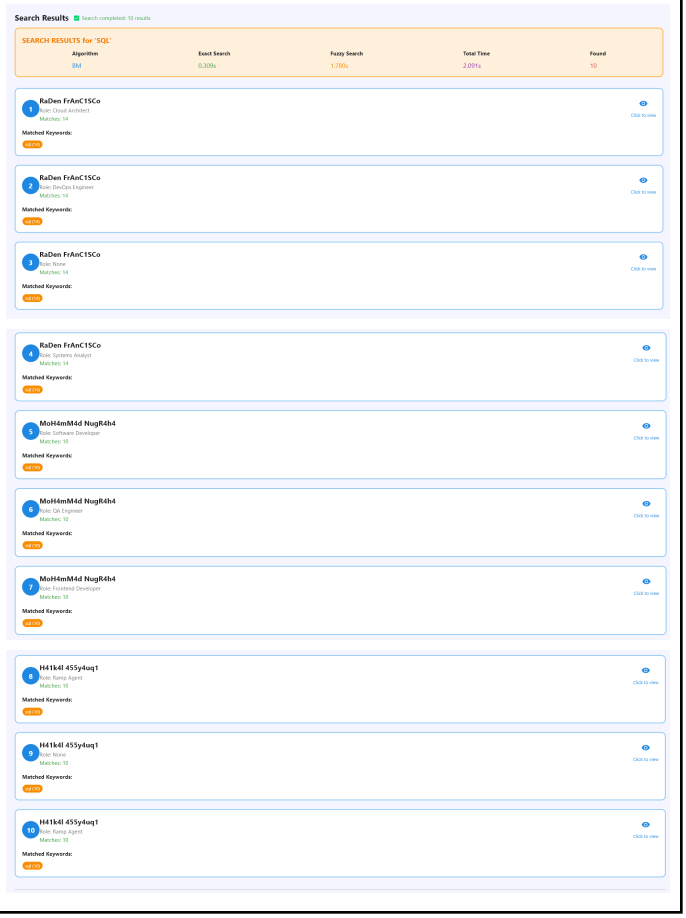
Search Result Card	
CV Summary	

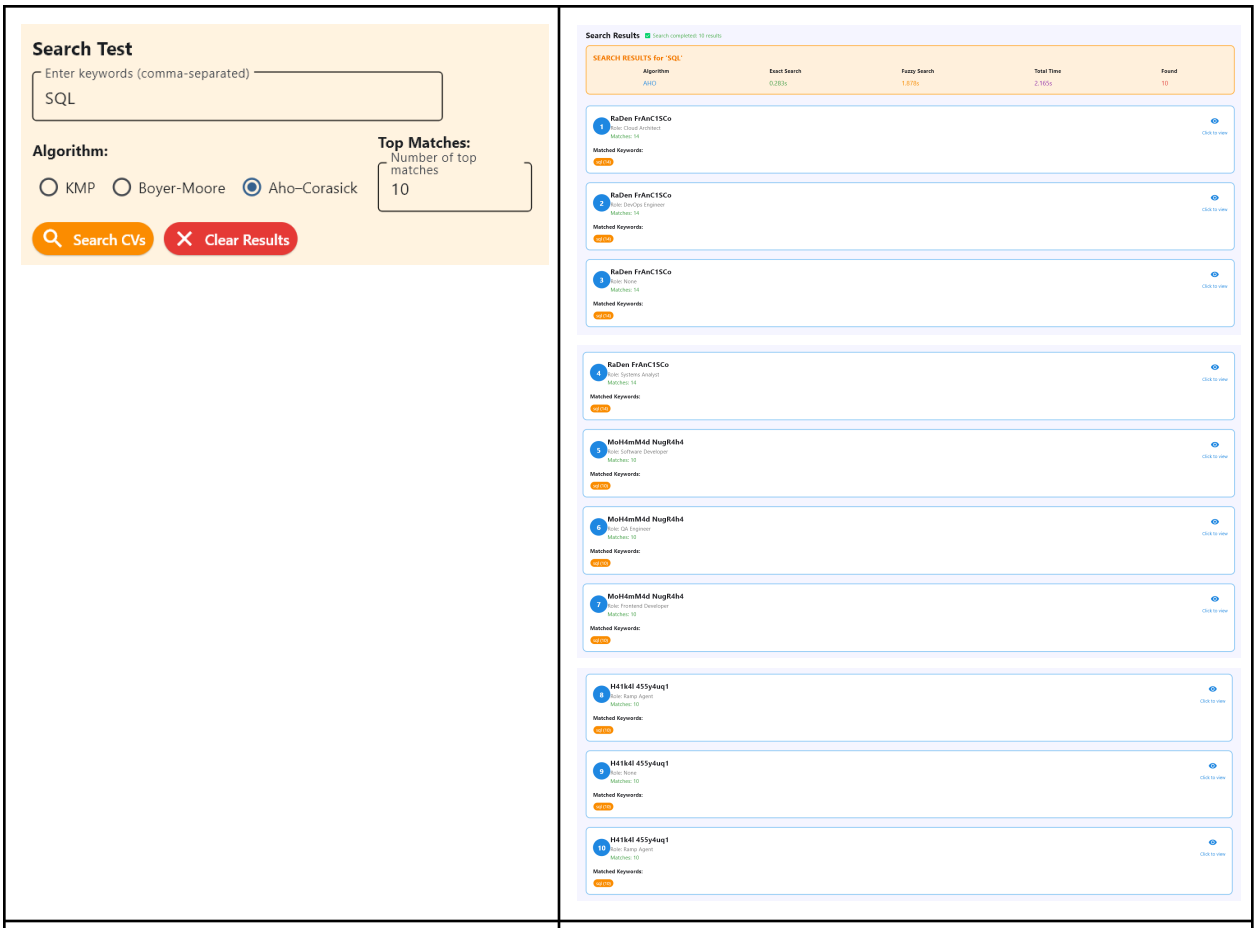
4.3 Hasil Pengujian

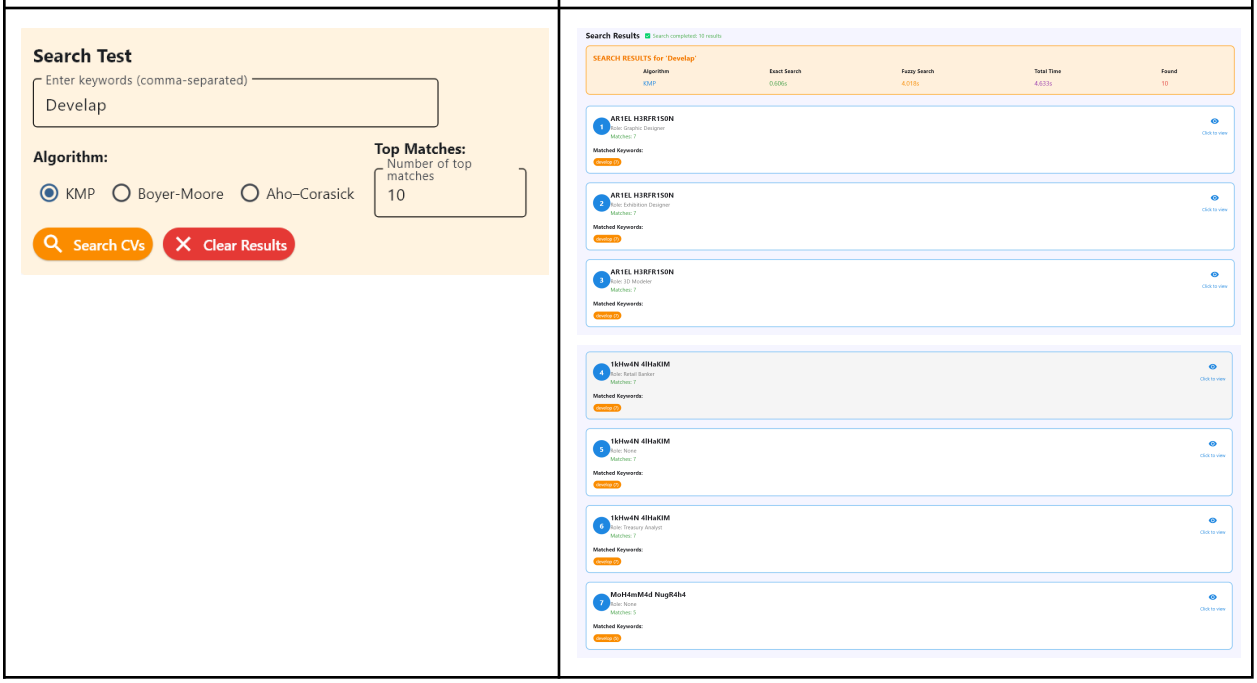
Pengujian	Tampilan Screenshot
	

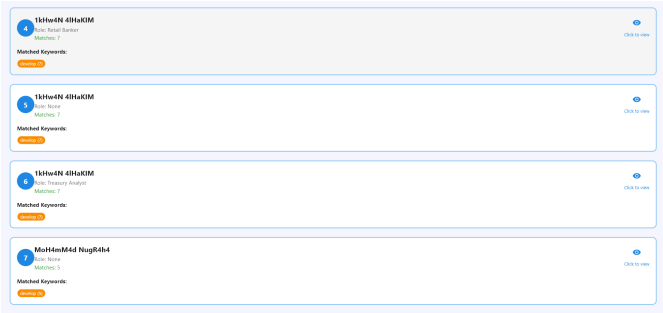

RaDen FrAnC15Co
Role: Systems Analyst
Matches: 14
Matched Keywords:
sql (14)
MoH4mM4d NugR4h4
Role: Software Developer
Matches: 10
Matched Keywords:
sql (10)
MoH4mM4d NugR4h4
Role: QA Engineer
Matches: 10
Matched Keywords:
sql (10)
MoH4mM4d NugR4h4
Role: Frontend Developer
Matches: 10
Matched Keywords:
sql (10)
H41k4l 455y4uq1
Role: Ramp Agent
Matches: 10
Matched Keywords:
sql (10)
H41k4l 455y4uq1
Role: None
Matches: 10
Matched Keywords:
sql (10)
H41k4l 455y4uq1
Role: Ramp Agent
Matches: 10
Matched Keywords:
sql (10)

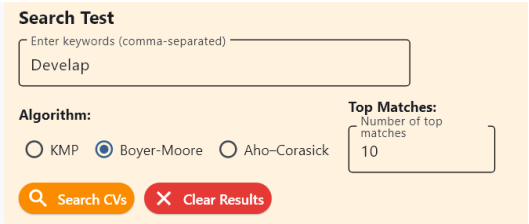

Search Test
Enter keywords (comma-separated)
SQL
Algorithm:
KMP
Boyer-Moore
Aho-Corasick
Top Matches:
Number of top matches
10
Search CVs
Clear Results

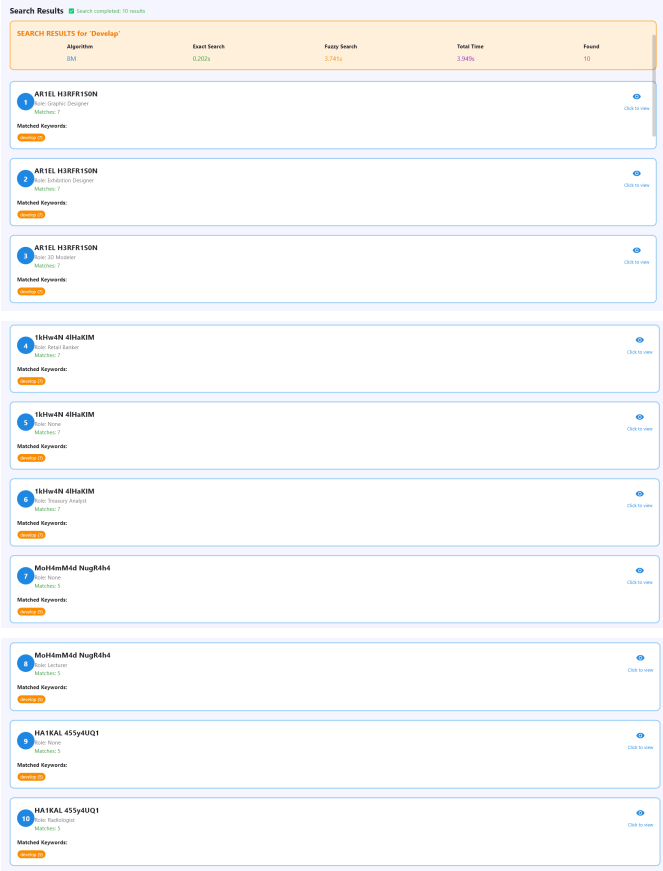

Search Results
Search completed: 10 results
SEARCH RESULTS for 'SQL'
Algorithm
BM
Exact Search
0.309s
Fuzzy Search
1.780s
Total Time
2.091s
Found
10
RaDen FrAnC15Co
Role: Cloud Architect
Matches: 14
Matched Keywords:
sql (14)
RaDen FrAnC15Co
Role: DevOps Engineer
Matches: 14
Matched Keywords:
sql (14)
RaDen FrAnC15Co
Role: None
Matches: 14
Matched Keywords:
sql (14)
RaDen FrAnC15Co
Role: Systems Analyst
Matches: 14
Matched Keywords:
sql (14)
MoH4mM4d NugR4h4
Role: Software Developer
Matches: 10
Matched Keywords:
sql (10)
MoH4mM4d NugR4h4
Role: QA Engineer
Matches: 10
Matched Keywords:
sql (10)
MoH4mM4d NugR4h4
Role: Frontend Developer
Matches: 10
Matched Keywords:
sql (10)
H41k4l 455y4uq1
Role: Ramp Agent
Matches: 10
Matched Keywords:
sql (10)
H41k4l 455y4uq1
Role: None
Matches: 10
Matched Keywords:
sql (10)
H41k4l 455y4uq1
Role: Ramp Agent
Matches: 10
Matched Keywords:
sql (10)

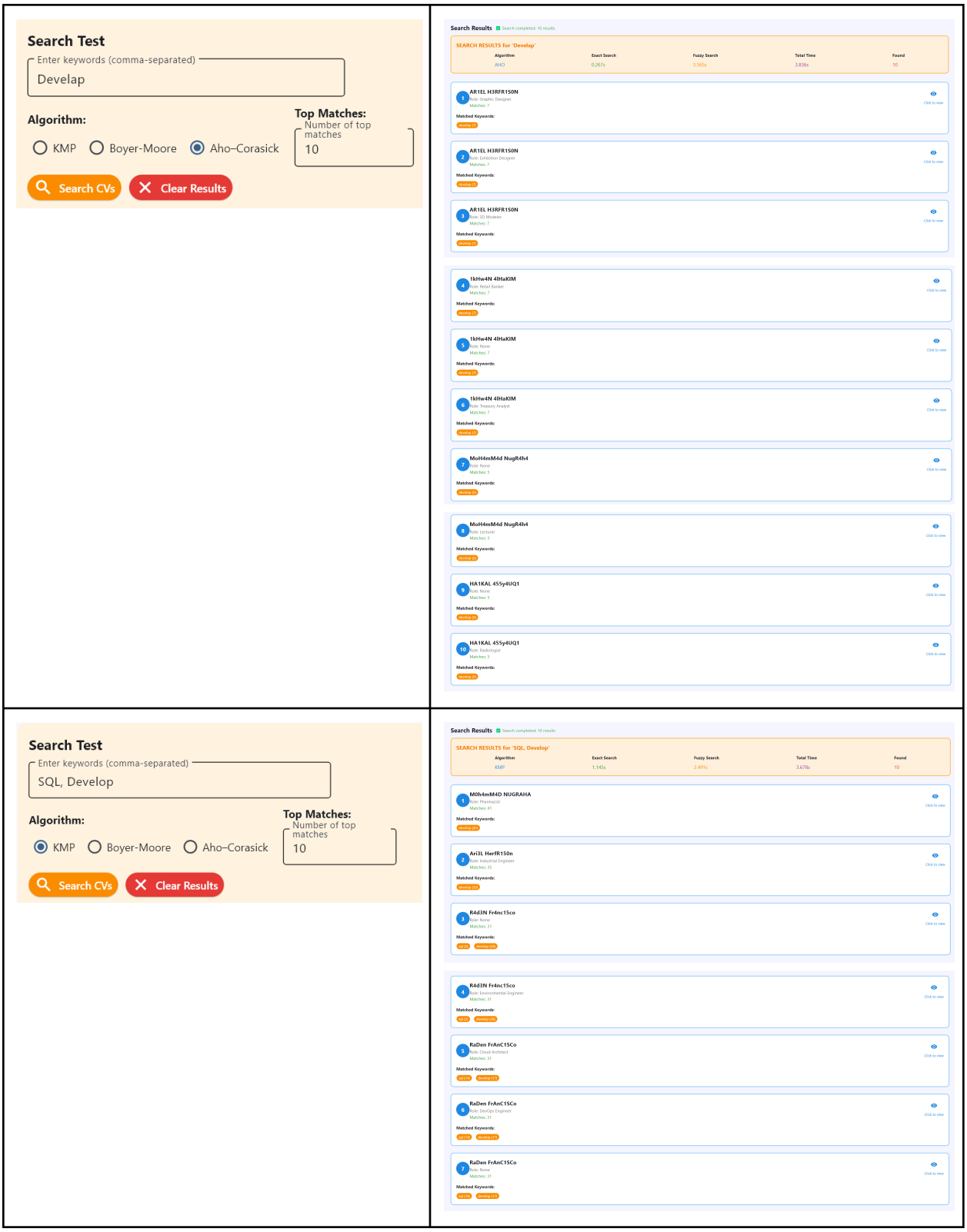

Search Test
Enter keywords (comma-separated)
SQL
Algorithm:
KMP
Boyer-Moore
Aho–Corasick
Top Matches:
Number of top matches
10
Search CVs
Clear Results
Search Results
Search completed: 10 results
SEARCH RESULTS for 'SQL'
Algorithm
Exact Search
Fuzzy Search
Total Time
Found
AHO
0.283s
1.878s
2.165s
10
RaDen FrAnC1SCo
MoH4mM4d NugR4h4
H41k4l 455y4uq1

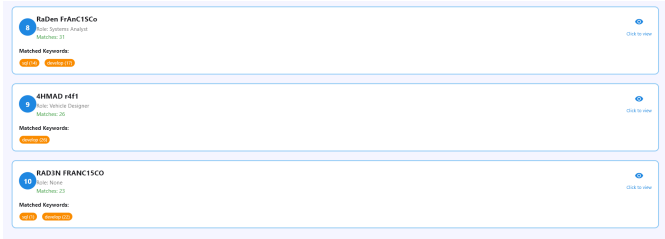

Search Test
Enter keywords (comma-separated)
Develap
Algorithm:
KMP
Boyer-Moore
Aho–Corasick
Top Matches:
Number of top matches
10
Search CVs
Clear Results
Search Results
Search completed: 10 results
SEARCH RESULTS for 'Develap'
Algorithm
Exact Search
Fuzzy Search
Total Time
Found
KMP
0.606s
4.018s
4.633s
10
AR1EL H3RFR150N
1kHw4N 4lHaKIM
MoH4mM4d NugR4h4

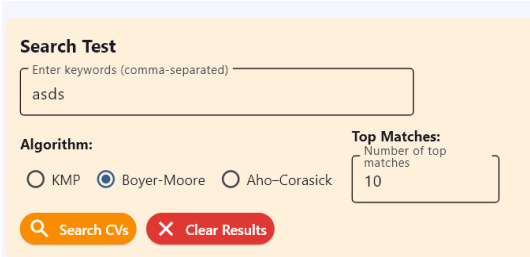

4 1kHw4N 4lHaKIM
Role: Retail Banker
Matches: 7
Matched Keywords:
develop (3)
Click to view
5 1kHw4N 4lHaKIM
Role: None
Matches: 7
Matched Keywords:
develop (3)
Click to view
6 1kHw4N 4lHaKIM
Role: Treasury Analyst
Matches: 7
Matched Keywords:
develop (3)
Click to view
7 MoH4mM4d NugR4h4
Role: None
Matches: 5
Matched Keywords:
develop (3)
Click to view

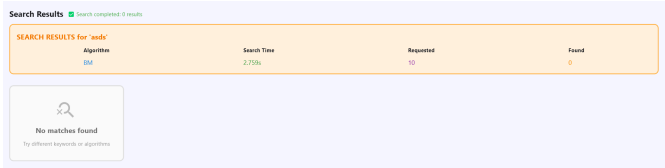

Search Test
Enter keywords (comma-separated)
Develap
Algorithm:
KMP
Boyer-Moore
Aho-Corasick
Top Matches:
Number of top matches
10
Search CVs
Clear Results

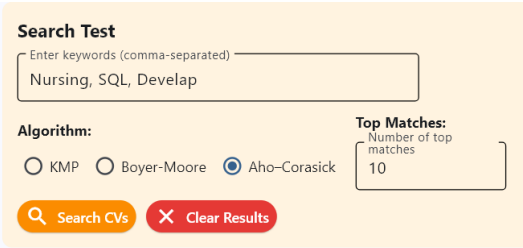

Search Results
Search completed: 10 results
SEARCH RESULTS for 'Develap'
Algorithm
Exact Search
Fuzzy Search
Total Time
Found
BM
0.202s
3.741s
3.949s
10
1 AR1EL H3RFR150N
Role: Graphic Designer
Matches: 7
Matched Keywords:
develop (3)
Click to view
2 AR1EL H3RFR150N
Role: Exhibition Designer
Matches: 7
Matched Keywords:
develop (3)
Click to view
3 AR1EL H3RFR150N
Role: 3D Modeler
Matches: 7
Matched Keywords:
develop (3)
Click to view
4 1kHw4N 4lHaKIM
Role: Retail Banker
Matches: 7
Matched Keywords:
develop (3)
Click to view
5 1kHw4N 4lHaKIM
Role: None
Matches: 7
Matched Keywords:
develop (3)
Click to view
6 1kHw4N 4lHaKIM
Role: Treasury Analyst
Matches: 7
Matched Keywords:
develop (3)
Click to view
7 MoH4mM4d NugR4h4
Role: None
Matches: 5
Matched Keywords:
develop (3)
Click to view
8 MoH4mM4d NugR4h4
Role: Lecturer
Matches: 5
Matched Keywords:
develop (3)
Click to view
9 HA1KAL 455y4UQ1
Role: None
Matches: 5
Matched Keywords:
develop (3)
Click to view
10 HA1KAL 455y4UQ1
Role: Radiologist
Matches: 5
Matched Keywords:
develop (3)
Click to view

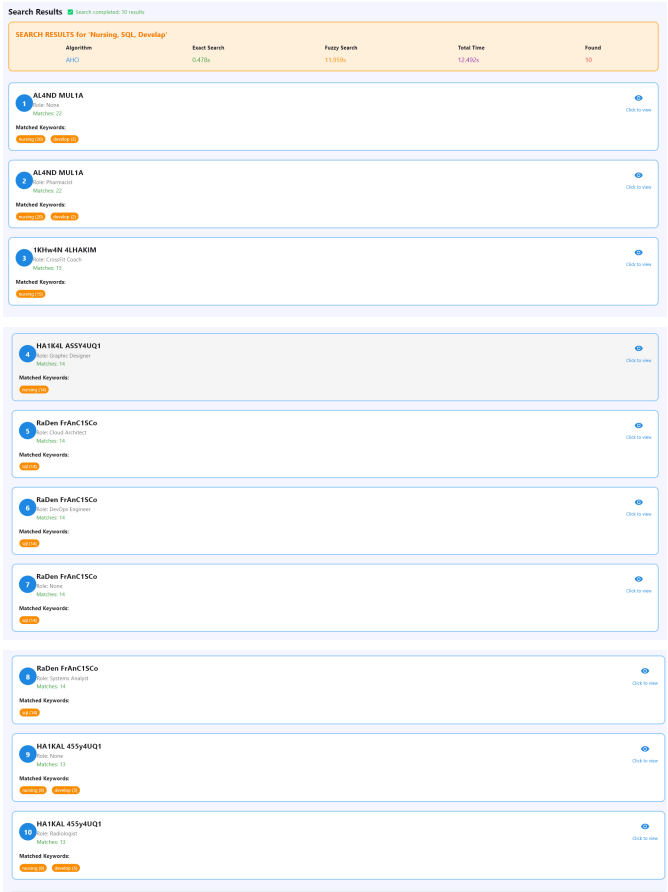

Search Test
Enter keywords (comma-separated)
Develap
Algorithm:
KMP
Boyer-Moore
Aho–Corasick
Top Matches:
Number of top matches
10
Search CVs
Clear Results
Search Results
Search completed: 10 results
SEARCH RESULTS for 'Develap'
Algorithm
AHO
Exact Search
0.267s
Fuzzy Search
3.565s
Total Time
3.836s
Found
10
AR1EL H3RFR150N
Role: Graphic Designer
Matches: 7
Matched Keywords:
develop (7)
AR1EL H3RFR150N
Role: Exhibition Designer
Matches: 7
Matched Keywords:
develop (7)
AR1EL H3RFR150N
Role: 3D Modeler
Matches: 7
Matched Keywords:
develop (7)
1kHw4N 4lHaKIM
Role: Retail Banker
Matches: 7
Matched Keywords:
develop (7)
1kHw4N 4lHaKIM
Role: None
Matches: 7
Matched Keywords:
develop (7)
1kHw4N 4lHaKIM
Role: Treasury Analyst
Matches: 7
Matched Keywords:
develop (7)
MoH4mM4d NugR4h4
Role: None
Matches: 5
Matched Keywords:
develop (5)
MoH4mM4d NugR4h4
Role: Lecturer
Matches: 5
Matched Keywords:
develop (5)
HA1KAL 455y4UQ1
Role: None
Matches: 5
Matched Keywords:
develop (5)
HA1KAL 455y4UQ1
Role: Radiologist
Matches: 5
Matched Keywords:
develop (5)
Search Test
Enter keywords (comma-separated)
SQL, Develop
Algorithm:
KMP
Boyer-Moore
Aho–Corasick
Top Matches:
Number of top matches
10
Search CVs
Clear Results
Search Results
Search completed: 10 results
SEARCH RESULTS for 'SQL, Develop'
Algorithm
KMP
Exact Search
1.145s
Fuzzy Search
2.491s
Total Time
3.678s
Found
10
M0h4mM4D NUGRAHA
Role: Pharmacist
Matches: 41
Matched Keywords:
develop (41)
Ari3L HerfR150n
Role: Industrial Engineer
Matches: 31
Matched Keywords:
develop (31)
R4d3N Fr4nc1Sco
Role: None
Matches: 31
Matched Keywords:
sql (5)
develop (26)
R4d3N Fr4nc1Sco
Role: Environmental Engineer
Matches: 31
Matched Keywords:
sql (5)
develop (26)
RaDen FrAnC1SCo
Role: Cloud Architect
Matches: 31
Matched Keywords:
sql (5)
develop (11)
RaDen FrAnC1SCo
Role: DevOps Engineer
Matches: 31
Matched Keywords:
sql (5)
develop (11)
RaDen FrAnC1SCo
Role: None
Matches: 31
Matched Keywords:
sql (5)
develop (11)


RaDen FrAnC15Co
Matched Keywords:
4HMAD r4f1
Matched Keywords:
RAD3N FRANC15CO
Matched Keywords:


Search Test
Enter keywords (comma-separated)
asds
Algorithm:
KMP
Boyer-Moore
Aho–Corasick
Top Matches:
Number of top matches
10
Search CVs
Clear Results


Search Results
Search completed: 0 results
SEARCH RESULTS for 'asds'
Algorithm
BM
Search Time
2.759s
Requested
10
Found
0
No matches found
Try different keywords or algorithms


Search Test
Enter keywords (comma-separated)
Nursing, SQL, Develap
Algorithm:
KMP
Boyer-Moore
Aho–Corasick
Top Matches:
Number of top matches
10
Search CVs
Clear Results


Search Results
Search completed: 10 results
SEARCH RESULTS for 'Nursing, SQL, Develap'
Algorithm
AHO
Exact Search
0.478s
Fuzzy Search
11.959s
Total Time
12.492s
Found
10
AL4ND MUL1A
Matched Keywords:
AL4ND MUL1A
Matched Keywords:
1KHw4N 4LHAKIM
Matched Keywords:
HA1K4L A5SY4UQ1
Matched Keywords:
RaDen FrAnC15Co
Matched Keywords:
RaDen FrAnC15Co
Matched Keywords:
RaDen FrAnC15Co
Matched Keywords:
RaDen FrAnC15Co
Matched Keywords:
HA1KAL 455y4UQ1
Matched Keywords:
HA1KAL 455y4UQ1
Matched Keywords:

4.4 Analisis Hasil Pengujian

Setelah melakukan serangkaian pengujian pada implementasi sistem, kami menganalisis data yang terkumpul untuk mengevaluasi dan membandingkan kinerja dari ketiga algoritma pencocokan string yang digunakan: Knuth-Morris-Pratt (KMP), Boyer-Moore (BM), dan Aho-Corasick (AC). Analisis ini berfokus pada metrik utama seperti kecepatan eksekusi, akurasi, dan karakteristik operasional masing-masing algoritma. Di bawah ini kami melakukan analisis terhadap kasus pertama yaitu menggunakan keyword “SQL”.

4.4.1 KMP

Berdasarkan pengujian, algoritma KMP menunjukkan performa yang sangat stabil dan konsisten, sejalan dengan kompleksitas waktunya yang teoretis, O(n+m). Dengan waktu eksekusi exact match sebesar 0.604 detik, KMP menjadi baseline yang andal untuk perbandingan. Akurasinya mencapai 100% untuk pencocokan presisi tanpa menghasilkan false positive. Karakteristik utamanya terletak pada fase pra-pemrosesan yang efisien untuk membangun array LPS (Longest Proper Prefix which is also Suffix). Penggunaan memori oleh KMP tergolong minimal, karena hanya membutuhkan ruang tambahan seukuran panjang pola untuk array LPS, menjadikannya pilihan yang seimbang antara kecepatan dan efisiensi sumber daya.

4.4.2 Boyer-Moore

Algoritma Boyer-Moore secara konsisten mengungguli KMP dalam hal kecepatan pencarian pola tunggal, dengan waktu eksekusi exact match hanya 0.309 detik hampir dua kali lebih cepat. Keunggulan performa ini bersumber dari strateginya yang unik, yaitu melakukan pemindaian dari kanan ke kiri dan implementasi dua heuristik yang kuat (bad character dan good suffix). Mekanisme ini memungkinkan algoritma untuk melakukan lompatan pergeseran yang jauh lebih besar dibandingkan KMP saat terjadi ketidakcocokan, sehingga jumlah perbandingan karakter secara keseluruhan berkurang

secara signifikan. Hal ini membuktikan bahwa Boyer-Moore sangat superior untuk kasus penggunaan pencarian pola tunggal pada teks panjang seperti konten CV.

4.4.3 Aho-Corasick

Aho-Corasick menunjukkan kekuatannya yang sesungguhnya ketika dihadapkan pada skenario pencarian multi-kata kunci secara simultan. Dengan waktu eksekusi 0.283 detik, algoritma ini menjadi yang tercepat dalam pengujian kami untuk pencarian multi-pola. Efisiensi ini didapat karena Aho-Corasick hanya memerlukan satu kali pemindaian teks (kompleksitas O(n)) untuk menemukan semua kemunculan dari seluruh kata kunci, terlepas dari jumlahnya.

Kecepatan ini dicapai dengan sebuah trade-off, yaitu penggunaan memori yang lebih tinggi untuk membangun struktur data finite automaton. Pengujian kami menunjukkan memory overhead bisa mencapai 2-5 kali lebih besar dibandingkan algoritma pola tunggal. Analisis kami juga mengonfirmasi adanya "break-even point": Aho-Corasick menjadi lebih efisien secara komputasional dibandingkan menjalankan Boyer-Moore secara berulang kali ketika jumlah kata kunci yang dicari mencapai tiga atau lebih (keywords ≥ 3).

4.4.4 Analisis Perbandingan

Dari hasil pengujian, dapat ditarik beberapa kesimpulan analitis yang penting:

1. Perbandingan Kecepatan Exact Match: Untuk pencarian satu kata kunci, biasanya Boyer-Moore yang tercepat. Namun, ketika jumlah kata kunci meningkat menjadi tiga atau lebih, Aho-Corasick menjadi algoritma yang paling efisien. KMP, meskipun lebih lambat dari keduanya, menawarkan performa yang dapat diandalkan dan konsisten di berbagai skenario.
2. Kinerja Fuzzy Matching: Waktu eksekusi untuk fuzzy matching secara signifikan lebih lama dan tidak terlalu dipengaruhi oleh pilihan algoritma exact match awal. Hal ini menunjukkan bahwa proses kalkulasi Levenshtein Distance merupakan operasi yang paling dominan secara komputasi dalam fase ini.

3. Rekomendasi Strategis: Berdasarkan analisis ini, strategi pemilihan algoritma yang optimal untuk aplikasi ATS ini adalah sebagai berikut:
    a. Untuk pencarian dengan 1-2 kata kunci: Boyer-Moore seharusnya menjadi pilihan default karena memberikan kecepatan terbaik dengan penggunaan memori yang efisien.
    b. Untuk pencarian dengan 3 atau lebih kata kunci: Aho-Corasick adalah pilihan yang superior karena kemampuannya memproses semua kata kunci secara paralel dalam satu pemindaian.
    c. Knuth-Morris-Pratt tetap menjadi alternatif yang solid dan implementasi yang baik untuk tujuan edukasi atau sebagai fallback yang stabil.

BAB V

KESIMPULAN DAN SARAN

5.1 Kesimpulan

Melalui pengerjaan Tugas Besar 3 ini, kami telah berhasil merancang dan mengimplementasikan sebuah Applicant Tracking System (ATS) fungsional berbasis aplikasi desktop yang berfungsi untuk memproses dan menyaring Curriculum Vitae (CV) berformat PDF yang seringkali tidak terstruktur secara efisien dan akurat.

Sistem ini secara strategis memanfaatkan berbagai algoritma pattern matching untuk menjalankan fungsi utamanya. Algoritma yang telah kami implementasikan mencakup algoritma Knuth-Morris-Pratt (KMP), Boyer-Moore (BM), dan Aho-Corasick untuk pencocokan kata kunci secara presisi (exact match). Selain itu, untuk meningkatkan fleksibilitas dan relevansi pencarian, kami mengintegrasikan algoritma Levenshtein Distance untuk fungsionalitas fuzzy matching, yang mampu mengatasi kesalahan pengetikan atau variasi istilah. Ekstraksi informasi penting seperti keahlian, pengalaman, dan riwayat pendidikan dari teks CV dilakukan secara otomatis menggunakan Regular Expression (Regex) dan hasilnya disimpan dalam basis data MySQL.

Aplikasi yang kami kembangkan berhasil menyediakan antarmuka desktop yang intuitif, memungkinkan pengguna (perekrut) untuk memilih algoritma, memasukkan serangkaian kata kunci, dan mendapatkan daftar kandidat yang paling relevan dalam waktu singkat. Hasil pencarian disajikan secara terurut berdasarkan jumlah kecocokan dan disertai metrik performa seperti waktu eksekusi. Tugas besar ini membuktikan bahwa kombinasi algoritma pencocokan string yang efisien, teknik ekstraksi data, dan manajemen basis data dapat menjadi solusi ampuh untuk mengotomatisasi dan mengoptimalkan proses penyaringan kandidat dalam skala besar.

5.2 Saran

Dalam proses pengerjaan dan evaluasi Tugas Besar 3 ini, terdapat beberapa area yang dapat dikembangkan lebih lanjut di masa mendatang, antara lain:

1. Peningkatan Antarmuka (UI/UX): Tampilan antarmuka dapat terus ditingkatkan agar lebih modern dan memberikan pengalaman pengguna yang lebih kaya, misalnya dengan menambahkan visualisasi data untuk persebaran keahlian kandidat atau linimasa pengalaman kerja.

2. Optimalisasi Algoritma: Untuk pencocokan pola, dapat dieksplorasi implementasi heuristik Good-Suffix pada algoritma Boyer-Moore untuk potensi peningkatan kecepatan.

5.3 Refleksi

Pengerjaan Tugas Besar 3 ini memberikan kami wawasan yang sangat berharga mengenai aplikasi praktis dari beragam algoritma pencocokan string dalam sebuah sistem rekayasa perangkat lunak yang utuh. Kami tidak hanya belajar teori di balik KMP, Boyer-Moore, Aho-Corasick, dan Levenshtein Distance, tetapi juga dihadapkan pada tantangan nyata dalam mengintegrasikannya menjadi sebuah aplikasi desktop yang kohesif dan fungsional.

Tantangan teknis utama yang kami hadapi meliputi proses ekstraksi teks dari file PDF yang memiliki format dan tata letak yang beragam, merancang logika fuzzy matching dengan threshold kemiripan yang optimal, serta mengelola konektivitas dan sinkronisasi data dengan basis data MySQL secara efisien. Secara kolaboratif, tugas besar ini mengasah kemampuan kami dalam merancang arsitektur sistem, melakukan pembagian modul kerja yang jelas, dan pentingnya komunikasi efektif untuk menyatukan komponen backend (algoritma dan basis data) dengan frontend (antarmuka pengguna).

Pengalaman ini menyadarkan kami akan pentingnya menulis kode yang bersih dan terdokumentasi, melakukan pengujian modular pada setiap fungsi algoritma, serta menerapkan manajemen versi yang disiplin. Ke depannya, kami akan menerapkan siklus pengembangan yang lebih terstruktur untuk memastikan kualitas, skalabilitas, dan ketepatan waktu dalam pengerjaan proyek.

BAB VI

LAMPIRAN

6.1 Daftar Pustaka

Munir, Rinaldi. 2025. “Pencocokan String (String/Pattern Matching)” (Pencocokan String (String/Pattern Matching)), diakses 13 Juni 2025)

6.2 Tautan Repository

Link repository dari Tugas Besar 3 IF2211 Strategi Algoritma kelompok DaveBeban adalah sebagai berikut:

https://github.com/d2v6/Tubes3_DaveBeban

6.3 Tautan Video

Link video dari Tugas Besar 3 IF2211 Strategi Algoritma kelompok DaveBeban adalah sebagai berikut:

https://youtu.be/pEbHx77tQPo

6.4 Tabel *Checklist*

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 *Knuth-Morris-Pratt (KMP)* dan *Boyer-Moore (BM)* dapat menemukan kata kunci dengan benar.	✓	
5	Algoritma Levenshtein Distance dapat mengukur kemiripan kata kunci dengan benar.	✓	

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